Bilkent University
Department of Computer Engineering

CS319 Term Project

Project short-name: BilBoard

Section 3

Group 3a

Design Report Iteration 2

Project Group Member Names:

İlke Doğan	(21702215)
Hacı Çakın	(21802641)
Metehan Saçakçı	(21802788)
Muhammet Abdullah Koç	(21802832)
Aslı Dinç	(21802527)

Instructor: Eray Tüzün

18 December 2021

Table of Contents

1 Introduction

1.1 Purpose of the system

This application provides the users with the opportunity to access and communicate with all the clubs established within the framework of the school and enables them to follow the developments organized by the clubs in an interactive, instantaneous manner. Besides being an application where the activities, projects, and participants of the clubs are kept, it is aimed and offered to provide ease of use to all actors in the club network by offering specific pages for each actor of the application. This application has adopted the principle of appealing to all actors with unique features such as users' access to events and related information, access to club management's accounts via the website application, event status arrangements, and event authorisation and editing page that the student office directly manages.

1.2 Design goals

This application offers a user-friendly, reliable, and secure interface with functionality, maintainability, and scalability to appeal to all kinds of users.

1.2.1 Usability

It is a priority to accommodate a user-friendly interface so as to provide multi-dimensional ease of operation and intelligibility to various users. Easy-to-read font, annotated button/text field service, a comfortable transition between tabs, and straightforward access by everyone are offered. Being a single-page application will ensure that users are responsive when browsing the application.

1.2.2 Rapid Development

It is planned to develop the project within three months. This process should be managed meticulously and the task distribution should be given appropriately. Additionally, rapid prototyping of the product, which focuses on the feedback of the target user group and adopts agile software, is aimed.

1.2.3 Functionality

It is intended to be practiced for numerous user types. Users can perform certain functional actions within the application. The application has an extensible feature in itself and more functionality can be added as needed in the later stages of the project. Restriction states that customize specific tasks of more than one user and that can limit access to confidential information contribute to the functionality.

2 High-level Software Architecture

2.1 Subsystem Decomposition

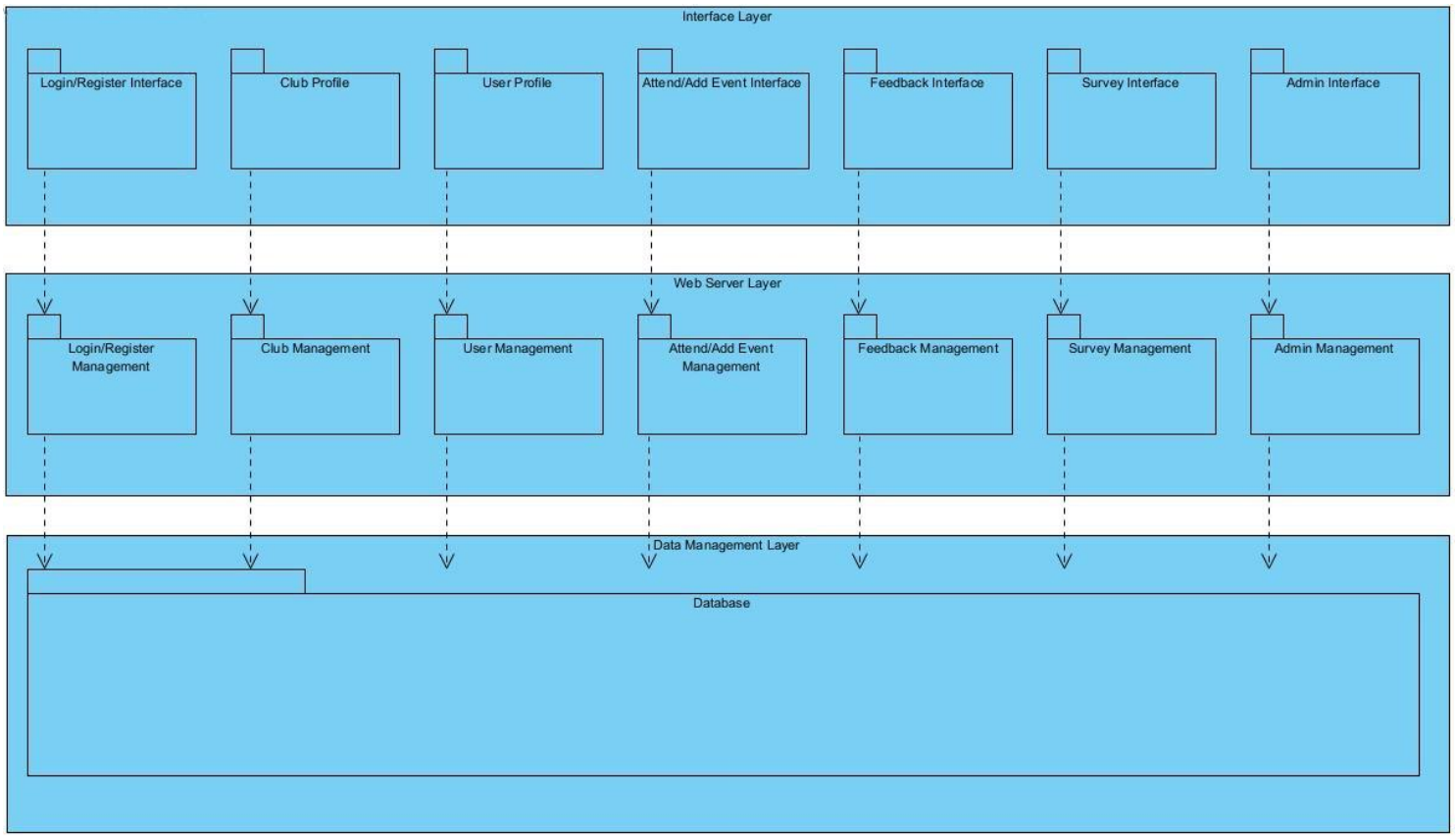

Fig. 1. Subsystem Decomposition.

To create an area to maintain our project, we decided to use three-layer architecture.

First of all, the UI Interface Layer is designed as a boundary area that contains our web application pages. It should be seen that every user will interact with our application using these pages. The main reason for this approach is to make our application much more usable.

Secondly, the Web Server Layer is designed to carry out all of the backend operations. The diagram also shows that there are different systems to manage different operations in our application.

Thirdly, the Data Management Layer is designed to communicate with the database to store all necessary data.

Login/Register Subsystem: This subsystem is designed to handle operations related with login/register procedure such as login, register, forget password that requires authorization.

Club Subsystem: This subsystem is designed to handle operations that are related with management of the club such as adjusting sponsor, club hierarchy, club profile photo, etc. that requires an authorization.

User Subsystem: This subsystem is designed to handle operations that are related with user management such as adjusting GE250/GE251 course, editing profile settings, etc. that requires an authorization.

Attend/Add Event Subsystem: This subsystem is designed to handle operations that are related to event objects such as creation of event, attending of event, editing attributes of the event, changing visibility of the event, etc. that requires an authorization.

Feedback Subsystem: This subsystem is designed to handle operations that are related to the feedback feature which is basically given to clubs with specifying the content. This subsystem also requires an authorization.

Survey Subsystem: This subsystem is designed to handle operations that are related to adding questions and answers to a survey object. This subsystem also requires an authorization.

Admin Subsystem: This subsystem is designed to handle operations of admin such as create/delete a club, assign president/advisor to a club that requires an authorization.

2.2 Hardware/Software Mapping

Since we are developing a project to run on the web, our project does not require any special hardware components. However, it should be underlined that the system

where our project will be run should handle standard web browsers. Also, we are expecting to run our project on Google Chrome, Mozilla Firefox, Internet Explorer, Opera and Safari with the latest updated version. We are mainly aiming that our project should work perfectly on personal computers. That is why standard components of computers such as a mouse, keyboard, monitor are essential in order to use our application.

The minimum requirements of our server that will host BilBoard depend on three critical attributes. The first attribute is traffic. If the website is used by more people, more RAM will be used [1]. The second attribute is the content management system. That is, the RAM depends on the host that we will upload to our website [1]. The last attribute is online applications. Because our application is online, its RAM usage is more [2]. Considering these requirements, 2GB RAM will be enough. [2]

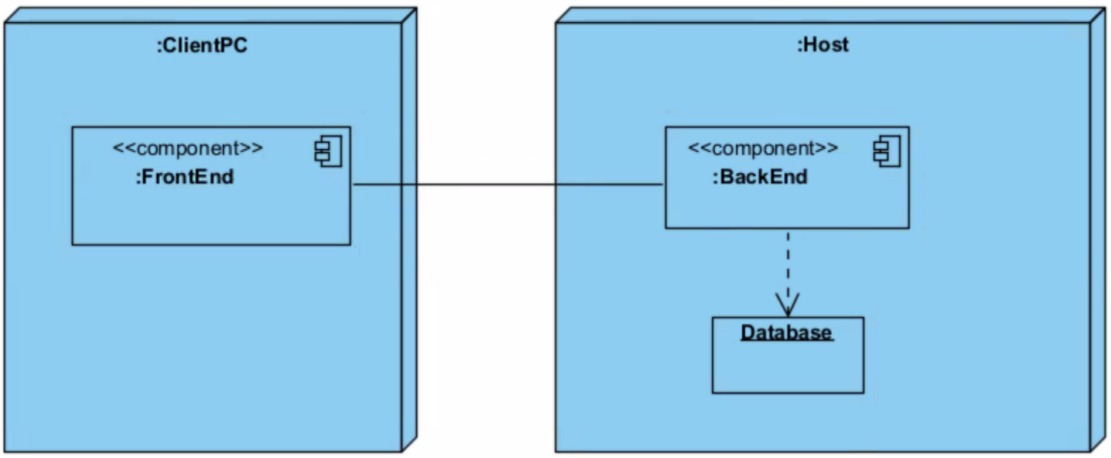

Fig. 2. Deployment Diagram.

2.3 Persistent Data Management

Since our application is a web application, we decided to use cloud options, not local ones. At this point, we had two options that we could prefer. One of them was SQL(relational databases), the other one was NoSQL(not relational databases). Because none of us knows Graph based databases, we haven't considered it. Relational databases are suitable when the system object has lots of many-to-many relationships. However, NoSQL databases are a better option when we do not have lots of queries and relationships. In NoSQL, lots of the properties of the objects are

embedded in a single entity object. Unlike this, SQL has rows and objects construct relations via their ids. Performing more queries makes SQL a better option for us. Moreover, because our data is structured, SQL takes advantage. If our schemas or models were not exact, NoSQL would be a good alternative. As a technology, we chose to use MySQL because members of our team are familiar with it. In our system, we will store some sensitive data related to users such as name, surname, email, id, and encrypted password. We also save the required information to trace object relations and activities. Detailed mapping related to storing can be found in the data management layer.

2.4 Access Control and Security

To achieve security within the Bilkent system, Bilkent offers the option to register users only in Bilkent by accepting Bilkent emails. For the sake of security, the passwords of users will be encrypted. All the users' requests which will be handled on the client-side will be authorized by using JSON Web Token(JWT)[3]. Each JWT will be created when the user logs in and will be valid for 1 hour. Also, whole requests which are not originated from the front-end will be responded with on CORS error [4]. All the passwords must be at least 8 characters.

	Student/Academic	Member	Active Member	Board Member	Advisor	President	Administrative Assistant	Admin
Login	X	X	X	X	X	X	X	X
SignUp	X							
Forgot Password	X	X	X	X	X	X	X	X
Confirm Password	X	X	X	X	X	X	X	X
View Clubs	X	X	X	X	X	X		X
View Events	X	X	X	X	X	X		
Enroll Clubs	X	X	X	X	X	X		
Enroll Events	X	X	X	X	X	X		
View Event Location Requests							X	
Accept/Decline Event Location Requests							X	

Create Event Location Request				X	X	X		
Vote Survey		X	X	X	X	X		
Vote Survey on Election			X	X	X	X		
Accept/ Decline Club Enroll Request				X	X	X		
Remove Member From Club				X	X	X		
Create Event				X	X	X		
Create Survey				X	X	X		
Update Event Visibility				X	X	X		
Update Survey Visibility				X	X	X		
View Members				X	X	X		
View Board Members				X	X	X		
View Club Sponsorship	X	X	X	X	X	X		
Add /Delete Sponsorship				X	X	X		
Enter Event Code	X	X	X	X	X	X		
Assign a Classroom							X	
Add/Delete Buildings							X	
Add/Delete Classroom							X	
Add/Delete Timeslot							X	
Add/Delete Board Member						X		
Add/Delete Club								X
Assign President								X
Assign Advisor								X

Fig. 3. Access Matrix.

2.5 Boundary Conditions

2.5.1 Initialization

Because our application is designed as a web application, it will be run on a web server at any time, except the server shuts down the application. When a system is deployed to AWS, it is automatically started by aws tools. Therefore, to start an application, uploading Jar build to AWS is enough. That is why if a user with an internet connection is found on the application's database, he/she will be allowed to the main screen of the program. It is also allowed that users can join the application from different devices.

2.5.2 Termination

Our application will be terminated in the case that any subsystem crashes. The main reason for this approach is to prevent any problematic operation from happening when a subsystem is down. Also, before the termination process starts all of the last saved data will be saved to the database to prevent data loss.

2.5.3 Failure

Paas products of AWS(open form) particularly [5], Elastic Beanstalk [6] and RDS(Relational Database System) [7] can be used to track and record failures. Also, some products can restart the entire environment.

There is also the possibility that failure might not be resolved with restart operation. In this case, the developer will be notified. Our application service layer has an HTTP response implementation for bugs that come from problematic operations.

3 Low-Level Design

3.1 Object Design Trade-offs

Functionality vs Usability: Our application can be used by students, student club board members, administrative assistants, and instructors. Also, many functions are unique for these user types. This makes the application functional. On the other hand, the application and interface are easy to use. We tried to keep a balance between functionality and usability. However, because the application focuses on students' use, we focused more on usability.

Rapid development vs Functionality: Because of time limitations on the development of the application, rapid development is one of the main concerns in our application. This time constraint causes the application to be less functional.

Security vs Usability: When a user logs in to the application, unless the user signs out, the session remains active for one hour. This increases usability but security decreases as there is no need to log in again in one hour.

