Bilkent University

Department of Computer Engineering

CS319 Term Project

Fall 2022

Section 1

Group 1B

Design Report

Group members:

Ayşe Kelleci 21902532

Yusuf Doğan 21702970

Zeynep Hanife Akgül 22003356

Kardelen Ceren 22003017

Melisa Tanrıkulu 21703437

Instructor:

Eray Tüzün

Contents

1. Introduction

1.1. Purpose of the System

The purpose of the system is to create a user-friendly website where outgoing students and coordinators can access and handle their Erasmus-related tasks easily. Its difference from the current Erasmus website is, the communication between students and coordinators is done on the website instead of email, all document related tasks (e.g, generating and signing) are done digitally, and they can track their status from the website.

1.2. Design Goals

1.2.1. Reliability

The system will create documents (e.g. Pre-Approval Form) and enable coordinators to sign them digitally, so all files must be created and updated correctly. Also, the course selection is done through the system, hence the information of previously approved courses should be current and correct. Moreover, all users have a to-do list. This list must be up-to-date since the coordinators need to take action according to the contents of the list before the deadline.

1.2.2. Functionality

Our system has many features to ease any process during the Erasmus period. To accomplish this, we included functionalities such as posting on forum, status viewing, to-do list, digitizing the form handling, enabling communication between students and coordinators etc. Also, since the form handling is done through our system, we will enable creating forms automatically and revising them by board members.

1.3. Trade-Offs

1.3.1. Rapid development vs. Functionality

Since we are trying to make our system as functional as possible, the development process is expected to take a long time. We included as much functionality as possible to enhance productivity by making the process easy and quick, so compromise from time is inevitable.

1.3.2. Usability vs. Functionality

Having functionality as one of our main design goals has some drawbacks on usability. More usable is more simple. That is to say, if we compare the usability of more functional and less functional software in which other aspects are equal, we would find less functional one more usable. To narrow this idea to our Erasmus Application software, having only course related operation in the app is usable, however, we trade off usability with functionality by integrating additional features to our software such as, forum, direct messaging, ToDo.

2. System Architecture

2.1. Subsystem Decomposition

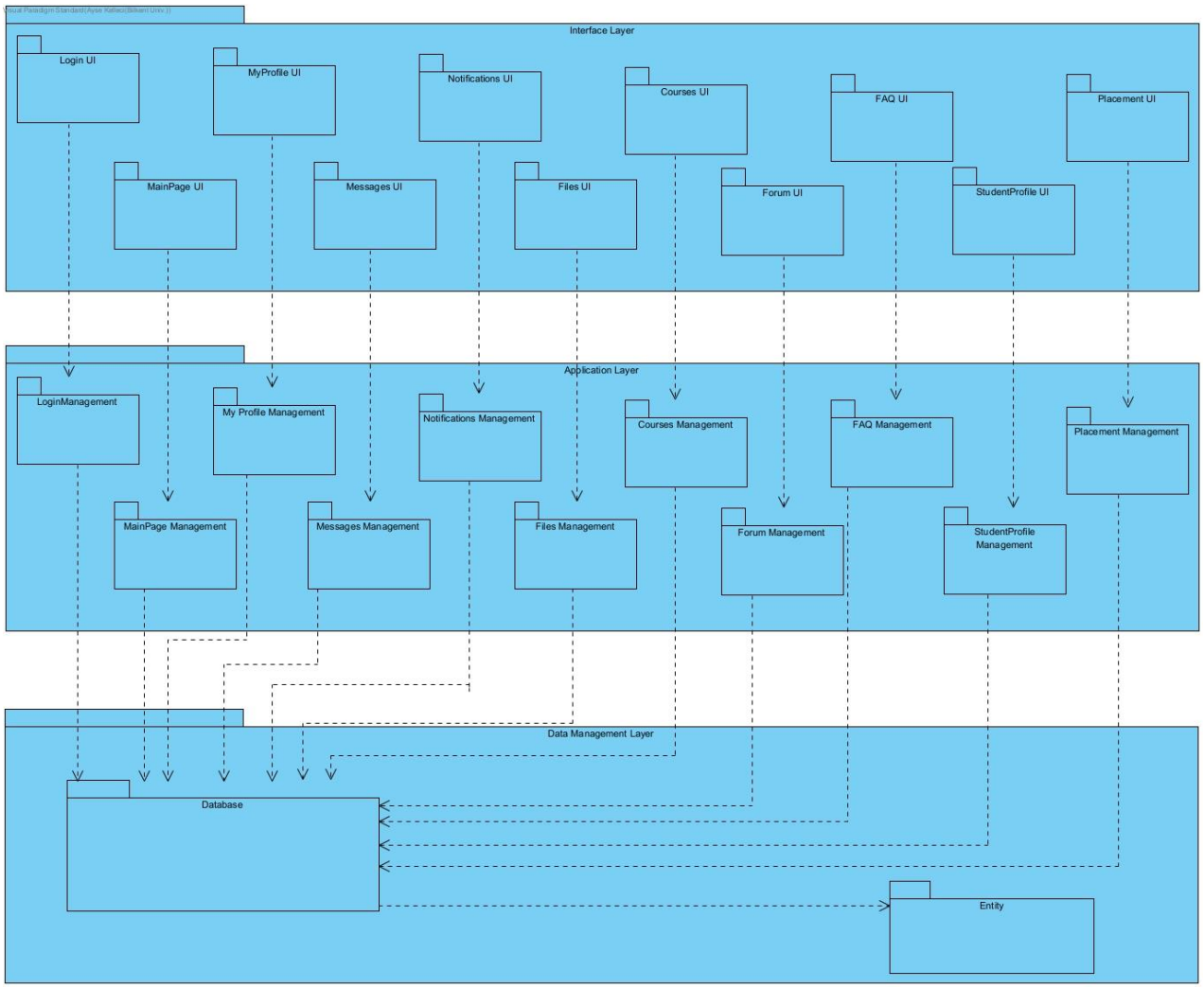


Fig. 2.1.1: Subsystem Decomposition of the System

In this part, we will describe our style of decomposition into subsystems, the reason why we chose it, and the responsibilities of each subsystem. During the decomposition, our goal was to reduce the complexity while facilitating the modification of the system. We decided that three-layer architectural style suited our purpose the best, because of the following reasons:

- Each actor has different functionalities that are assigned to them, so different interfaces for each actor are required. This architectural style enables the development and modification of such for the same application logic.