3.2 Final Object Design

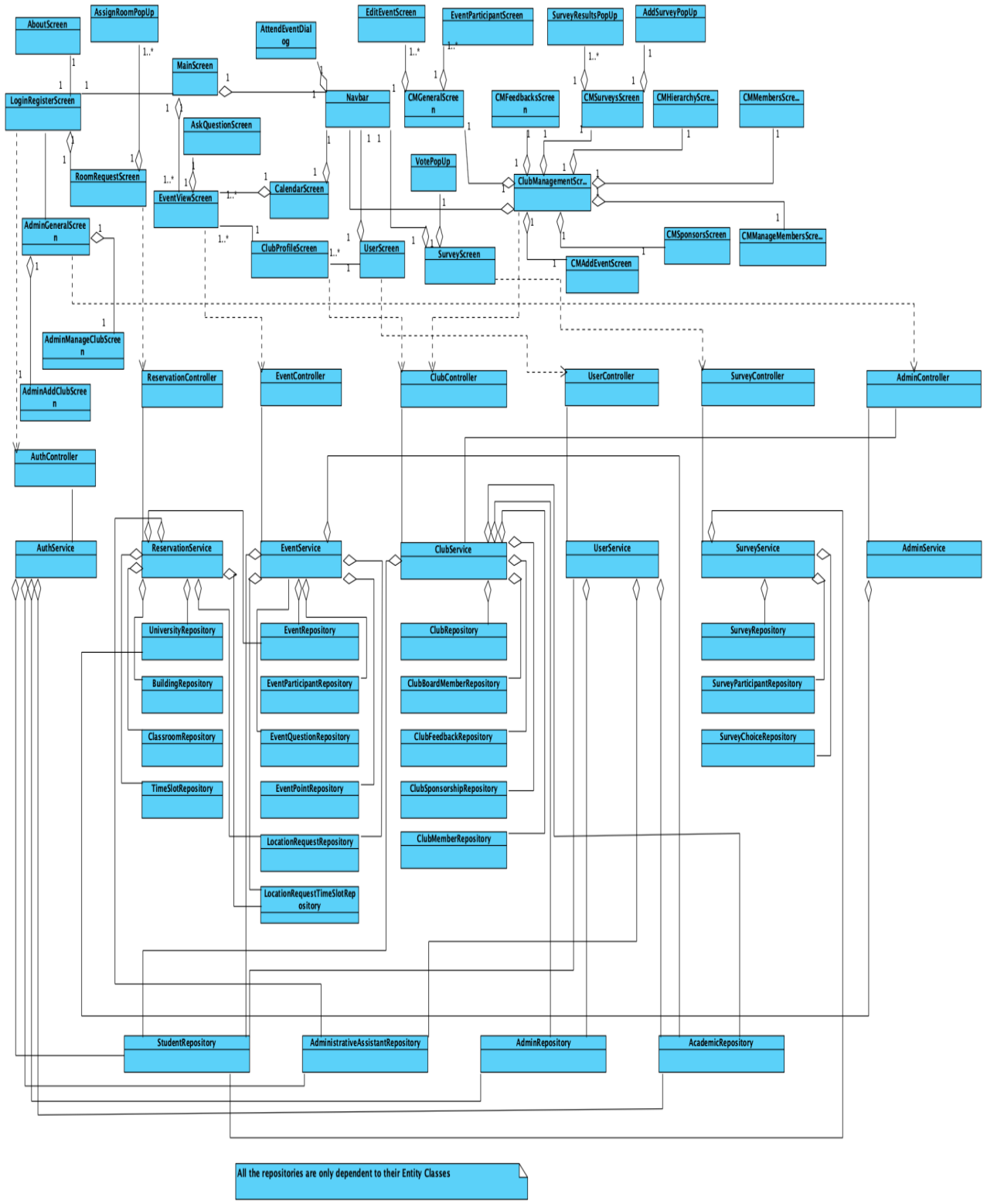

Fig. 4. Final Object Design.

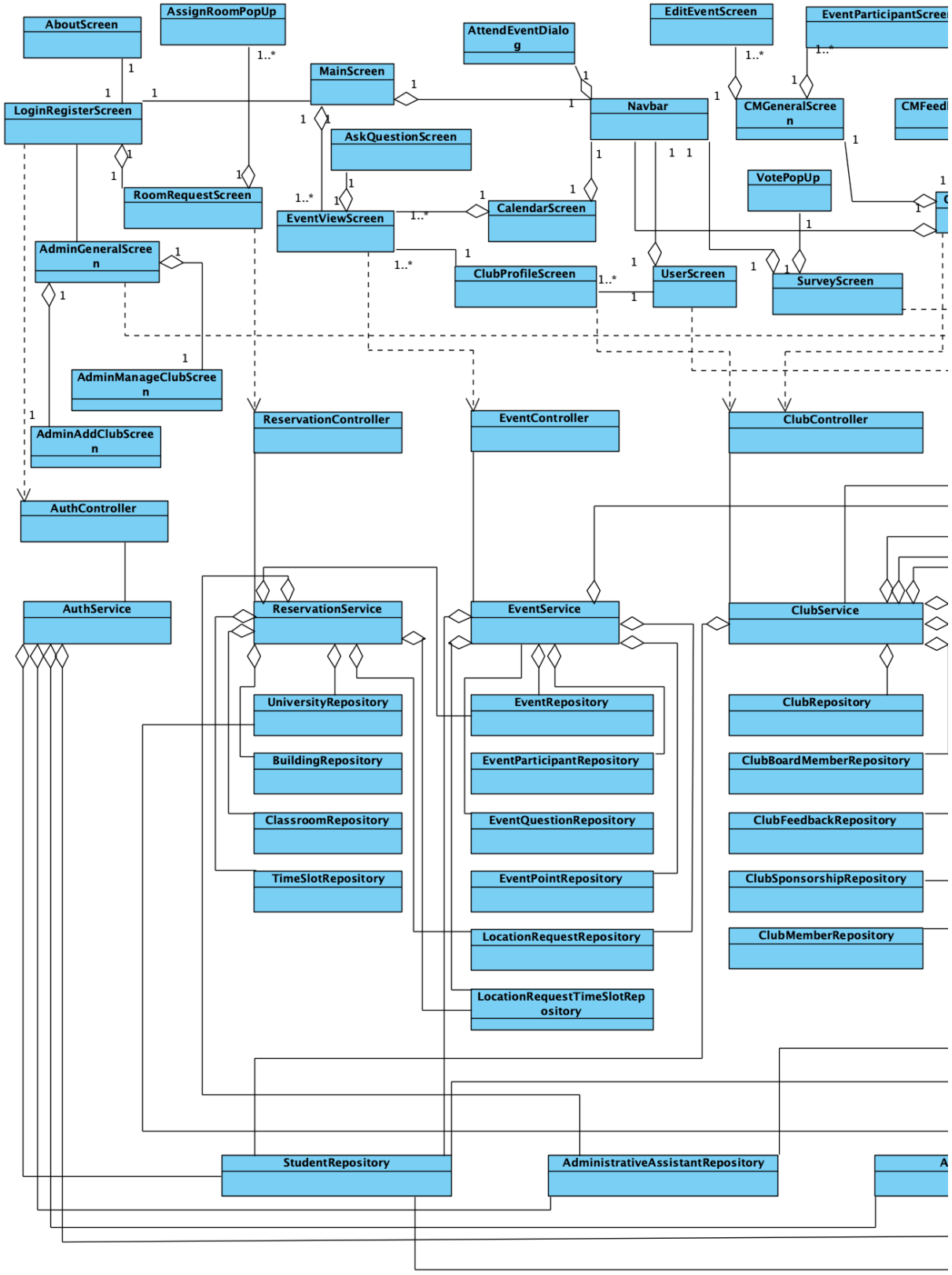

Fig. 5. Left Part of the Final Object Design.

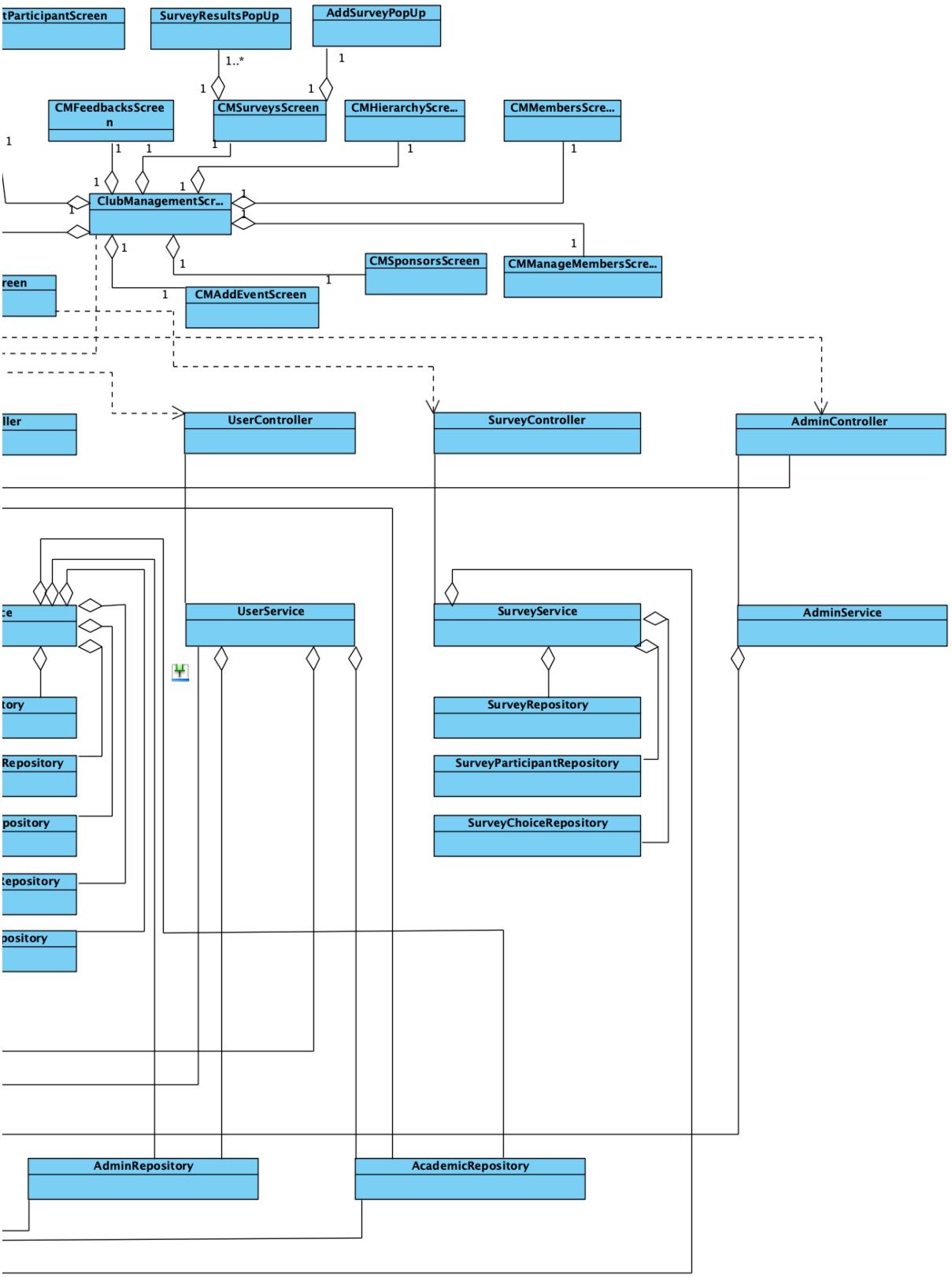

Fig. 6. Right Part of the Final Object Design.

3.3 Packages

3.3.1 Packages Introduced by Developers

3.3.1.1 controller

This package has all the controller classes that handle requests coming from the clients.

3.3.1.2 entity

The entity package has all the entity classes that are the base of the database.

3.3.1.3 repository

This package has all the repository interfaces that perform queries and communicate with tables.

3.3.1.4 requestModel

This package has all models that request bodies have.

3.3.1.5 responseModel

This package has all models that response data is constructed.

3.3.1.6 security

This package has security-related classes that perform access limitation, authorization, CORS policy etc.

3.3.1.7 service

This package has all service interfaces that help controllers to handle requests.

3.3.1.8 serviceImplementation

This package has all service implementations that have implementations of all functions in its interfaces.

3.3.1.9 dto

This package has all dto objects that are helping communicate between layers.

3.3.1.10 util

This package has classes that help other classes by utilizing some functionalities.

3.3.1.11 exception

This package has common exceptions that can occur in runtime.

3.3.1.12 resources

This package has environment variables that are used in the program.

3.3.2 External Library Packages

3.3.2.1 springframework.data.jpa

This library makes it easy to implement JPA based repositories and queries. This module deals with enhanced support for entity data access layers.

3.3.2.2 springframework.security

This module provides the implemented spring functionalities like authentication, session trace, defining public accessible endpoints or password encoding and decoding.

3.3.2.3 io.jsonwebtoken

This library helps to construct, encode and decode JWT to trace user authorization.

3.3.2.4 springframework.mysql

This implemented library helps to construct a connection with the MySQL database.

3.3.2.5 javax.mail

This package provides IMAP, SMTP objects and functions to handle email traffic.

3.3.2.6 org.modelmappeer

This package helps to make object transactions between layers

3.4 Class Interfaces

3.4.1 User Interface Management Layer

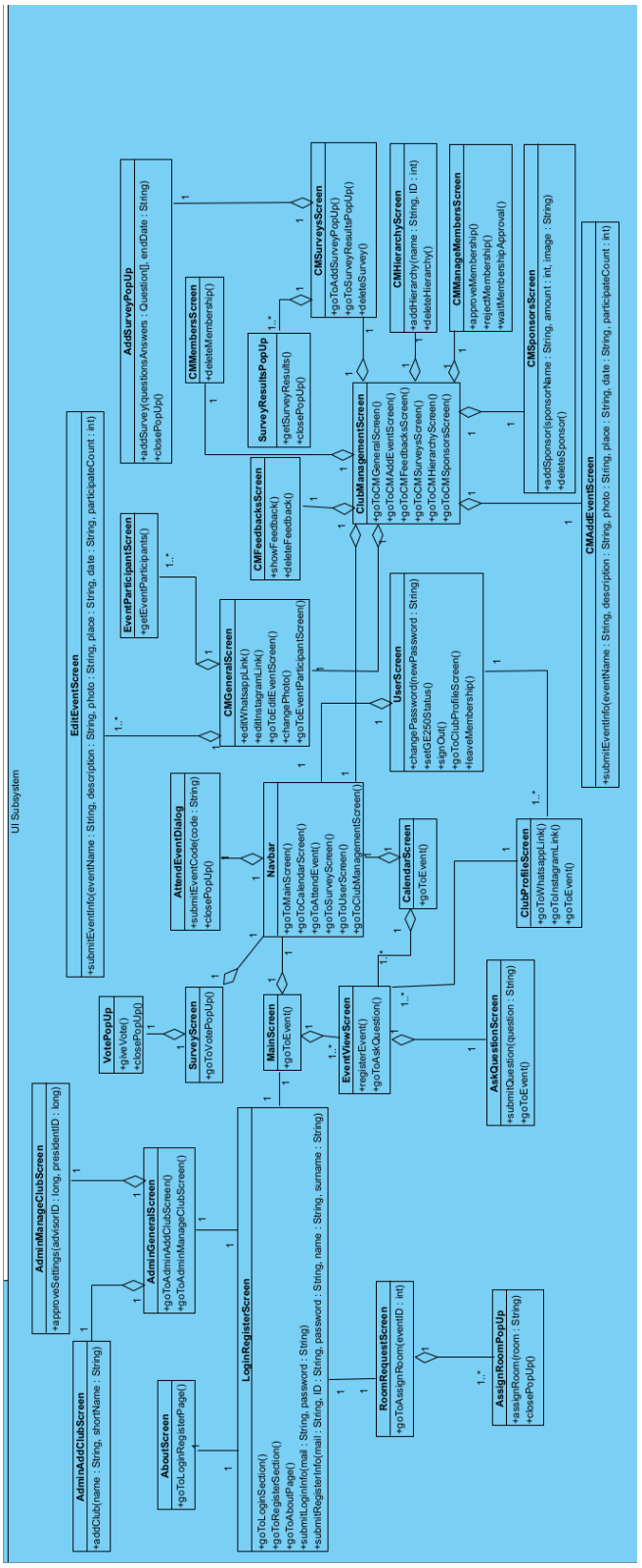

Fig. 7. User Interface Management Layer Composition.

This layer is a bridge between the user and bottom layers. In this layer, all classes are JavaScript files with the React library. The classes in this layer communicate with Web Server Layer's classes. Classes in this layer use components that can easily be modified. Two types of associations were used in order to implement the layer. The first association type is aggregation, which means that screens can be accessed from the aggregate. For example, to navigate to the calendar screen, navbar should be used; or to access the club sponsors page, board members should first go to the club management screen, and proceed to the sponsors page. The other type of association is basic association that shows direct relations and connects screens directly. For instance, after logging in, the main screen will be opened. The diagram is divided into following pages for easy reading.

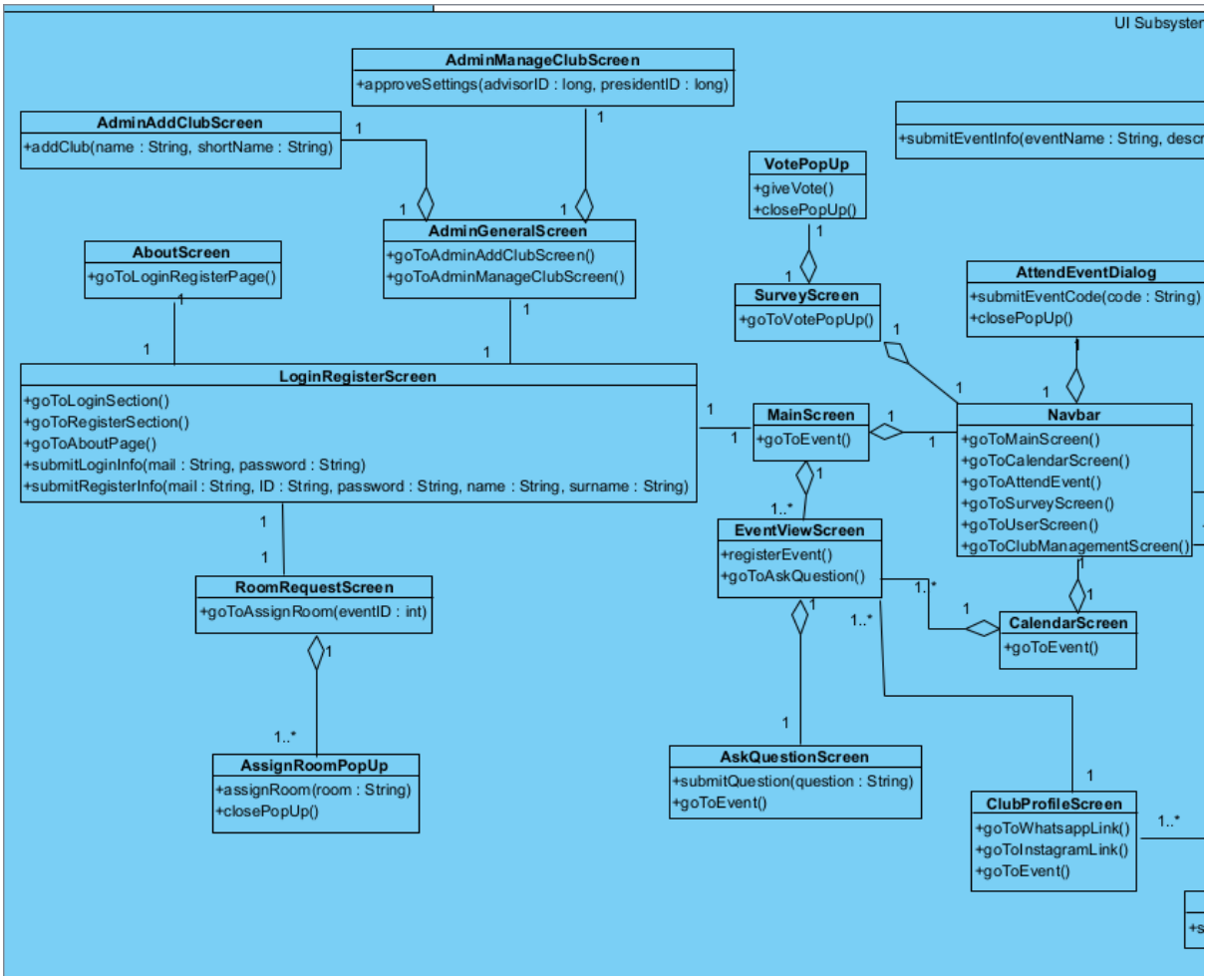

Fig. 8. Left Part of the User Interface Management Layer Composition.

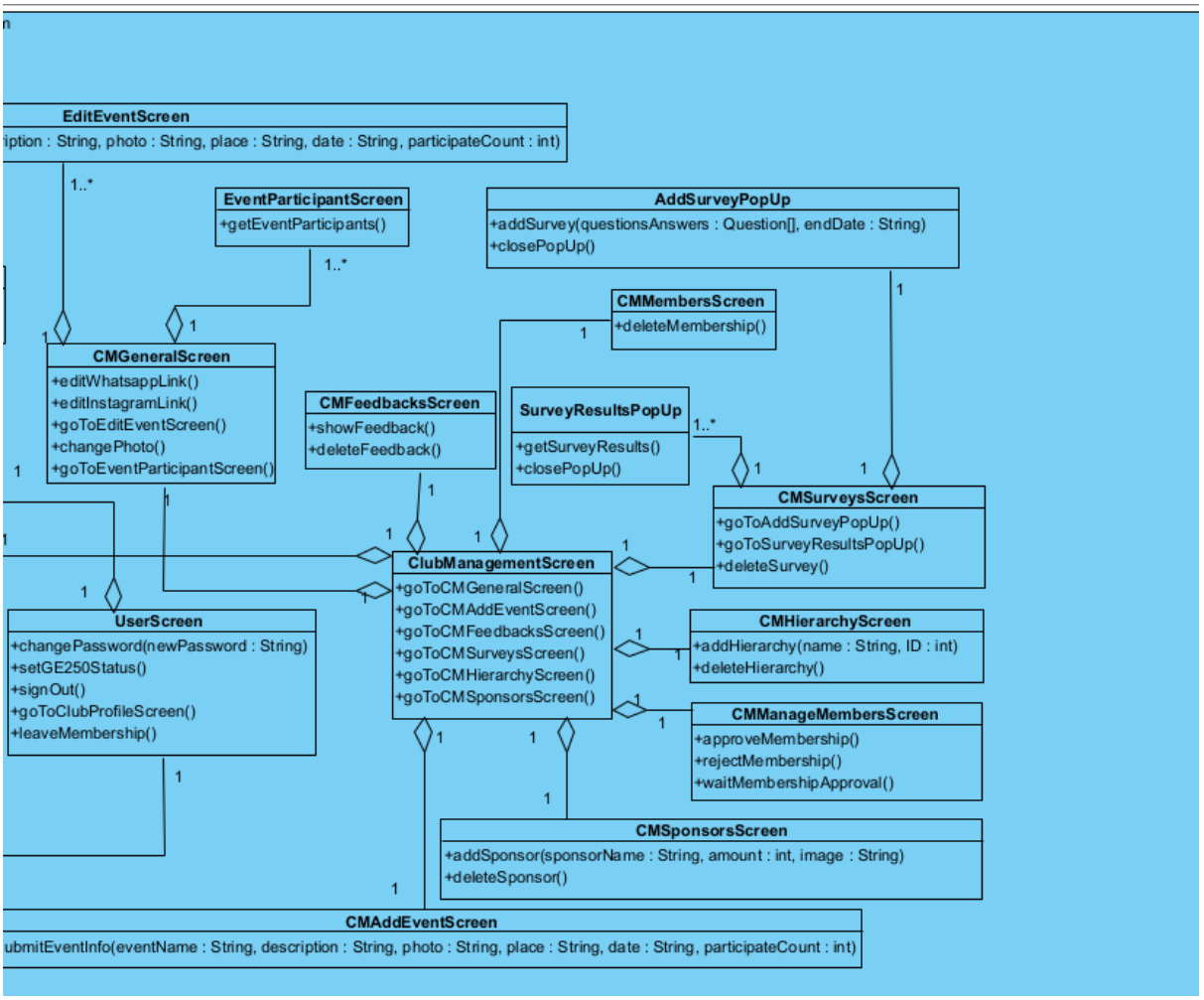

Fig. 9. Right Part of the User Interface Management Layer Composition.

Detailed descriptions of the classes are shown below:

3.4.1.1 AboutScreen

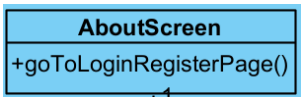

Fig. 10. About Screen.

This page shows some information about BilBoard.

Operations:

public goToLoginRegisterPage(): On click, login/register page opens.

3.4.1.2 LoginRegisterScreen

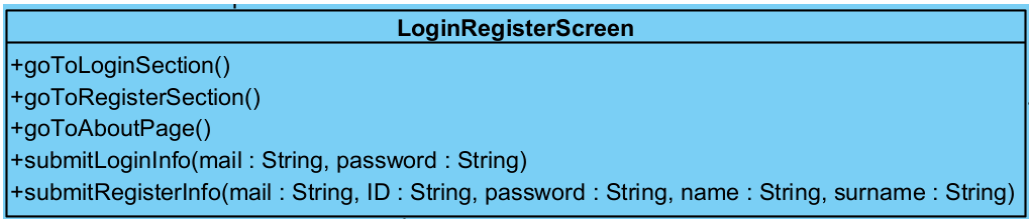

Fig. 11. Login Register Screen.

Users can login and register with this page.

Operations:

public goToLoginSection(): On click, the login section opens.

public goToRegisterSection(): On click, the register section opens.

public goToAboutPage(): On click, the page opens.

public submitLoginInfo(String mail, String password): On click, the information provided is sent to the database and checked, if correct, main screen opens for all members except administrative assistants. Room request screen opens for administrative assistants.

public submitRegisterInfo(String mail, String ID, String password, String name, String surname): On click, the information provided is sent to the database, and the user is created in the database.

3.4.1.3 RoomRequestScreen

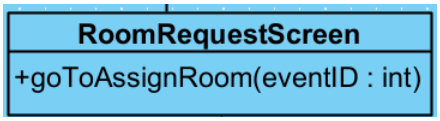

Fig. 12. Room Request Screen.

This screen is designed for administrative assistants and they can manage room requests from this screen.

Operations:

public goToAssignRoom(int eventID): On click, the administrative assistant goes to the assigned room for the specified event.

3.4.1.4 AssignRoomPopUp

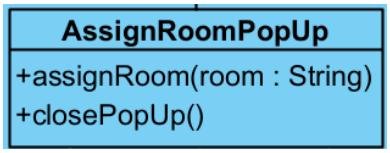

Fig. 13. Assign Room Pop Up.

This pop up is for assigning room to an event.

Operations:

public assignRoom(String room): Assistants can assign the room by this operation.

public closePopUp(): On click, pop up closes.

3.4.1.5 MainScreen

Fig. 14. Main Screen.

This screen includes the events that the user is a member of. Also, discovering other events can be done with this screen.

Operations:

public goToEvent(): On click, the event screen opens.

3.4.1.6 Navbar

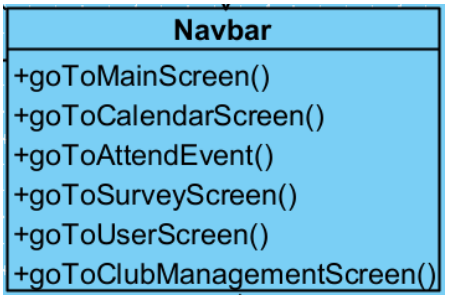

Fig. 15. Navbar.

Navbar provides navigation between screens of the BilBoard.

Operations:

public goToMainScreen(): On click, main screen opens.

public goToCalendarScreen(): On click, calendar screen opens.

public goToAttendEvent(): On click, attend event dialog opens.

public goToSurveyScreen(): On click, survey screen opens.

public goToUserScreen(): On click, user screen opens.

public goToClubManagementScreen(): On click, club management screen opens.

3.4.1.7 CalendarScreen

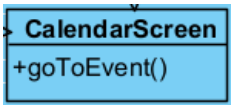

Fig. 16. Calendar Screen.

This screen shows upcoming and past events.

Operations:

public goToEvent(): On click, the event screen opens.

3.4.1.8 AttendEventDialog

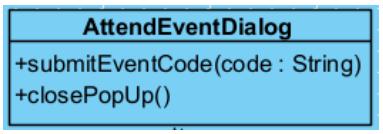

Fig. 17. Attend Event Dialog.

This pop up provides attending events.

Operations:

public submitEventCode(String code): On click, event code is sent to the database and checked. If the code is true, event attendance is taken. Otherwise, an error message is shown.

public closePopUp(): On click, the pop up closes.

3.4.1.9 EventViewScreen

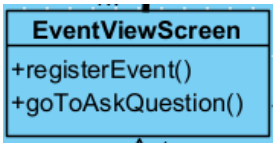

Fig. 18. Event View Screen.

This screen is an overview of an event.

Operations:

public registerEvent(): On click, the user becomes registered to the event.

public goToAskQuestion(): On click, asking question screen opens.

3.4.1.10 AskQuestionScreen

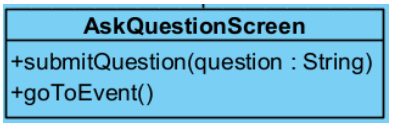

Fig. 19. Ask Question Screen.

This screen is for asking questions about a specific event.

Operations:

public submitQuestion(String question): On click, the question is submitted and sent to board members.

public goToEvent(): On click, the event screen opens.

3.4.1.11 Survey Screen

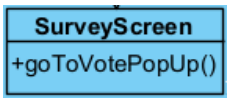

Fig. 20. Survey Screen.

This screen includes surveys that are organized by board members.

Operations:

public goToVotePopUp(): On click, voting for a specific survey pop up opens.

3.4.1.12 VotePopUp

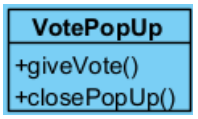

Fig. 21. Vote Pop Up.

This pop up provides voting for members.

Operations:

public giveVote(): On click, selected answers are submitted and sent to the database.

public closePopUp(): On click, pop up closes.

3.4.1.13 UserScreen

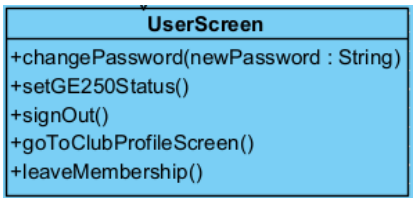

Fig. 22. User Screen.

This screen shows the information about the user and allows some operations.

Operations:

public changePassword(String newPassword): On click, password changes.

public setGE250Status(): On click, GE250 status is updated.

public signOut(): On click, the user signs out.

public goToClubProfileScreen(): On click, the user can go to the club screens that he/she has membership.

leaveMembership(): On click, the user can leave membership of the specified club.

3.4.1.14 ClubProfileScreen

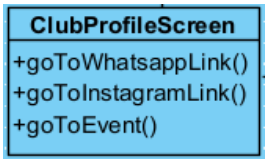

Fig. 23. Club Profile Screen.

This screen shows information about the clubs.

Operations:

public goToWhatsappLink(): On click, the Whatsapp group of the club opens.

public goToInstagramLink(): On click, the Instagram account of the club opens.

public goToEvent(): On click, the event page opens.

3.4.1.15 ClubManagementScreen

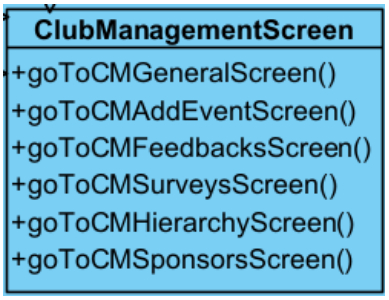

Fig. 24. Club Management Screen.

This screen behaves as navigator for all club management screens.

Operations:

public goToCMGeneralScreen(): On click, goes to club management general screen.

public goToCMAddEventScreen(): On click, goes to club management adding an event screen.

public goToCMFeedbacksScreen(): On click, goes to club management feedbacks screen.

public goToCMSurveysScreen(): On click, goes to club management surveys screen.

public goToCMHierarchyScreen(): On click, goes to club management club hierarchy screen.

public goToCMSponsorsScreen(): On click, goes to club management sponsors screen.