- Each layer can run on separate server platforms, so in case of failure/customization of any server, the others won't be affected. This property is significant as there will be a lot of essential data in our system that must not be lost.

We will now describe the responsibilities of each subsystem. As mentioned above, there are three layers: Interface Layer, Application Layer and Data Management Layer. We tried to decompose the systems so that the high coherence of each class and low coupling between each layer is achieved. The layers are as follows:

2.1.1. Interface Layer

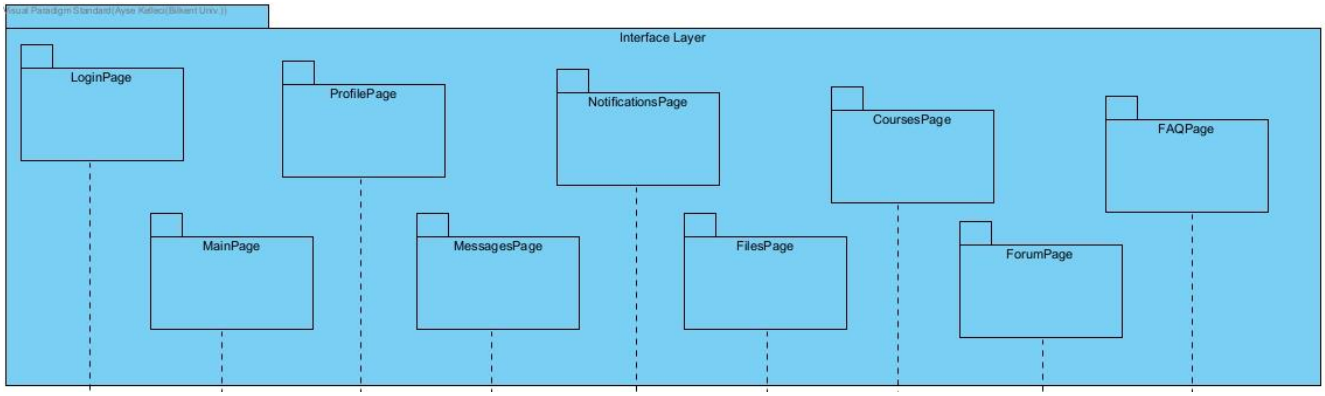


Fig. 2.1.2: Interface Layer of the Subsystem Decomposition of the System

The interface layer is created for every interface a user interacts with. This layer acts like a boundary object as every interface has certain elements that allow actions like clicking buttons or receiving inputs like typing text to corresponding areas. As interfaces tend to be changed often, creating this layer by separating it from the whole project enables quick adaptations without changes in application and data layers.

2.1.2. Application Layer

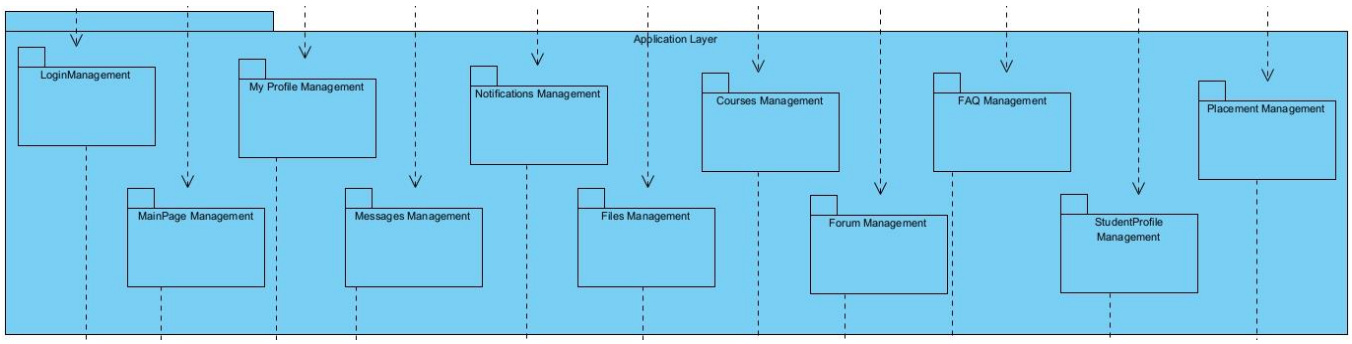


Fig. 2.1.3: Application Layer of the Subsystem Decomposition of the System

The application layer consists of management systems and logic operations, which are handled within back-end development. Every subsystem in this layer corresponds to a function in the application. As the application layer consists of controllers, it acts as a controller object that is dependent on the data layer.

2.1.3. Data Management Layer

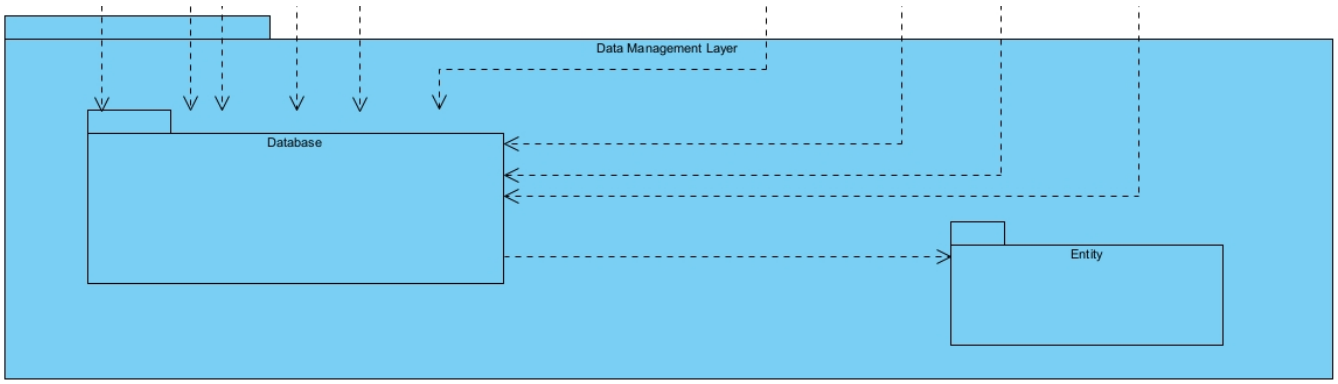


Fig. 2.1.4: Data Management Layer of the Subsystem Decomposition of the System

Data management layer handles the storage of information as well as retrieving them. Database and entity are two subsystems of this layer which contain repository interfaces and persistent entity classes, respectively.