3.4.1.16 CMGeneralScreen

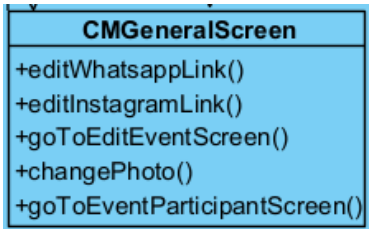

Fig. 25. CM General Screen.

This screen is used to manage general settings of the club.

Operations:

public editWhatsappLink(): It is used to change the Whatsapp link of the club.

public editInstagramLink(): It is used to change the Instagram link of the club.

public goToEditEventScreen(): It is used for editing events.

public changePhoto(): It is used to change club photos.

public goToEventParticipantScreen(): It is used to see the participants of a specific event.

3.4.1.17 EditEventScreen

Fig. 26. Edit Event Screen.

This screen is used to edit the event information.

Operations:

public submitEventInfo(String eventName, String description, String photo, String place, String date, int participateCount): On click, edited event information is submitted.

3.4.1.18 EventParticipantScreen

Fig. 27. Event Participant Screen.

It is used to see the participants of an event.

Operations:

public getEventParticipants(): On click, event participants are seen.

3.4.1.19 CMAddEventScreen

It is used for adding events.

Fig. 28. CM Add Event Screen.

Operations:

public submitEventInfo(String eventName, String description, String photo, String place, String date, int participateCount): On click, added event information is submitted.

3.4.1.20 CMMembersScreen

This screen shows the members of the club.

Fig. 29. CM Members Screen.

Operations:

public deleteMembership(): On click, the membership is deleted from the club.

3.4.1.21 CMManageMembersScreen

Board members can manage membership requests with this screen.

Fig. 30. CM Manage Members Screen.

Operations:

public approveMembership(): On click, membership request is approved and sent to the database.

public rejectMembership(): On click, membership request is rejected and sent to the database.

public waitMembershipApproval(): On click, membership request becomes pending and waits for other board members' approval.

3.4.1.22 CMFeedbacksScreen

This screen is used to see the feedback given to the club.

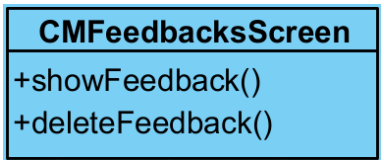

Fig. 31. CM Feedbacks Screen.

Operations:

public showFeedback(): On click, the board member can see the feedback.

public deleteFeedback(): On click, the feedback is deleted.

3.4.1.23 CMSurveysScreen

This screen is used for creating and deleting surveys.

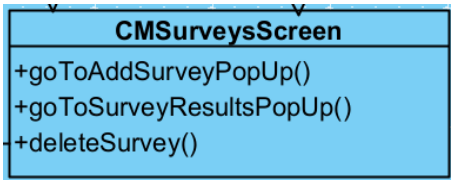

Fig. 32. CM Surveys Screen.

Operations:

public goToAddSurveyPopUp(): On click, adding survey pop up opens.

public goToSurveyResultsPopUp(): On click, survey results pop up opens.

public deleteSurvey(): On click, the survey is deleted.

3.4.1.24 AddSurveyPopUp

This pop up provides adding surveys.

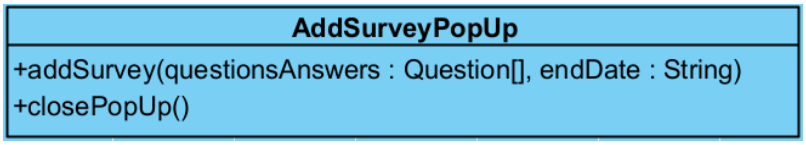

Fig. 33. Add Survey Pop Up.

Operations:

public addSurvey(Question[] questionsAnswers, String endDate): On click, the survey is added.

public closePopUp(): On click, pop up closes.

3.4.1.25 SurveyResultsPopUp

This pop up shows the survey results.

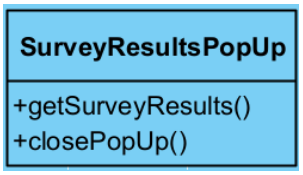

Fig. 34. Survey Results Pop Up.

Operations:

public getSurveyResults(): On click, survey results are obtained.

public closePopUp(): On click, the pop up closes.

3.4.1.26 CMHierarchyScreen

This screen provides editing club hierarchy (adding and deleting board members).

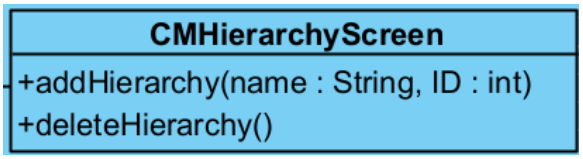

Fig. 35. CM Hierarchy Screen.

Operations:

public addHierarchy(String name, int ID): On click, a board member with specific name and id is added.

public deleteHierarchy(): On click, the board member is deleted from the hierarchy.

3.4.1.27 CMSponsorsScreen

This screen is used to add and delete sponsors.

Fig. 36. CM Sponsors Screen.

Operations:

public addSponsor(String sponsorName, int amount, String image): On click, the sponsor with specified information is added.

public deleteSponsor(): On click, the sponsor is deleted.

3.4.1.28 AdminGeneralScreen

This screen is for admins and provides adding and managing clubs.

Fig. 37. Admin General Screen.

Operations:

public goToAdminAddClubScreen(): On click, add club screen opens.

public goToAdminManageClubScreen(): On click, manage club screen opens.

3.4.1.29 AdminAddClubScreen

This screen is designed for adding clubs.

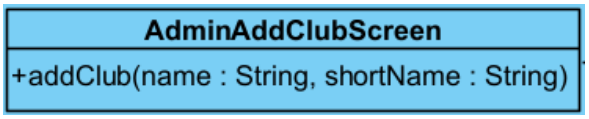

Fig. 38. Admin Add Club Screen.

Operations:

public addClub(String name, String shortName): On click, the club is added to BilBoard.

3.4.1.30 AdminManageClubScreen

This screen is designed for managing club advisor and president.

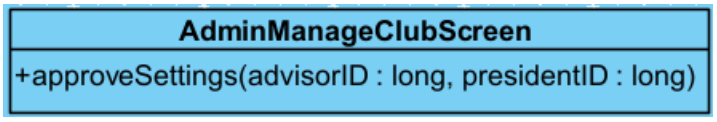

Fig. 39. Admin Manage Club Screen.

Operations:

public approveSettings(long advisorID, long presidentID): On click, the club advisor and president is set.

3.4.2 Web Server Layer

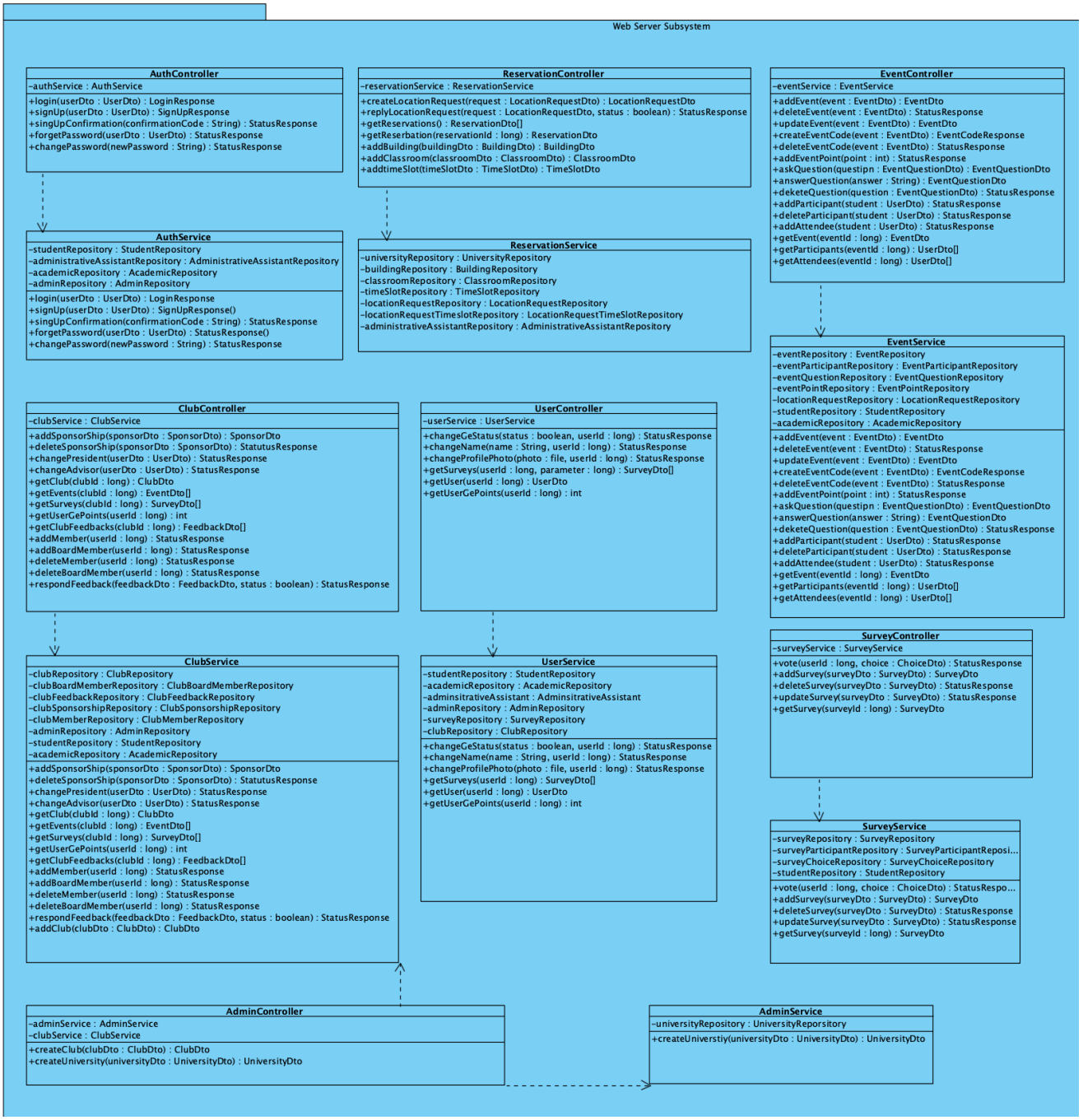

Fig. 40. Web Server Layer Composition.

This layer handles the request coming from the client and returns required responses. It constructs the communication between user interface and data management layer.

3.4.2.1 AuthController

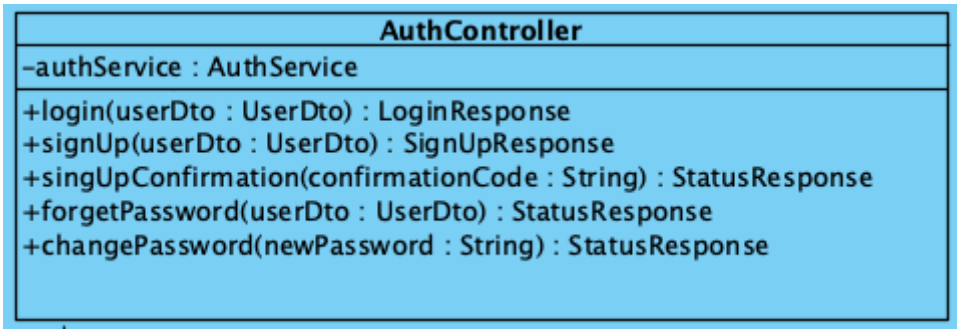

Fig. 41. Auth Controller.

A class that controls the authorization functionalities through authService.

Attributes:

private AuthService authService: This is a service that actual functionality happens.

Operations:

public LoginResponse login(UserDto userDto): This function takes password and email from the userDto objects. It sends them to authService.

public SignUpResponse signUp(UserDto userDto): This function takes all required parameters from the userDto object to create a new user. It sends it to service to handle service operation.

public StatusResponse signUpConfirmation(String confirmationCode): This method confirms account. Client calls this function.

public StatusResponse forgetPassword(UserDto userDto): This one creates a new passwordResetToken to handle reset operations. It determines who forget password and put token its entity in service

public StatusResponse changePassword(String newPassword): This method changes user password. It sends the password to the service function directly.

3.4.2.2 AuthService

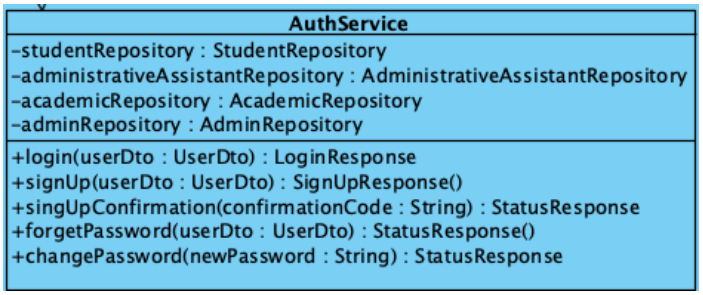

Fig. 42. Auth Service.

This class controls all the operations that are related to authorization.

Attributes:

private StudentRepository studentRepository: This is a repository to handle queries from the student entities.

private AdministrativeAssistantRepository administrativeAssistantRepository: This is a repository to handle queries from the administrativeAssistant entities.

private AcademicRepository academicRepository: This is a repository to handle queries from the academic entities.

private AdminRepository adminRepository: This is a repository to handle queries from the admin entities.

Operations:

public LoginResponse login(UserDto userDto): This method takes the user from the database while searching it by email. When it finds it, it compares the encrypted password and password that comes from the client.

public SignUpResponse signUp(UserDto userDto): This method creates a new row in the database which is equal to UserEntity.

public StatusResponse signUpConfirmation(String confirmationCode): This method change isConfirmed status of the user object in the database.

public StatusResponse forgetPassword(UserDto userDto): This method creates a new passwordResetToken and saves it to handle reset operations.

public StatusResponse changePassword(String newPassword): This method changes user password. It saves an encrypted version of newPassword.

3.4.2.3 ReservationController

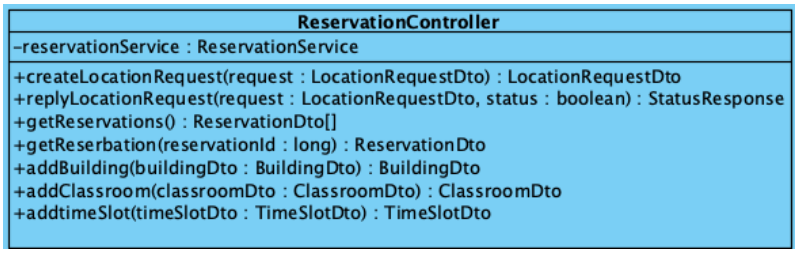

Fig. 43. Reservation Controller.

This class constructs location requests operations.

Attributes:

private ReservationService reservationService: It is the reservation service that operates all the reservation operations.

Operations:

public LocationRequestDto createLocationRequest(LocationRequestDto locationRequestDto): It is called by the user and triggered when a location request is created for events.

public StatusResponse replyLocationRequest(LocationRequestDto request, boolean status): It is for handling location request responses.

public ReservationDto[] getReservations(): It returns all the reservation objects that are not given response.

public BuildingDto addBuildings(BuildingDto buildingDto): It adds a new building by using buildingDto.

public ClassroomDto addClassroom(ClassroomDto classroomDto): It creates a new class from classroomDto.

public TimeSlotDto addTimeSlot(TimeSlotDto timeSlotDto): It adds a new time slot from timeSlotDto object.

3.4.2.4 ReservationService

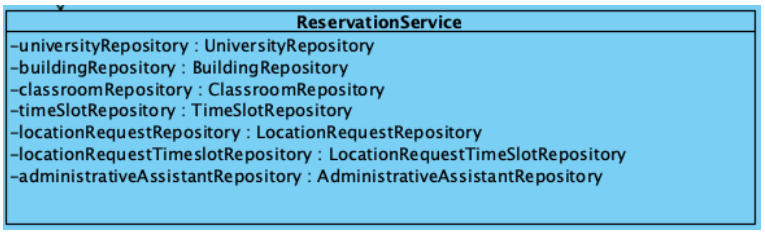

Fig. 44. Reservation Service.