Consequently, our system is extensible and flexible as we have decomposed it into three layers. Possible changes in a layer will not affect others which increases reliability.

2.2. Hardware/Software Mapping

Our system does not require any specialized hardware components to run successfully. The system is web-based; therefore, the hardware systems are expected to run a web browser. The followings are the minimum requirements for a computer to be able to use the system:

- 1.8 GHz x86 or x64 bit processor,
- 30 GB HDD,
- 512 MB RAM, and
- Stable internet connection with a minimum bandwidth of 50 KB/s.

We will use a variety of libraries in the development process and since some browsers or their older versions do not support all of them, we need to specify which versions are able to support them. The followings are the minimum required versions of browsers to be able to use our system:

- Google Chrome 23
- Internet Explorer 10
- Safari 6
- Opera 15
- Mozilla Firefox 21

2.3. Persistent Data Management

For data management, we will use the PostgreSQL database. PostgreSQL database is an open-source relational database management system. It is convenient to implement an object-oriented web application with the aid of relational systems since they perform better when the number of relations between data tables increases. In our project, we have a lot of dependencies and relations. Therefore, we prefer to use the relational database management system. Moreover, PostgreSQL is the most preferred database system for Django because they are compatible with each other, and connecting them is easy. Since it is a common way, there are also many resources on the web.

2.4. Access Control and Security

As well as security is provided with a secure database PostgreSQL, restraints in user views also support privacy and security. Access control is made based on the user type, and corresponding interfaces and information are displayed to users only if they are allowed to reach out.

	Student	Coordinator	Board Member
Log In	x	x	x
Add Course	x		
Post on Forum	x	x	x
Reply a Post	x	x	x
View Post Replies	x	x	x
View FAQ	x	x	x
Add FAQ		x	
Approve Course		x	x
Direct Message	x	x	x
Message View	x	x	x
Edit Profile	x	x	x
Add ToDo	x	x	x
View User's Profile	x	x	x
View Student List		x	x
Upload File	x	x	x
Download File	x	x	x
Change Password	x	x	x

Table 2.4.1: Access Control Matrix

2.5. Boundary Conditions

2.5.1. Initialization

Our system does not require any installation since it runs on a web server. In order to log in to the system, there must be an account assigned to the user in the database by administrators; otherwise, they can only see login and forgot password. If a user is logged in, the same account can also log in from other devices as well. When the user is logged in, all information on the page is initialized from the database.

2.5.2. Termination

If one of the subsystems is terminated, then the whole application terminates as well since the system works as a whole structure. If an admin initiates the termination process, in order to accomplish zero data loss, all data is saved to the database. When a user wants to exit from the system, they can click the button “Log out” from the right top of the screen.

2.5.3. Failure

Due to performance issues, we don’t have any case for unexpected crashes. The current data will be lost and none of the functions will work. However, a pop-up will be displayed to the user to explain the situation and the developers will receive a notification through email about the failure.

3. Low-Level Design

3.1. Final Object Design

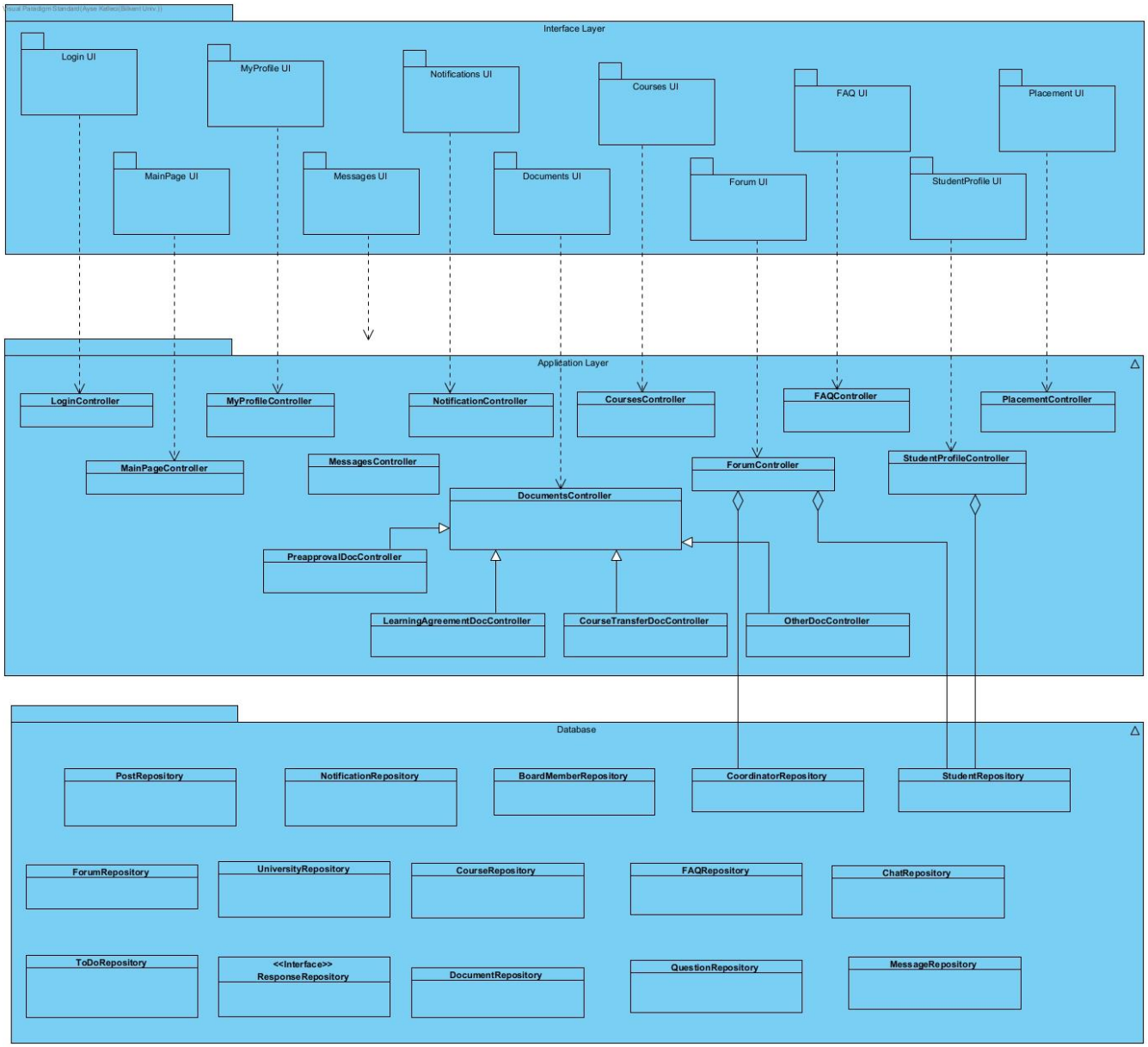


Fig. 3.1.1: Final Object Design Diagram of the System

In the Database layer; studentRepository, coordinatorRepository and boardMemberRepository are connected to each controller (except ForumController and StudentProfileController as shown above) with an aggregation relationship, but for simplicity purposes, not every link is shown in the above diagram.

Each repository is connected to corresponding entities but again for simplicity, the linkages are not shown.

3.2. Application Layer

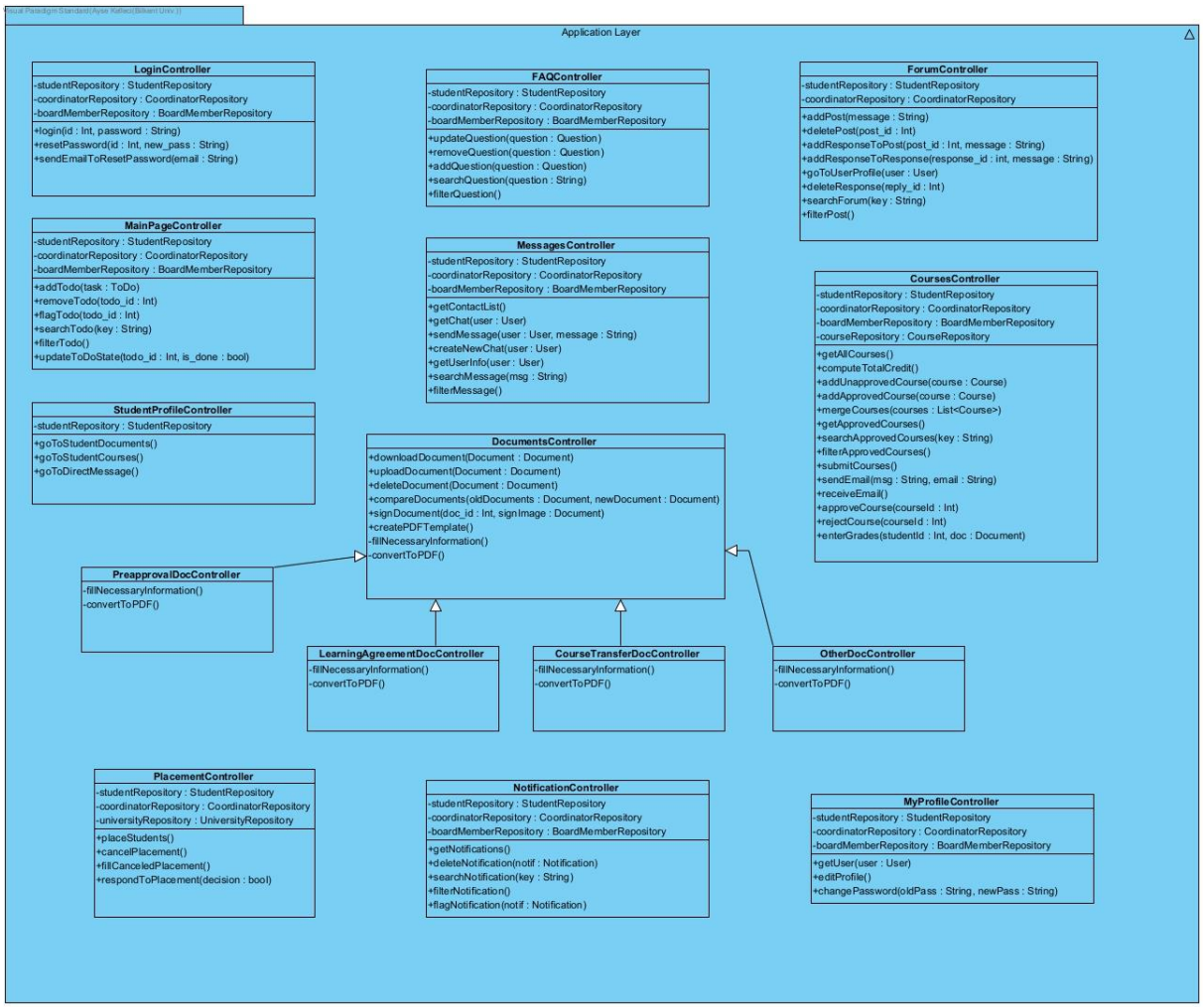


Fig. 3.2.1: Application Layer of the Subsystem Decomposition of the System

Application layer handles the linking of users' input with data management. When necessary, required information is taken by the user to be returned to the data management layer.