This is a service interface that handles administrative assistant operations.

Attributes:

private UniversityRepository universityRepository: This is a repository to handle queries from the university entities.

private BuildingRepository buildingRepository: This is a repository to handle queries from the building entities.

private ClassroomRepository classroomRepository: This is a repository to handle queries from the classroom entities.

private TimeSlotRepository timeSlotRepository: This is a repository to handle queries from the time slot entities.

private LocationRequestRepository locationRequestRepository: This is a repository to handle queries from the locationRequest entities.

private AdministrativeAssistantRepository administrativeAssistantRepository: This is a repository to handle queries from the administrativeAssistant entities.

Operations:

public LocationRequestDto createLocationRequest(LocationRequestDto locationRequestDto): It creates a new locationRequest object and saves it to the database.

public StatusResponse replyLocationRequest(LocationRequestDto request, boolean status): It changes the locationRequest status of the request object in the database.

public ReservationDto[] getReservations(): It finds all the reservationDto datas from the database.

public BuildingDto addBuildings(BuildingDto buildingDto): It creates a new building and saves it to the database.

public ClassroomDto addClassroom(ClassroomDto classroomDto): It creates a new classroom and saves it to the database.

public TimeSlotDto addTimeSlot(TimeSlotDto timeSlotDto): It adds a new timeSlot to the database while the field comes from the timeSlotDto object.

3.4.2.5 EventController

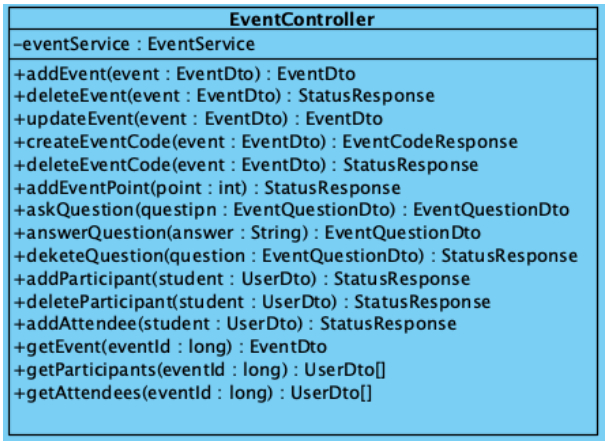

Fig. 45. Event Controller.

This is a controller that operates event functionalities.

Attributes:

private EventService eventService: It is the event service that operates all the event related operations.

Operations:

public EventDto addEvent(EventDto eventDto): It adds a new event by using eventDto properties.

public StatusResponse deleteEvent(EventDto eventDto): It deletes events from the system.

public EventDto updateEvent(EventDto eventDto): It updates events by getting parameters from the eventDto objects and returns updated events.

public EventCodeResponse createEventCode(EventDto eventDto): It constructs a new event code and returns eventCodeResponse.

public StatusResponse deleteEventCode(EventDto eventDto): It deletes events from the system. It returns whether the operation is done successfully or not.

public StatusResponse addEventPoint(int point): When a participant gives a point to an event, it is triggered and adds this point accordingly.

public EventQuestionDto askQuestion(EventQuestionDto questionDto): It creates a new questionDto object.

public EventQuestionDto answerQuestion(String answer,EventQuestionDto questionDto): When the board member or president answers the question from the client side, this method is triggered.

public StatusResponse deleteQuestion(EventQuestionDto eventQuestionDto): It is triggered when a question is removed from the client sice.

public StatusResponse addParticipant(UserDto userDto): It is triggered when a user enrolls in the event.

public StatusResponse deleteParticipant(UserDto userDto): It is triggered when a user cancels enrollment.

public StatusResponse addAttendee(UserDto userDto): It is triggered when the user enters event code.

public EventDto getEvent(long eventId): It returns an event whose id is equal to eventId.

public UserDto[] getParticipant(long eventId): It returns participants of events.

public UserDto[] getAttendees(long eventId): It returns participants who really attend the event.

3.4.2.6 EventService

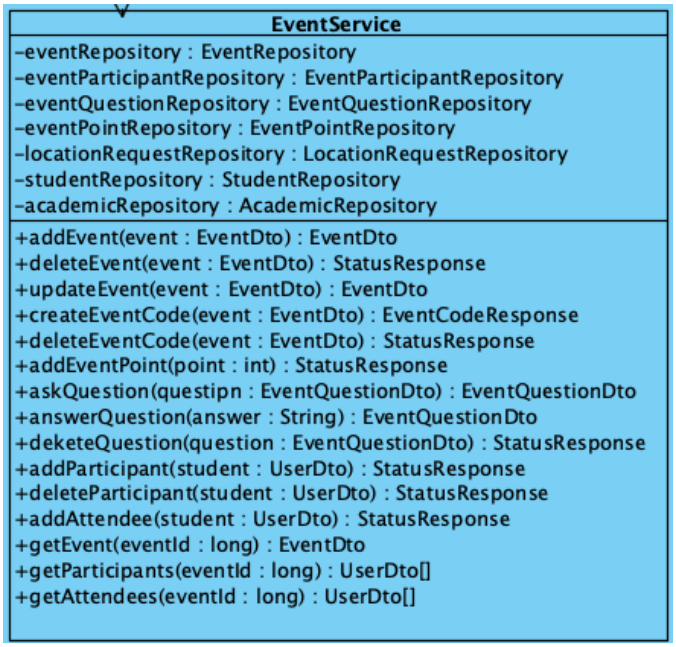

Fig. 46. Event Service.

This is a service interface that handles event operations.

Attributes:

private EventRepository eventRepository: This is a repository to handle queries from the event entities.

private EventParticipantRepository eventParticipantRepository: This is a repository to handle queries from the eventParticipant entities.

private EventQuestionRepository eventQuestionRepository: This is a repository to handle queries from the eventQuestion entities.

private EventPointRepository eventPointRepository: This is a repository to handle queries from the eventPoint entities.

private LocationRequestRepository locationRequestRepository: This is a repository to handle queries from the locationRequest entities.

private StudentRepository studentRepository: This is a repository to handle queries from the student entities.

private AcademicRepository academicRepository: This is a repository to handle queries from the academic entities.

Operations:

public EventDto addEvent(EventDto eventDto): It creates a new event in the database with given properties.

public StatusResponse deleteEvent(EventDto eventDto): It removes the event from the data system. It returns whether the operation is done successfully or not.

public EventDto updateEvent(EventDto eventDto): It updates the given eventDto in the database and returns an updated event object.

public EventCodeResponse createEventCode(EventDto eventDto): It creates a new event code and saves it in its event. It returns an eventCodeResponse which consists of the eventCode.

public StatusResponse deleteEventCode(EventDto eventDto): It removes deleteEventCode field from given event object and saves it to database.

public StatusResponse addEventPoint(int point): It adds points to events and calculates a new average rate. When these operations are done, results are saved to the database.

public EventQuestionDto askQuestion(EventQuestionDto questionDto): It creates a new eventQuestion object in the database.

public EventQuestionDto answerQuestion(String answer,EventQuestionDto questionDto): It fills the answer field of the eventQuestionDto object in dto.

public StatusResponse deleteQuestion(EventQuestionDto eventQuestionDto): It deletes question objects from the database and returns the operation result.

public StatusResponse addParticipant(UserDto userDto): It adds a new participant to the event. There are many to many tables which trace user-event relations. This method constructs a new row in that table.

public StatusResponse deleteParticipant(UserDto userDto): It removes participants from the eventParticipant table.

public StatusResponse addAttendee(UserDto userDto): It changes participant status from not attended to attended.

public EventDto getEvent(long eventId): It returns events with a given id.

public UserDto[] getParticipant(long eventId): It returns all the participants that are participants in the event given id.

public UserDto[] getAttendees(long eventId): It returns attended participants of an event with a given id.

3.4.2.7 ClubController

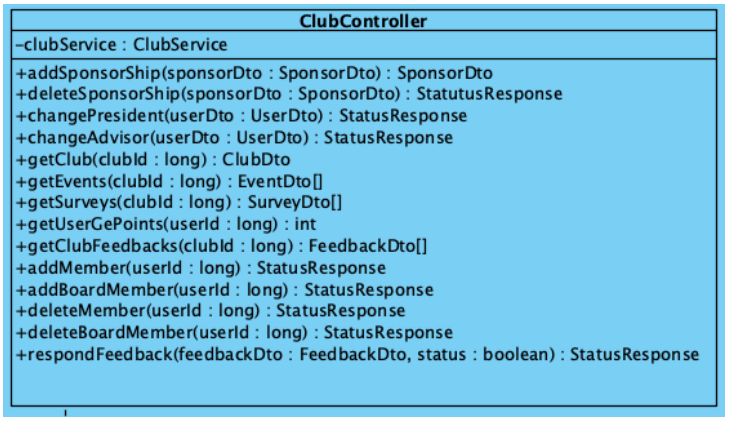

Fig. 47. Club Controller.

Attributes:

private ClubService clubService: It is the club service that operates all the club related operations.

Operations:

public SponsorDto addSponsorShips(SponsorDto sponsorDto): It adds sponsor to club with given parameters.

public StatusResponse deleteSponsorShip(SponsorDto sponsorDto): It deletes the sponsorship from the club.

public ClubDto getClub(long clubId): It returns a club with a given id.

public Event[] getEvents(long clubId): It returns all the events of the club given id.

public Survey[] getSurveys(long clubId): It returns all surveys of the club given id.

public GePoints[] getUserGePoints(long userId): It returns ge points of members in the club.

public FeedbackDto[] getClubFeedbacks(long userId): It returns all the feedback that the club gets.

public StatusResponse addMember(long userId): It adds a new member to the club**.** It is not triggered by the client.

public StatusResponse addBoardMember(long userId): It adds a new board member to the club. It only can be triggered by the president.

public StatusResponse deleteMember(long userId): It deletes members from the club.

public StatusResponse deleteBoardMember(long userId): It deletes board members by given id.

public StatusResponse respondFeedback(FeedbackDto feedbackDto, boolean status): Feedback is given responded by board members.

public ClubDto addClub(ClubDto clubDto): This method creates a new club with a given id. It can be triggered by only the admin.

3.4.2.8 ClubService

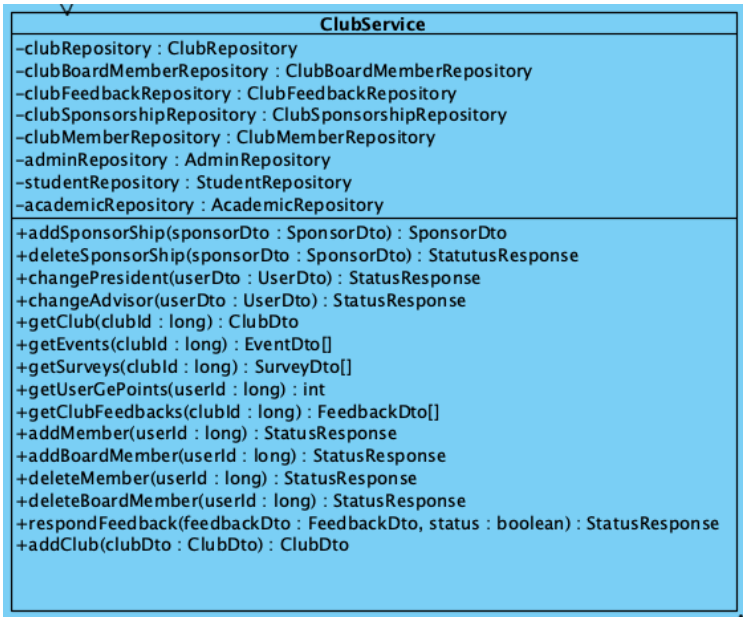

Fig. 48. Club Service.

Attributes:

private ClubRepository clubRepository: This is a repository to handle queries from the club entities.

private ClubBoardMemberRepository clubBoardMemberRepository: This is a repository to handle queries from the clubBoardMember entities.

private ClubFeedbackRepository clubFeedbackRepository: This is a repository to handle queries from the clubFeedback entities.

private ClubSponsorshipRepository clubSponsorshipRepository: This is a repository to handle queries from the clubSponsorship entities.

private ClubMemberRepository clubMemberRepository: This is a repository to handle queries from the clubMember entities.

private AdminRepository adminRepository: This is a repository to handle queries from the admin entities.

private StudentRepository studentRepository: This is a repository to handle queries from the student entities.

private AcademicRepository academicRepository: This is a repository to handle queries from the academic entities.

Operations:

public SponsorDto addSponsorShips(SponsorDto sponsorDto): This function adds a new sponsor by saving it to the database.

public StatusResponse deleteSponsorShip(SponsorDto sponsorDto): This function delete sponsor from the database. It returns operation results.

public StatusResponse changePresident(UserDto userDto): This function changes the president field of the club in the database.

public StatusResponse changeAdvisor(UserDto userDto): This function changes the advisor field of the club in the database.

public ClubDto getClub(long clubId): This function takes the club from the table with the given id.

public Event[] getEvents(long clubId): This function returns all the events that have given id in their club field.

public Survey[] getSurveys(long clubId): This function returns all the surveys that have given id in their club field.

public int getUserGePoints(long userId): This function gives users ge points.

public FeedbackDto[] getClubFeedbacks(long userId): It returns all the feedback that is given by user given id.

public StatusResponse addMember(long userId): This function adds a new member to the club member table. It returns the result of the operation.

public StatusResponse addBoardMember(long userId): This function adds a new board member to the board member table.

public StatusResponse deleteMember(long userId): This function deletes users from the board member table. It returns the result of the operation.

public StatusResponse deleteBoardMember(long userId): This method deletes board members from the board member table. It returns the result of the operation.

public StatusResponse respondFeedback(FeedbackDto feedbackDto, boolean status): This method fills the responde field of feedback. It saves updated data.

public ClubDto addClub(ClubDto clubDto): This method adds a new club to the database. It returns created clubDto.

3.4.2.9 UserController

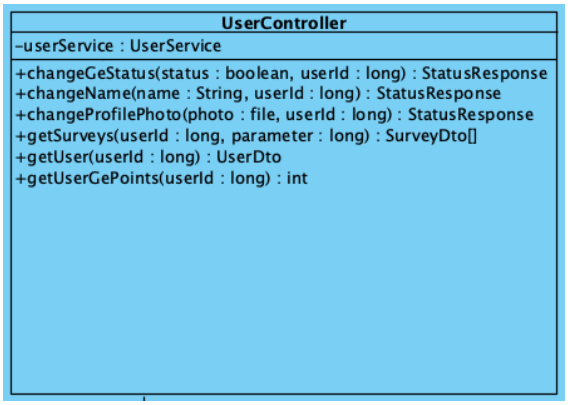

Fig. 49. User Controller.

Attributes:

private UserService userService: It is the user service that operates all the user related operations.

Operations:

public StatusResponse changeGeStatus(boolean status, long userId): This function is used to change the current GE status. System's response will be returned to the user.

public StatusResponse changeName(String name, long userId): This function is used to change the name of the user. System's response will be returned to the user.

public StatusResponse changeProfilePhoto(File photo, long userId): This function is used to change the photo of the profile. System's response will be returned to the user.

public SurveyDto[] getSurveys(long userId): This function is used to get the surveys where users can participate.

public UserDto getUser(long userId): This function is used to get an UserDto with specifying userId.

public int getUserGePoint(long userId): This function is used to get the GE point of a user.

3.4.2.10 UserService

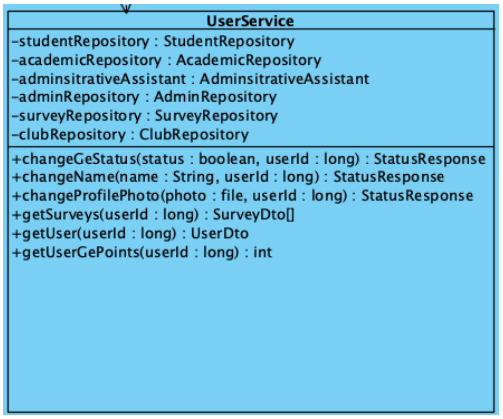

Fig. 50. User Service.

Attributes:

private StudentRepository studentRepository: This is a repository to handle queries from the student entities.

private AcademicRepository academicRepository: This is a repository to handle queries from the academic entities.

private AdministrativeAssistantRepository administrativeAssistantRepository: This is a repository to handle queries from the administrativeAssistant entities.

private AdminRepository adminRepository: This is a repository to handle queries from the admin entities.

private SurveyRepository surveyRepository: This is a repository to handle queries from the survey entities.

private ClubRepository clubRepository: This is a repository to handle queries from the club entities.

Operations:

public StatusResponse changeGeStatus(boolean status, long userId): It changes ge250/251 status of the user.

public StatusResponse changeName(String name, long userId): It changes user's name in table.

public StatusResponse changeProfilePhoto(File photo, long userId): This method takes userId and photo from the client, and updates the user's photo whose id is userId .

public SurveyDto[] getUser(long userId): This returns surveyDto list from table that users need to vote.

public UserDto getUserGePoint(long userId): This returns user's ge250/251 point with given id from table.

3.4.2.11 SurveyController

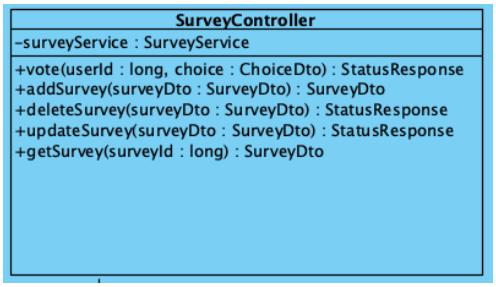

Fig. 51. Survey Controller.

Attributes:

private SurveyService surveyService: It is the survey service that operates all the survey related operations.

Operations:

public StatusResponse vote(long userId, ChoiceDto choiceDto): This function is used to give a vote to a survey. System's response will be returned to the user.

public SurveyDto addSurvey(SurveyDto surveyDto): This function is used to create a new survey which will be returned by the system.

public StatusResponse deleteSurvey(SurveyDto surveyDto): This function is used to delete a survey. System's response will be returned to the user.

public StatusResponse updateSurvey(SurveyDTo surveyDto): This function is used to update an already existing survey. System's response will be returned to the user.

public SurveyDto getSurvey(long surveyId): This function is used to get a survey object with specifying its id.

3.4.2.12 SurveyService

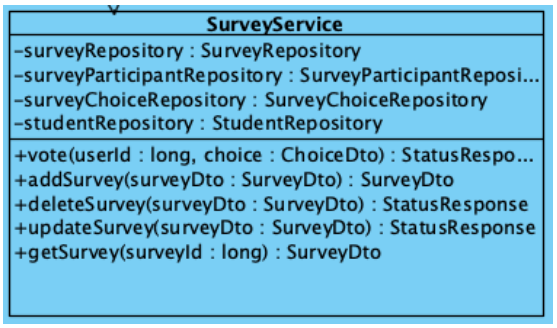

Fig. 52. Survey Service.

Attributes:

private SurveyRepository surveyRepository: This is a repository to handle queries from the survey entities.

private SurveyParticipantRepository surveyParticipantRepository: This is a repository to handle queries from the surveyParticipant entities.

private SurveyChoiceRepository surveyChoiceRepository: This is a repository to handle queries from the surveyChoice entities.

private StudentRepository studentRepository: This is a repository to handle queries from the student entities.

Operations:

public StatusResponse vote(long userId, ChoiceDto choiceDto): This function is used to vote on a question by taking user id and the choice. System's response will be returned to the user.

public SurveyDto addSurvey(SurveyDto surveyDto): This function is used to create SurveyDto object and to save to the database.

public StatusResponse deleteSurvey(SurveyDto surveyDto): This function is used to delete a SurveyDto object. System's response will be returned to the user.

public StatusResponse updateSurvey(SurveyDto surveyDto): This function is used to update a Survey object. System's response will be returned to the user.

public SurveyDto getSurvey(long surveyId): This function is used to get the SurveyDto with specifying surveyId.

3.4.2.13 AdminController

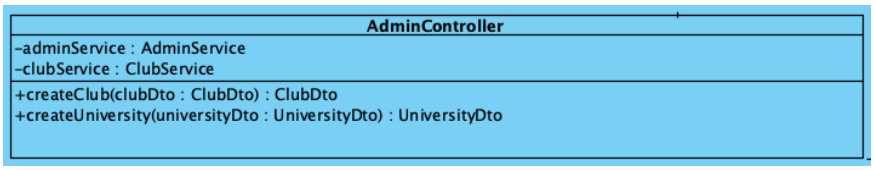

Fig. 53. Admin Controller.

Attributes:

private AdminService adminService : It is the admin service that operates all the admin related operations.

private ClubService clubService : It is the club service that operates all the club related operations.

Operations:

public ClubDto createClub(ClubDto clubDto): This function is used to create a ClubDto object.

private StatusResponse assignPresident(UserDto userDto, ClubDto clubDto): This function is used to assign a president to a specific club.

private UniversityDto createUniversity(UniversityDto universityDto): This function is used to create an UniversityDto object.

private StatusResponse assignAdvisor(UserDto userDto, ClubDto clubDto): This function is used to assign an advisor to a specific club.

3.4.2.14 AdminService

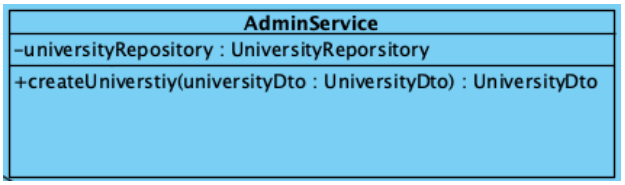

Fig. 54. Admin Service.

Attributes:

private UniversityRepository universityRepository: This is a repository to handle queries from the university entities.

Operations:

private UniversityDto createUniversity(UniversityDto universityDto): This function is used to create UniversityDto and to save to the database.

3.4.3 Data Management Layer

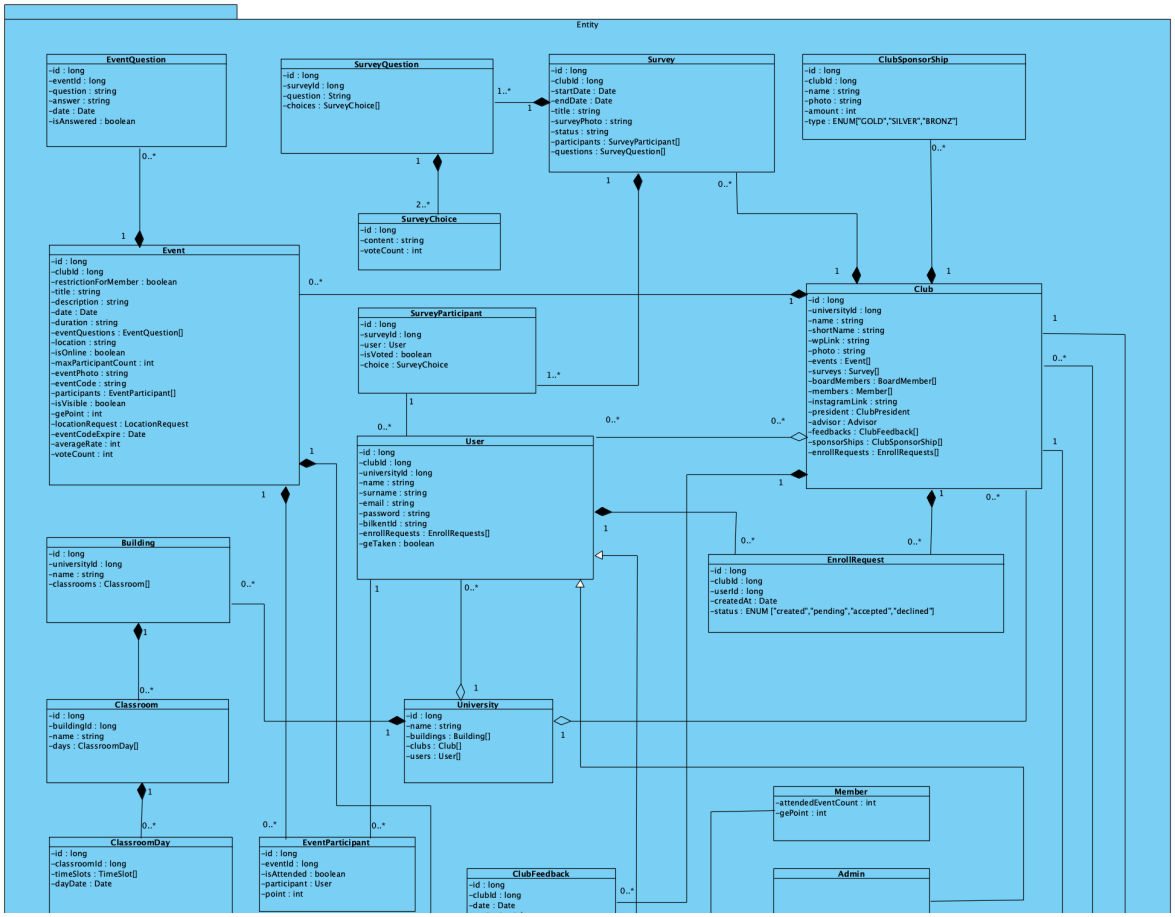

Fig. 55. Upper Part of the Entity of the Data Management Layer Composition.

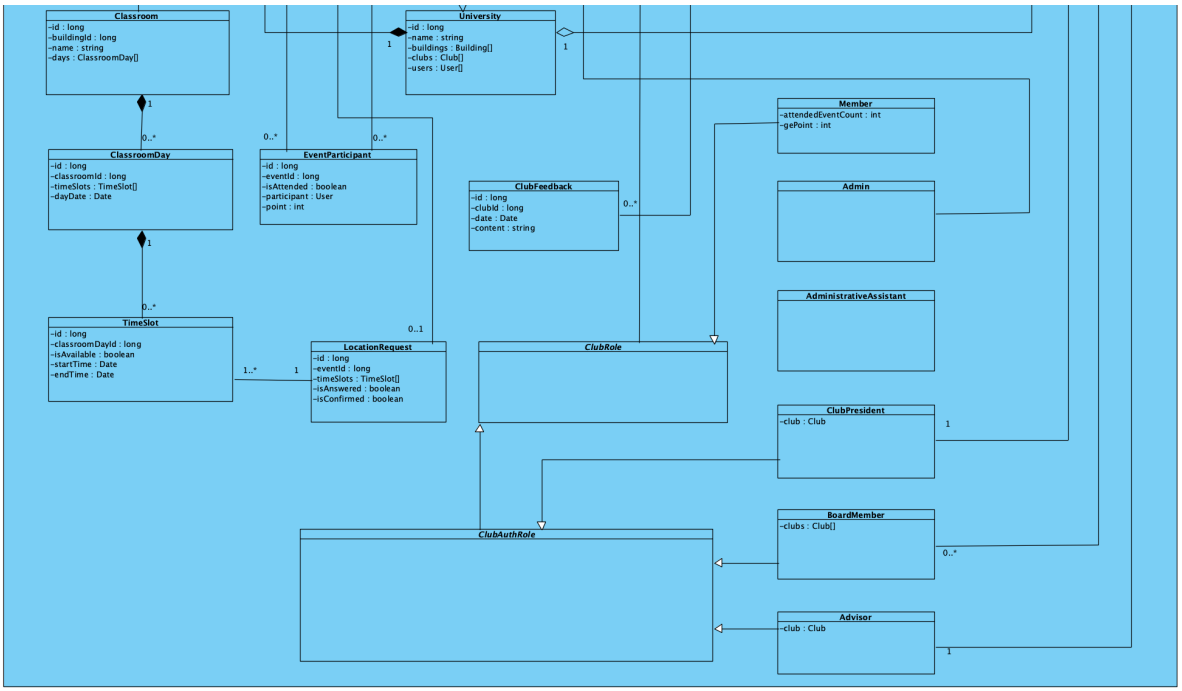

Fig. 56. Bottom Part of the Entity of the Data Management Layer Composition.

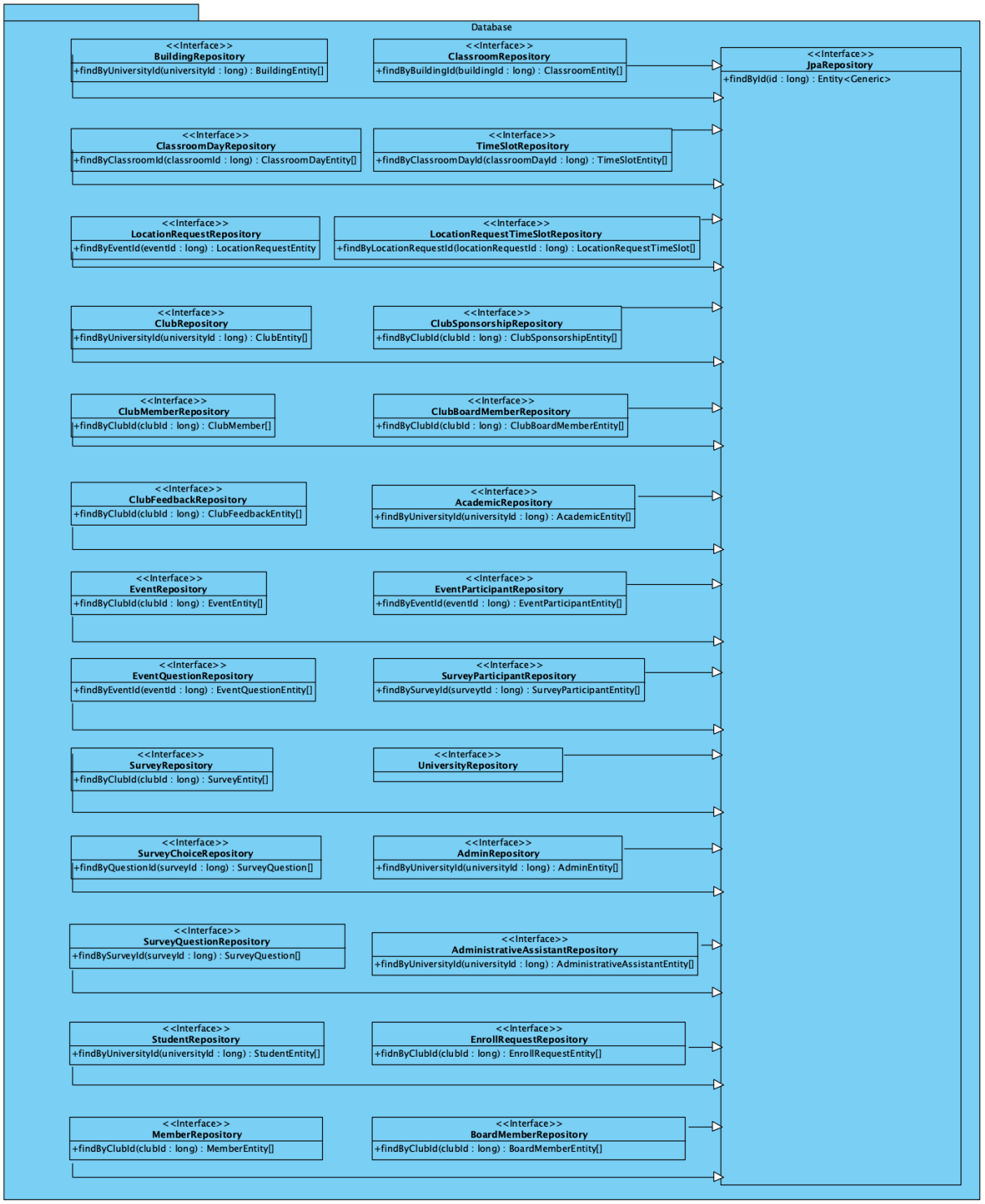

Fig. 57. Repository Part of the Data Management Layer Composition.