3.3. Data Management Layer

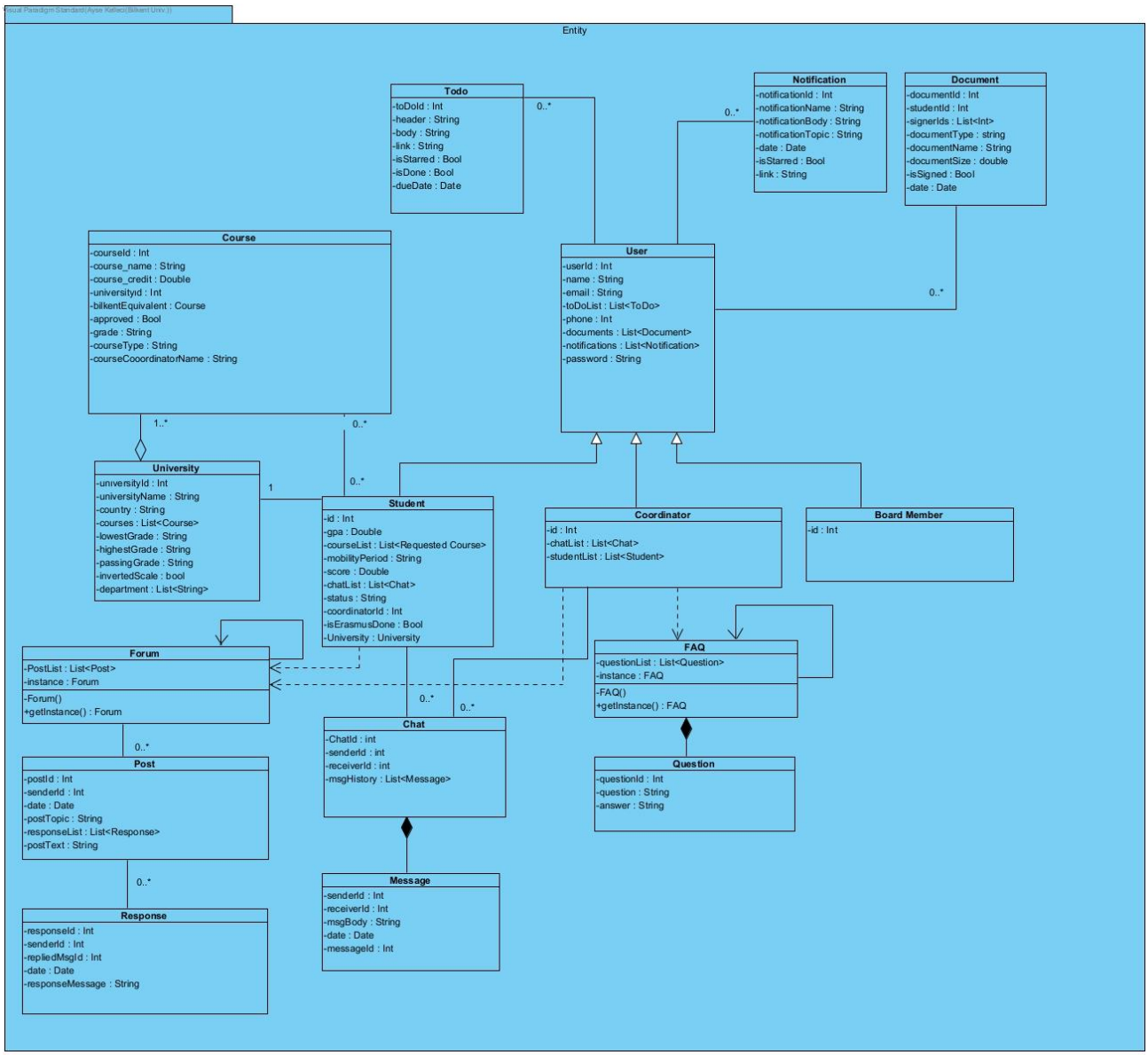


Fig. 3.3.1: Entity Substytem of the Data Management Layer

Each subsystem in Entity has its own constructors, getter, and setter methods but for simplicity, they are not explicitly written in the above diagram of Entity.

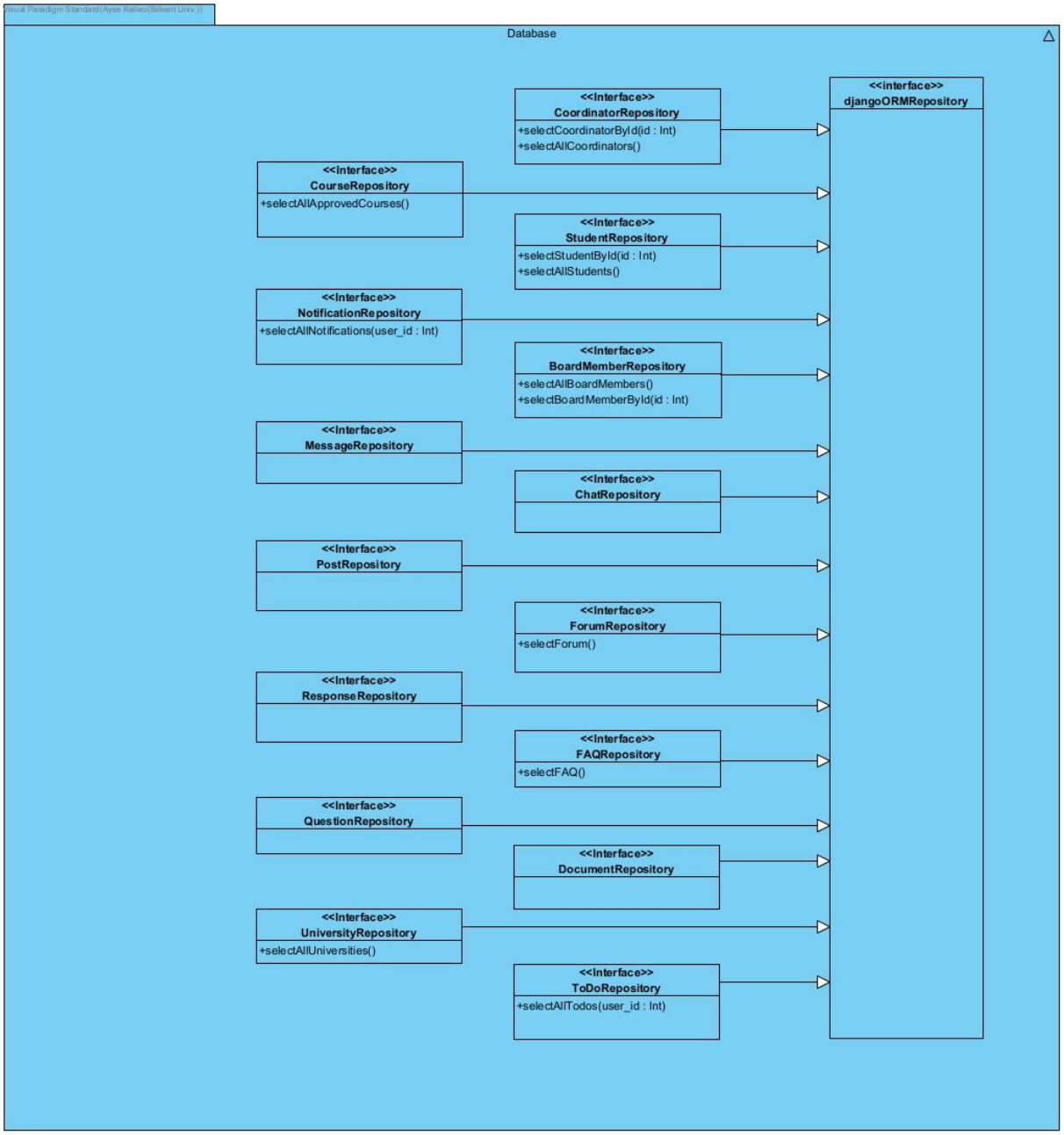


Fig. 3.3.2: Database Substytem of the Data Management Layer

Data Management layer consists of two subsystems and the database is one of them. This subsystem has repositories for specific categories of data which inherit from djangoORMRepository. The repositories are separated according to which data they handle for a better readability and less complexity.

3.4. Design Patterns

3.4.1. Singleton Design Pattern

Singleton Design ensures that a certain class has only one instance and provides a global access point to that instance [2]. This pattern is used in FAQ and Forum classes as seen in Entity. Usage of only one instance occupies less memory while making the code less complex thus, more readable.

3.4.2. Template Design Pattern

Template design pattern suggests defining a system as the skeleton of operations and leaving the implementation part to child classes. The overall structure is created by the parent classes [3]. The reflection of template design in this project is in the application layer, documentsController class defines fillNecessaryInformation() and convertToPDF() methods as skeletons which are overridden by its child classes. createPDFTemplate() in documentsController class is the caller of these methods.

3.5. Packages

3.5.1. Controller Package

It contains all the controller classes. It also handles interaction between server and other packages.

3.5.2. Interface Package

In the interface part we do not use any package or library. The user interface will be produced by using HTML, CSS and JavaScript.

3.5.3. Entity Package

It is a package that includes all entity objects such as student objects or a course object. Entity package can only interact with controller package.

3.5.4. Database Package

The database package will keep a database class that handles the storage of information as well as retrieving them. The package interacts with controllers for receiving data to be stored and retrieving data to controller operations.

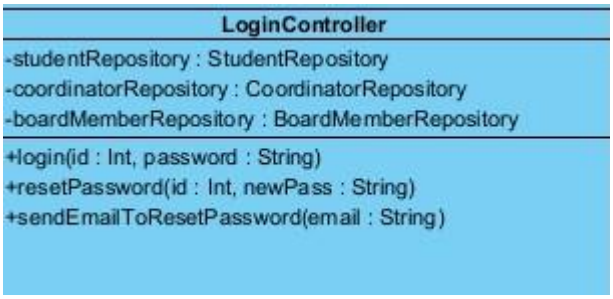
3.5.5. External Package

3.5.5.1. **Django:** It is a fully featured server-side web framework written in Python, which we utilize in our software.

3.6. Class Interfaces

3.6.1. Application Layer Interfaces

3.6.1.1. LoginController



A class that controls log-in operations.

Operations:

public login(int id, String password):

This operation checks if the given ID and password match with a user on the database and if so, returns the user.

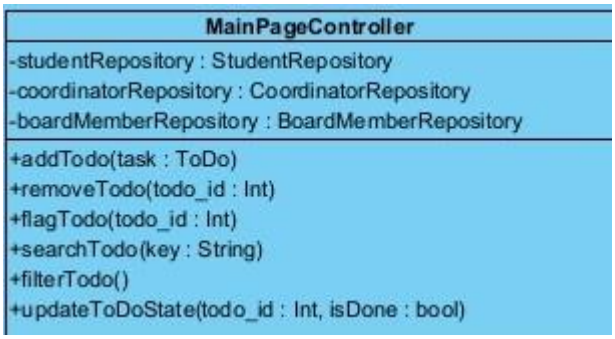
public resetPassword(int id, String newPass):

This operation renews the password with the new password that comes through user email.

public sendEmailToResetPassword(String email);

This operation sends an email to the user for reset password request.

3.6.1.2. MainPageController



A class that controls operations related to the to-do list and the status shown on the main page.

Operations:

public getStatus():

This operation gets the status of the user from the database.

public getTodoList():

This operation gets the todo list of the user from the database.

public addTodo(ToDo task):

This operation adds a todo task to the user's todo list.

public removeTodo(int todo_id):

This operation removes a todo task from the user's todo list.

public flagTodo(int todo_id):

This operation flags a todo task in the user's todo list.

public searchTodo(String key):

This operation searches for todo tasks in the user's todo list.

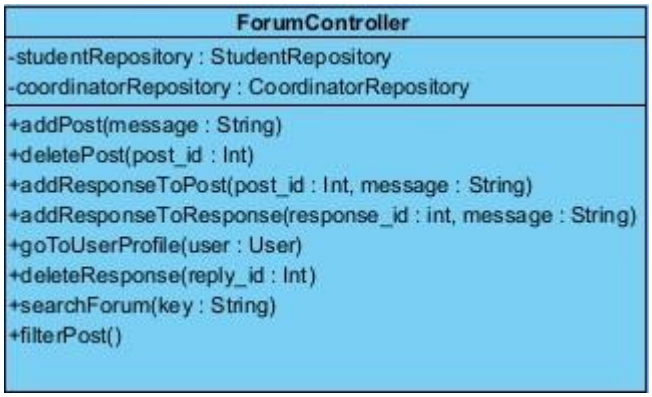
public filterTodo():

This operation filters todo tasks.

public updateToDoState(int todo_id, bool isDone):

This operation updates the todo task's state (whether it's done or not)

3.6.1.3. ForumController



A class that controls forum related operations.