Important Note: In each entity class getters and setters are provided for each property. Also Each class has its own empty constructor. Because we are using a model mapper, we do not need constructors with parameters.

Detailed descriptions of the classes are shown below:

3.4.3.1 EventQuestion

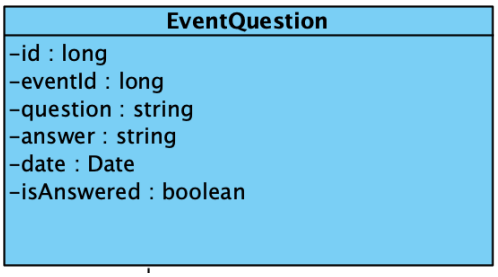

Fig. 58. Event Question.

Attributes:

private long id: This is the object's ID for tracing purposes in the database.

private long eventId: This is Event class' ID due to many to one relation.

private String question: This is event question content.

private String answer: This is the answer for the question.

private Date date: This is the date that the question was asked.

private boolean isAnswered: This is the information for whether the question is answered or not.

3.4.3.2 Event

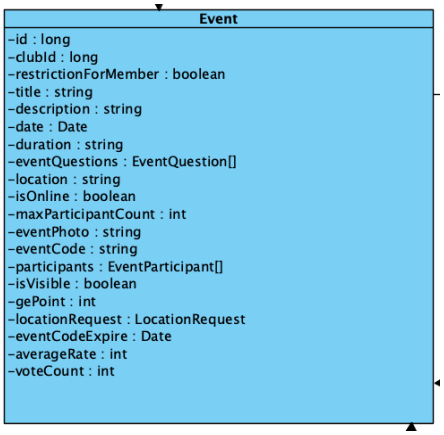

Fig. 59. Event.

Attributes:

private long id: This is the object's ID for tracing purposes in the database.

private long clubId: This is Club class' ID due to many to one relation.

private boolean restrictionForMember: This is the information that determines whether the event is open for non-members or not.

private String title: This is the name of the event.

private String description: This is the description for the event details.

private Date date: This is the date that the event will be held.

private String duration: This is the event's duration.

private EventQuestion[] eventQuestions: These are the questions that are asked to the club about the event.

private String location: This is the information that where the event will be held.

3.4.3.3 SurveyQuestion

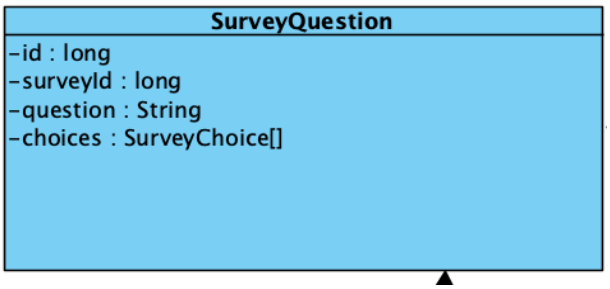

Fig. 60. Survey Question.

Attributes:

private long id: This is the object's ID for tracing purposes in the database.

private long surveyId: This is Survey class' ID due to many to one relation.

private String question: This is the question that is asked in the survey.

private SurveyChoice[] choices: This is the choice for the question.

3.4.3.4 SurveyChoice

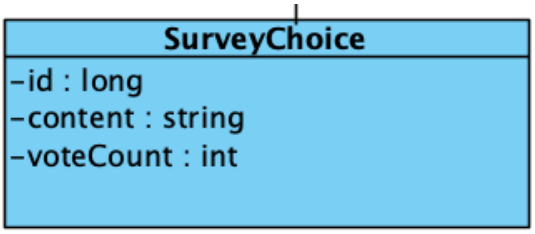

Fig. 61. Survey Choice.

Attributes:

private long id: This is the object's ID for tracing purposes in the database.

private String content: This is the content of the choice.

private int voteCount: This is the participant count that chose this choice.

3.4.3.5 SurveyParticipant

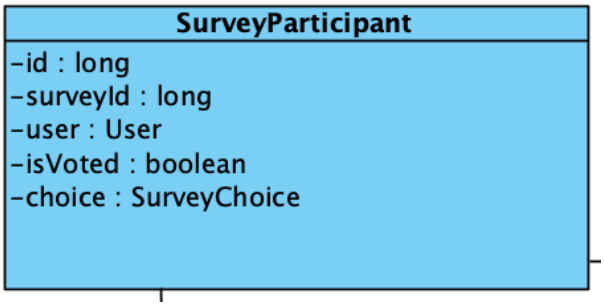

Fig. 62. Survey Participant.

Attributes:

private long id: This is the object's ID for tracing purposes in the database.

private long surveyId: This is Survey class' ID due to many to one relation.

private User user: This is the user that participated in the survey.

private boolean isVoted: This is the information that whether the student that can participate in the survey did vote or not.

private SurveyChoice choice: This is the choice of the student for the survey question.

3.4.3.6 Survey

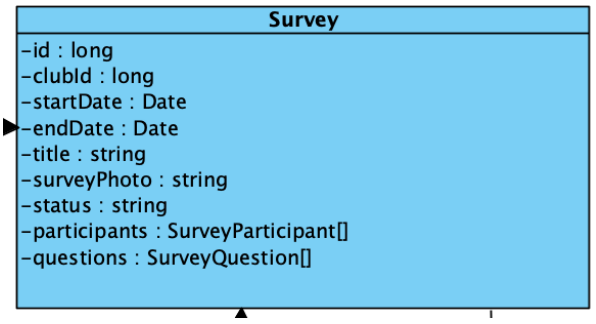

Fig. 63. Survey.

Attributes:

private long id: This is the object's ID for tracing purposes in the database.

private long clubId : This is Club class' ID due to many to one relation.

private Date startDate: This is the start date of the survey.

private Date endDate: This is the end date of the event.

private String title: This is the title of the survey.

private String surveyPhoto: This is the photo of the survey.

private String status: This is the information that determines whether the survey is active for the voting or not.

private SurveyParticipant[] participants: This is the participants of the survey.

private SurveyQuestion[] questions: These are the questions of the survey.

3.4.3.7 ClubSponsorship

Fig. 64. Club Sponsorship.

Attributes:

private long id: This is the object's ID for tracing purposes in the database.

private long clubId: This is Club class' ID due to many to one relation.

private String name: This is the name of the sponsor.

private String photo (string): This is the photo of the sponsor.

private int amount: This is the amount of the donation by the sponsor.

private ENUM[“GOLD”, “SILVER”, “BRONZ”] type: This is the type of the sponsor.

3.4.3.8 Club

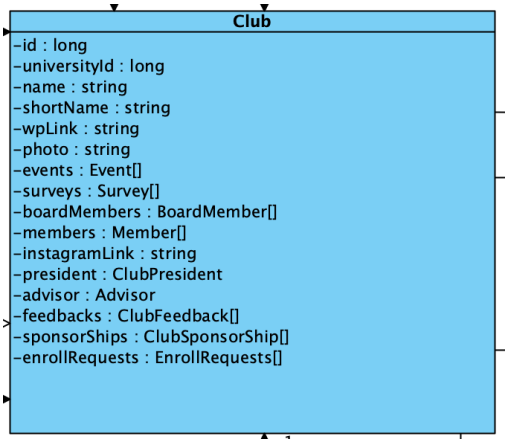

Fig. 65. Club.

Attributes:

private long id: This is the object's ID for tracing purposes in the database.

private long universityId: This is University class' ID due to many to one relation.

private String name: This is the name of the club.

private String shortName: This is the short name of the club.

private String wpLink: This is the Whatsapp link for the club group.

private String photo: This is the photo of the club.

private Event[] events: This is the events of the club.

private Survey[] surveys: This is the surveys of the club.

private BoardMember[] boardMembers: This is the board members of the club.

private Member[] members: This is the members of the club.

private String instagramLink: This is the Instagram link for the club page.

private ClubPresident president: This is the president of the club.

private Advisor advisor: This is the advisor of the club.

private ClubFeedback[] feedbacks: This is the feedbacks given for the club.

private ClubSponsorship[] sponsorships: This is the sponsors of the club.

private EnrollRequest[] enrollRequests: This is the enroll requests for the club.

3.4.3.9 EnrollRequest

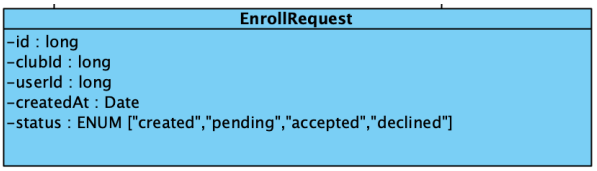

Fig. 66. Enroll Request.

Attributes:

private long id: This is the object's ID for tracing purposes in the database.

private long clubId: This is Club class' ID due to many to one relation.

private long userId: This is User class' ID due to many to one relation.

private Date createdAt: This is the date that the student requested for enrollment.

private ENUM[“created”, “accepted”, “declined”] status: This is the status of the enrollment.

3.4.3.10 Building

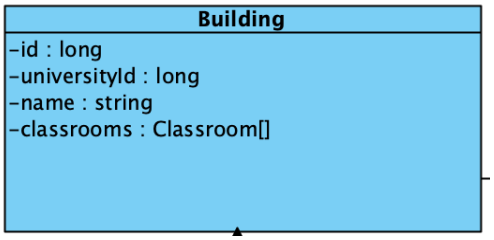

Fig. 67. Building.

Attributes:

private long id : This is the object's ID for tracing purposes in the database.

private long universityId : This is University class' ID due to many to one relation.

private String name: This is the name of the building.

private Classroom[] classrooms: These are the classrooms of the building.

3.4.3.11 Classroom

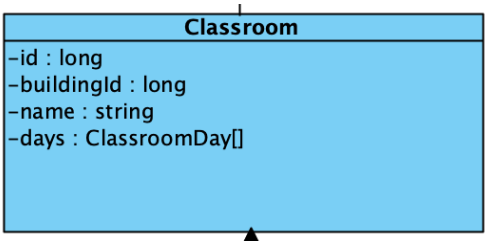

Fig. 68. Classroom.

Attributes:

private long id: This is the object's ID for tracing purposes in the database.

private long buildingId (long): This is Building class' ID due to many to one relation.

private String name: This is the name of the classroom.

private ClassroomDay[] days: This is the day that the classroom is available for an event.

3.4.3.12 ClassroomDay

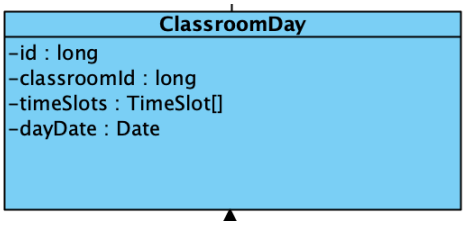

Fig. 69. ClassroomDay.

Attributes:

private long id : This is the object's ID for tracing purposes in the database.

private long classroomId: This is Classroom class' ID due to many to one relation.

private TimeSlot[] timeSlots: This is the time slot that the classroom is available.

private Date dayDate: This is the day that the available time slot belongs.

3.4.3.13 University

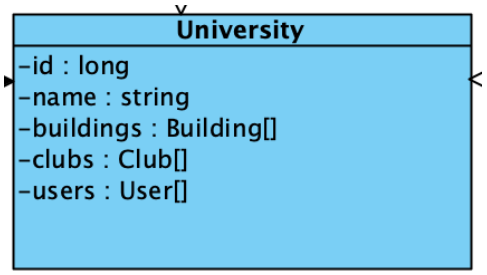

Fig. 70. University.

Attributes:

private long id: This is the object's ID for tracing purposes in the database.

private String name: This is the name of the university..

private Building[] buildings: These are the buildings that the university has.

private Clubs[] clubs : These are the student clubs that the university has.

private User[] users: This is the members of the university.

3.4.3.14 EventParticipant

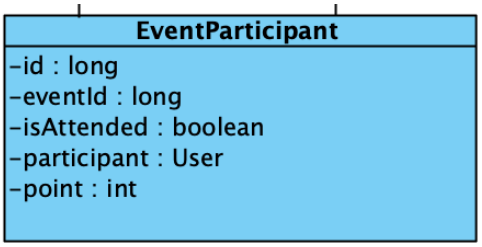

Fig. 71. Event Participant.

Attributes:

private long id : This is the object's ID for tracing purposes in the database.

private long eventId: This is Event class' ID due to many to one relation.

private boolean isAttended: This is the information that whether the student that registered for the event participated or not.

private User participant: This is the event participant user.

private int point: This is the point that the participant gave to the event.

3.4.3.15 ClubFeedback

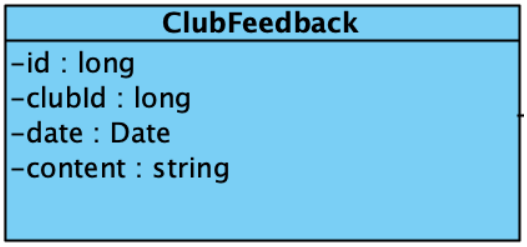

Fig. 72. Club Feedback.

Attributes:

private long id : This is the object's ID for tracing purposes in the database.

private long clubId: This is Club class' ID due to many to one relation.

private Date date: This is the date that the feedback was given.

private String content: This is the content of the feedback.

3.4.3.16 TimeSlot

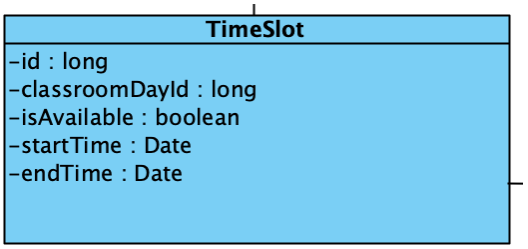

Fig. 73. Time Slot.

Attributes:

private long id : This is the object's ID for tracing purposes in the database.

private long classroomDayId: This is ClassroomDay class' ID due to many to one relation.

private boolean isAvailable: This is the information that determines whether the time slot is available or not.

private Date startTime: This is the start date of the time slot.

private Date endTime: This is the end date of the time slot.

3.4.3.17 LocationRequest

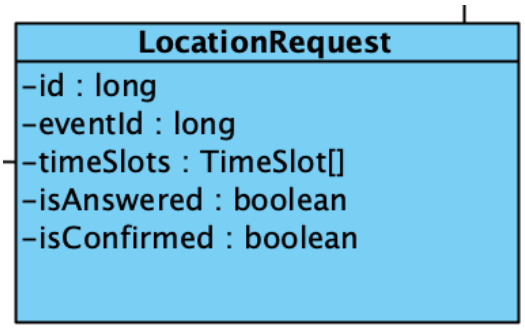

Fig. 74. Location Request.

Attributes:

private long id : This is the object's ID for tracing purposes in the database.

private long eventId: This is Event class' ID due to many to one relation.

private TimeSlot[] timeSlots: This is the desired time slot for the event.

private boolean isAnswered: This is the information that determines whether the request is answered or not.

private boolean isConfirmed: This is the information that determines whether the request is confirmed or not.