Operations:

public getAllPosts():

This operation gets all the posts in the forum.

public getPostDetail(int post_id):

This operation gets details post from given post id

public addPost(String message):

This operation creates post heading

public deletePost(int post_id):

This operation remove the post heading with the responses

public addResponseToPost(int post_id, String message):

This operation respond to post

public addResponseToResponse(int response_id, String message):

This operation respond to response of a post

public goToUserProfile(User user):

This operation leads to user´s profile

public deleteResponse(int reply_id):

This operation remove the response made to a post or a response

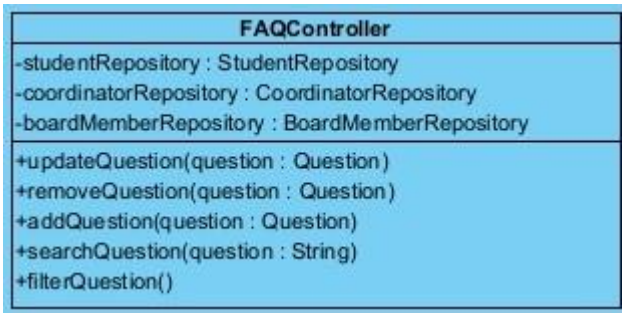
public searchForum(String key):

This operation search the given key in the entire forum

public filterPost()

This operation enable users to filter posts

3.6.1.4. FAQController



Operations:

public getQuestionList():

This operation gets all questions in FAQ

public updateQuestion(Question question):

This operation edits question contents

public removeQuestion(Question question):

This operation removes question

public addQuestion(Question question):

This operation adds question FAQ

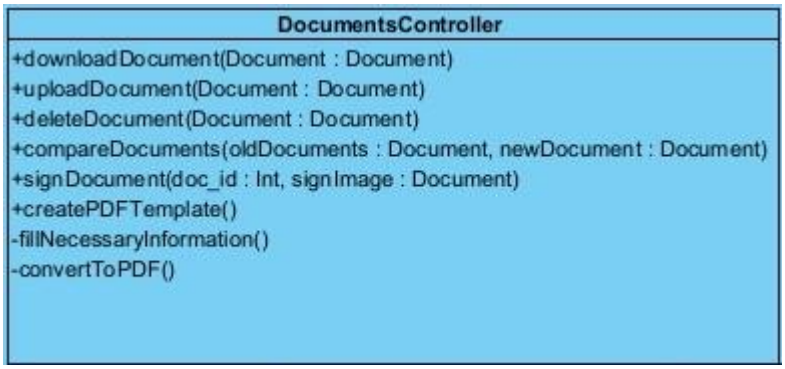
public searchQuestion(String question):

This operation searches questions by given string

public filterQuestion()

This operation is used to filter questions

3.6.1.5. DocumentsController



Operations:

public getAllDocuments():

This operation gets all documents

public downloadDocument(Document document):

This operation enable document downloading

public uploadDocument(Document document):

This operation enable document uploading 7

public deleteDocument(Document document):

This operation remove the document from user´s files section

public compareDocuments(Document oldDocument, Document newDocument):

This operation is used to compare two documents

public signDocuments(int documentId, Document signImage):

This operation puts sign on the document

public createPDFTemplate():

This operation creates pdf template of three type of document (learning agreement, course transfer, pre approval)

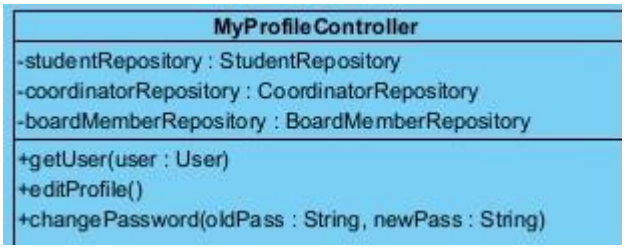
public fillNecessaryInformation():

This operation fills necessary user information in the template

public convertToPDF

This operation is used to convert document to pdf

3.6.1.6. MyProfileController



This class enable user to control his/her own profile

Operations:

public getUser(User user):

This getter operation receives the user.

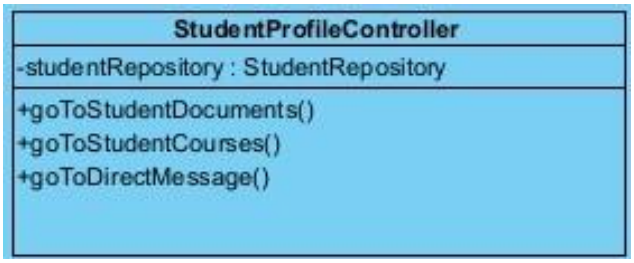
public editProfile():

This operation enables users to edit profiles.

public changePassword(String oldPass, String newPass):

This operation changes password of the user.

3.6.1.7. StudentProfileController



A class for all users to control a student profile

Operations:

public goToStudentFiles():

This operation leads user to student files.

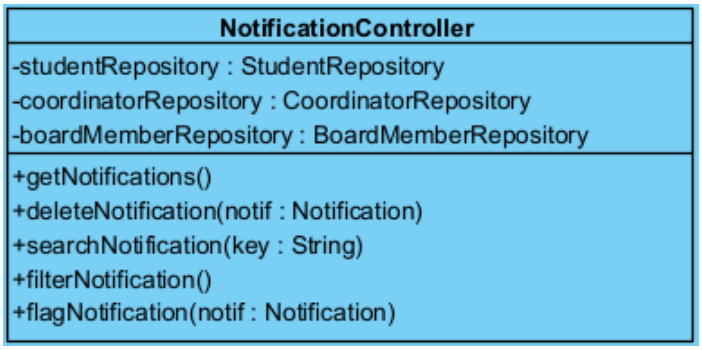
public goToStudentCourses():

This operation leads user to student´s courses.

public goToDirectMessage():

This operation leads user to direct messages

3.6.1.8. NotificationController



Operations:

public getNotifications():

This operation gets all notifications

public deleteNotification(Notification notif):

This method deletes notification

public searchNotification(String key):

This operation searches notification

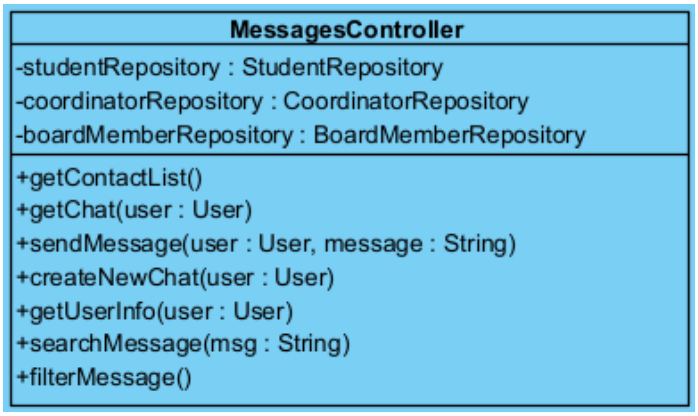
public filterNotification():