3.4.3.18 User

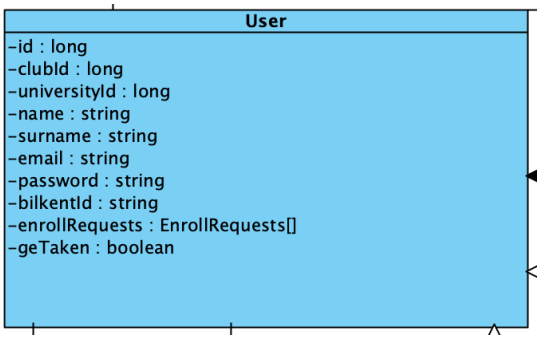

Fig. 75. User.

Attributes:

private long id: This is the object's ID for tracing purposes in the database.

private long clubId: This is Club class' ID due to many to one relation.

private long universityId: This is University class' ID due to many to one relation.

private String name: This is the name of the user.

private String surname: This is the surname of the user.

private String email: This is the email of the user.

private String password: This is the password of the user.

private String bilkentId: This is the Bilkent ID of the user.

private EnrollRequest[] enrollRequests: This is the user's enrollment requests for the clubs.

private boolean geTaken: This is the information that determines whether the user takes the GE250/251 course or not.

3.4.3.19 ClubRole

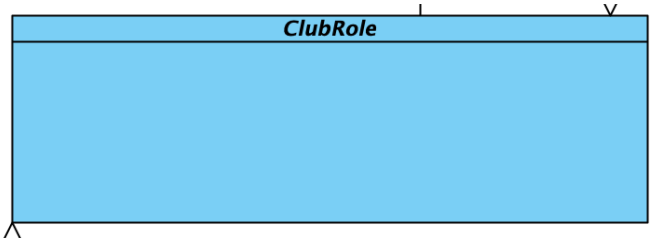

Fig. 76. Club Role.

This is the class of the club role.

Attributes: Empty

3.4.3.20 ClubAuthRole

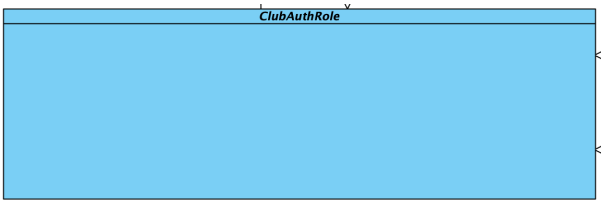

Fig. 77. Club Auth Role.

This is the class of the club authorization role.

Attributes: Empty

3.4.3.21 Member

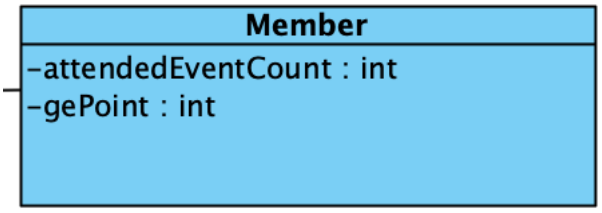

Fig. 78. Member.

Attributes:

private int attendedEventCount (): This is the event count that the member has attended.

private int gePoint (): This is the member's total GE250/251 points.

3.4.3.22 Admin

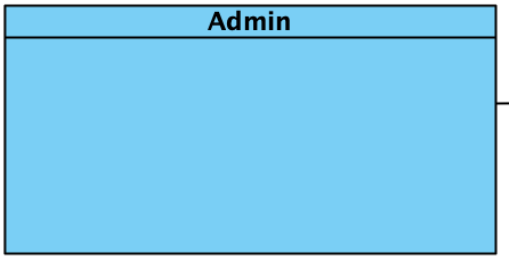

Fig. 78. Admin.

This is the class of the admin.

Attributes: Empty

3.4.3.23 AdministrativeAssistant

Fig. 79. Administrative Assistant.

This is the class of the administrative assistant.

Attributes: Empty

3.4.3.24 ClubPresident

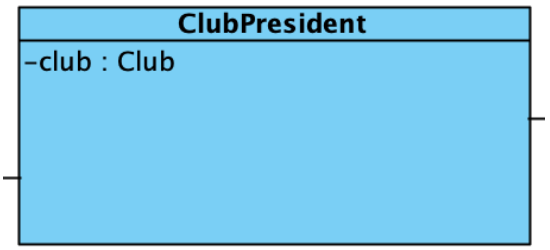

Fig. 80. Club President.

This is the class of the club president. It only has the club attribute that the user is president of.

3.4.3.25 BoardMember

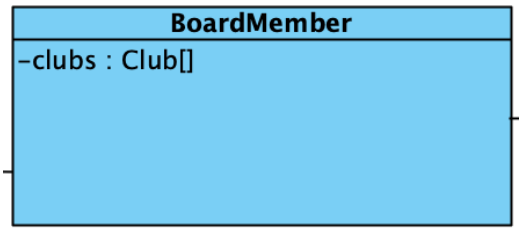

Fig. 81. Board Member.

This is the class of the club president. It only has the club list attribute that the user is a board member of.

3.4.3.26 Advisor

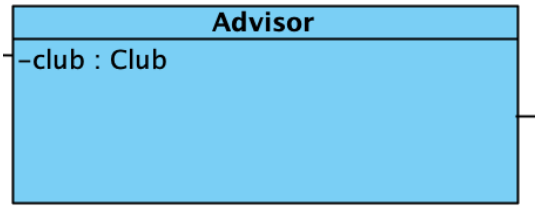

Fig. 82. Advisor.

Important Note: In each repository, there is findById(long id) method. It finds a unique object related entity.

3.4.3.27 BuildingRepository

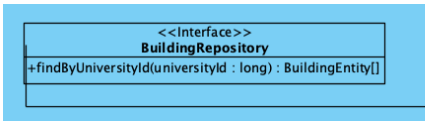

Fig. 83. Building Repository.

public Building findByUniversityId(long universityId): Find the building by university ID.

3.4.3.28 ClassroomRepository

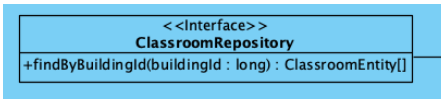

Fig. 84. Classroom Repository.

public Classroom findByBuildingId(long buildingId): Find the classroom by building ID.

3.4.3.29 ClassroomDayRepository

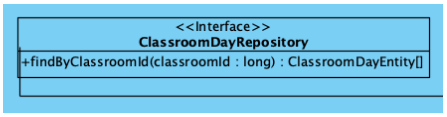

Fig. 85. Classroom Day Repository.

public ClassroomDay findByClassroomId(long classroomId): Find the classroom day by classroom ID.

3.4.3.30 TimeSlotRepository

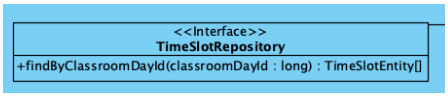

Fig. 86. Time Slot Repository.

public TimeSlot findByClassroomDayId(long classroomDayId): Find the time slot by classroom day ID.

3.4.3.31 LocationRequestRepository

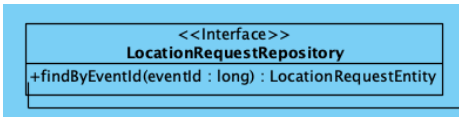

Fig. 87. Location Request Repository.

public LocationRequest findByEventId(long eventId): Find the location request by event ID.

3.4.3.32 LocationRequestTimeSlotRepository

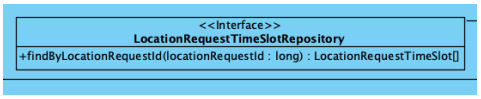

Fig. 88. Location Request Time Slot Repository.

public LocationRequestTimeSlot findByLocationRequestId(long locationRequestId): Find the location request time slot by location request ID.

3.4.3.33 ClubRepository

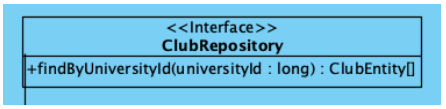

Fig. 89. Club Repository.

public Club findByUniversityId(long universityId): Find the club by university ID.

3.4.3.34 ClubSponsorshipRepository

Fig. 90. Club Sponsorship Repository.

public ClubSponsorship findByClubId(long clubId): Find the club sponsorship by club ID.

3.4.3.35 ClubMemberRepository

Fig. 91. Club Member Repository.

public ClubMember findByClubId(long clubId): Find the club member by club ID.

3.4.3.36 ClubBoardMemberRepository

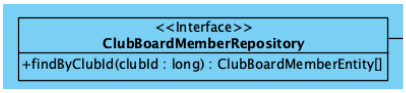

Fig. 92. Club Board Member Repository.

public ClubBoardMember findByClubId(long clubId): Find the club board member by club ID.

3.4.3.37 ClubFeedbackRepository

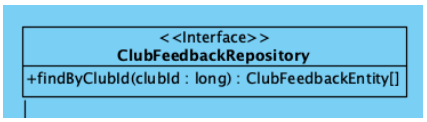

Fig. 93. Club Feedback Repository.

public ClubFeedback findByClubId(long clubId): Find the club feedback by club ID.

3.4.3.38 AcademicRepository

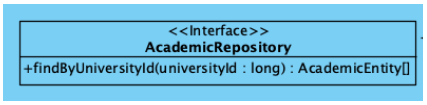

Fig. 94. Academic Repository.

public Academic findByUniversityId(long universityId): Find the academic by university ID.

3.4.3.39 EventRepository

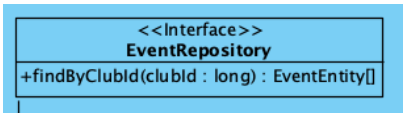

Fig. 95. Event Repository.

public Event findByClubId(long clubId): Find the event by club ID.

3.4.3.40 EventParticipantRepository

Fig. 96. Event Participant Repository.

public EventParticipant findByEventId(long eventId): Find the event participant by event ID.

3.4.3.41 EventQuestionRepository

Fig. 97. Event Question Repository.

public EventQuestion findByEventId(long eventId): Find the event question by event ID.

3.4.3.42 SurveyParticipantRepository

Fig. 98. Survey Participant Repository.

public SurveyParticipant findBySurveyId(long surveyId): Find the survey participant by survey ID.

3.4.3.43 SurveyRepository

Fig. 99. Survey Repository.

public Survey findByClubId(long clubId): Find the survey by club ID.

3.4.3.44 UniversityRepository

Fig. 100. University Repository.

It does not have any other method. It just consists of findById which comes from JPARepository.

3.4.3.45 SurveyChoiceRepository

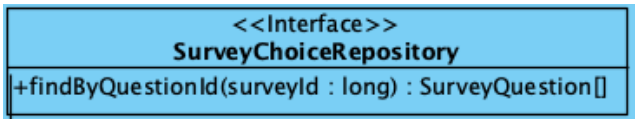

Fig. 102. Survey Choice Repository.

public SurveyChoice findBySurveyId(long surveyId): Find the survey choice by survey ID.

3.4.3.46 AdminRepository

Fig. 103. Admin Repository.

public Admin findByUniversityId(long universityId): Find the admin by university ID.

3.4.3.47 SurveyQuestionRepository

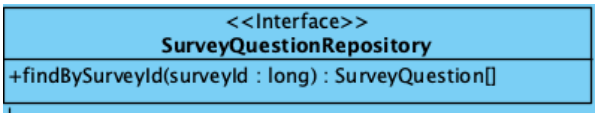

Fig. 104. Survey Question Repository.

public SurveyQuestion findBySurveyId(long surveyId): Find the survey question by survey ID.

3.4.3.48 AdministrativeAssistantRepository

Fig. 105. Administrative Assistant Repository.

public AdministrativeAssistant findByUniversityId(long universityId): Find the administrative assistant by university ID.

3.4.3.49 StudentRepository

Fig. 106. Student Repository.

public Student findByUniversityId(long universityId): Find the student by university ID.

3.4.3.50 EnrollRequestRepository

Fig. 107. Enroll Request Repository.

public EnrollRequest findByClubId(long clubId): Find the enroll request by club ID.

3.4.3.51 MemberRepository

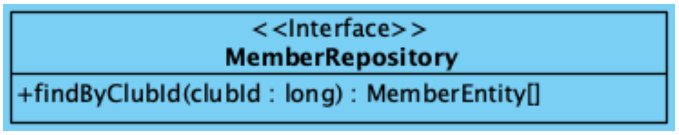

Fig. 108. Member Repository.

public Member findByClubId(long clubId): Find the member by club ID.

3.4.3.52 BoardMemberRepository

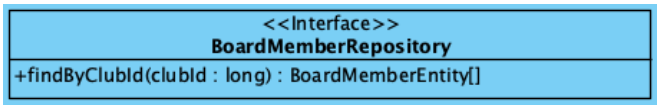

Fig. 109. Board Member Repository.

public BoardMember findByClubId(long clubId): Find the board member by club ID.

3.5 Design Patterns

3.5.1 Template Design Pattern

In our class diagram of the first iteration, it can be seen that we constructed the entire class diagram using required abstract classes. That is why we decided to use Template Design Pattern in our application which is a behavioral design pattern required to implement an abstract superclass and define the skeleton of an operation in terms of a number of high-level steps. In our app, there is AuthRole, functions of it should be implemented by classes that extend it.

3.5.2 Facade Design Pattern

In our class diagram, it can be seen that some of our classes need different functions from different classes and in order to accomplish this aim, we decided to use the Facade Design Pattern that enables us to call different classes' functions from a Facade Class. In our application, there are functions in the controller that call multiple functions from different classes.

4. Improvement Summary

This iteration of the design report has some improvements over the first iteration after feedback.

4.1 High-level Software Architecture

The title of the section was changed. Deployment diagram was added.

4.2 High-level Software Architecture - Subsystem Decomposition

Admin subsystems were added to the diagram. Subsystem explanations were added.

4.3 High-level Software Architecture - Persistent Data Management

The type of data that will be collected and stored was added.

4.4 High-level Software Architecture - Hardware-Software Mapping

System requirements were added.

4.5 High-level Software Architecture - Access Control and Security

Access control matrix was added.

4.6 High-level Software Architecture - Boundary Conditions

In the initialization part, actions taken to start the application in the production server were added.

4.7 Final Object Design

New classes in UI, web server and data management layers were added in final object design.

4.8 User Interface Layer

Admin screens and club members screens were added, meaning of associations was explained, and explanations of all UI classes were added.

4.9 Web Server Layer

Admin controller and service were added. Also, all classes, attributes, and methods were explained.

4.10 Data Management Layer

SurveyQuestionEntity and EnrollRequestEntity were added to the layer. EventPointEntity was removed. Some fields were changed in some tables. Association explanations were added. Also, all classes, attributes, and methods were explained.

5. Glossary & references

[1] “How much ram and CPU does my website need to function?,” *one.com*, 21-Sep-2021. [Online]. Available: https://www.one.com/en/hosting/ram-and-cpu. [Accessed: 18 Dec 2021].

[2] “What is Ram in website hosting? how much does it require ...” [Online]. Available: https://www.quora.com/What-is-RAM-in-website-hosting-How-much-does-it-require. [Accessed: 18 Dec 2021].

[3] "Introduction to JSON Web Tokens". [Online]. Available: https://jwt.io/introduction. [Accessed: 29 October 2021].

[4] "Cross-Origin Resource Sharing (CORS)". [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. [Accessed: 29 October 2021].

[5] K. Yadav, "What is AWS and What can you do with it". [Online]. Available: https://medium.com/@kunalyadav/what-is-aws-and-what-can-you-do-with-it-395b585b03c. [Accessed: 29 October 2021].

[6] "AWS Elastic Beanstalk". [Online]. Available: https://aws.amazon.com/elasticbeanstalk/?nc1=h_ls. [Accessed: 29 October 2021].

[7] B. Lutkevich , "Amazon RDS (Relational Database Service)". [Online]. Available: https://searchaws.techtarget.com/definition/Amazon-Relational-Database-Service-RDS. [Accessed: 29 October 2021].