This operation filters notification

public flagNotification(Notification notif):

This operation is used to flag notification

3.6.1.9. MessagesController



Operations:

public getContactList():

This operation gets the user's contact list.

public getChat(User user):

This operation gets the whole chat with the received user parameter.

public sendMessage(User user, String message):

This operation is used to send message to other users in the existing chat.

public createNewChat(User user):

This operation is to create a new chat with a new user.

public getUserinfo(User user):

This method gets user info

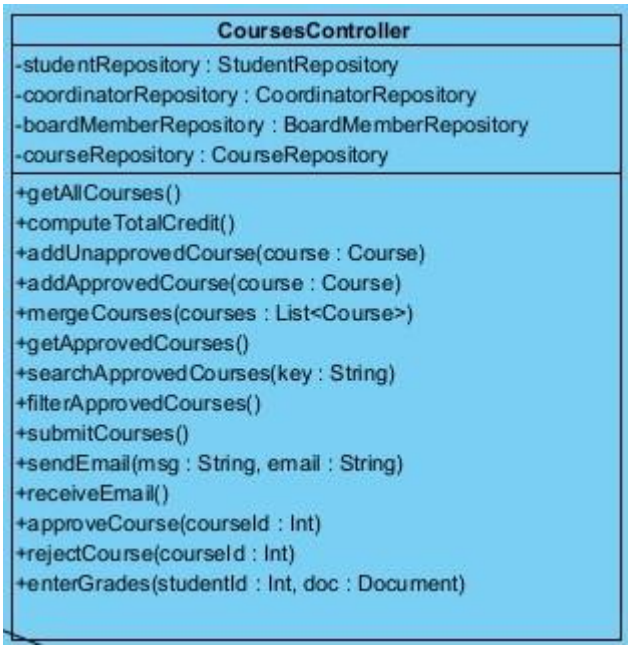
public searchMessage(String msg):

This operation searches a specific word/words in messages.

public filterMessage():

This operation filters our messages.

3.6.1.10. CoursesController



Operations:

public getAllCourses():

This operation gets all the courses in the system.

public compute TotalCredit():

This operation computes total credits of added courses.

public addUnapprovedCourse(Course course):

This operation adds an unapproved course.

public addApprovedCourse(Course course):

This operation adds an approved course.

public mergeCourses(List<Course> courses):

This operation merge courses that have less credits

public getApprovedCourses():

This operation gets previously approved courses.

public searchApprovedCourses(String key):

This operation is used to do search courses by a keyword.

public filterApprovedCourses():

This operation filters approved courses.

public submitCourses():

This operation submits added courses.

public sendEmail(String msg, String mail): This operation is used to send mail to student on course related contents

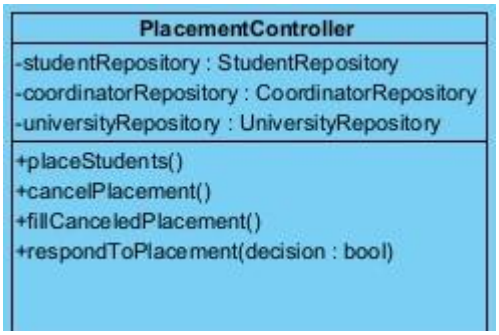
public receiveEmail(): This method receives email from corresponding mail addresses of faculty members on approval of submitted courses.

public approveCourse(int courseId): With this operation added courses is approved

public rejectCourse(int courseId): With this operation added courses is rejected

public enterGrades(int studentId, Document doc): This operation lets the coordinator enter grades to student documents.

3.6.1.11. Placement Controller



A class to control placement related operations

Operations:

public placeStudents():

This operation places students according to their preferences and scores

public cancelPlacement():

This operation cancels assigned placement.

public fillCancelledPlacement():

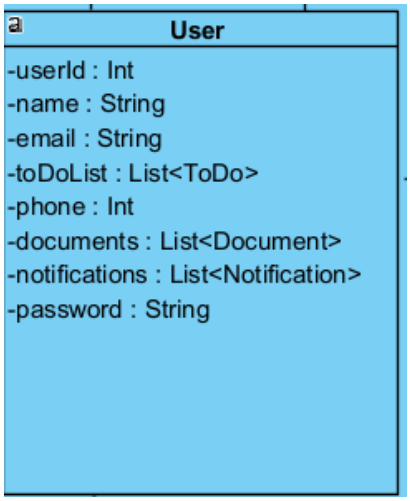
This operation fills canceled placements with the next student

public respondToPlacement(bool decision):

This operation responds yes/no to placement.

3.6.2. Data Management Layer Interfaces

3.6.2.1. Entity Interfaces



This class represents any registered user.

Attributes:

private int userId: ID associated with the user in the database

private String name: Full name of the user

private String email: email address of user

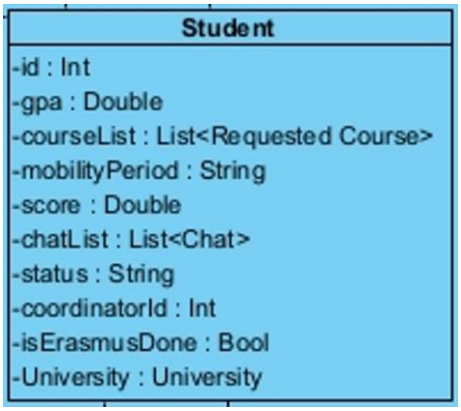
private List<ToDo> toDoList: List of user´s ToDo objects in database

private int phone: phone number of user

private List<Document> documents: List of user´s document objects in database

private List<Notification> notifications: List of user´s Notification objects in database

private String password: unique password of the user



This class represents a student user.

Attributes:

private int id: The Bilkent student id of the user (not the database id).

private double gpa: The cumulative GPA of the student.

private List<Requested Course> courseList: The list of courses the student is planning to take or taking.

private String mobilityPeriod: The mobility period of the student.

private double score: Total score of the student for placement purposes.

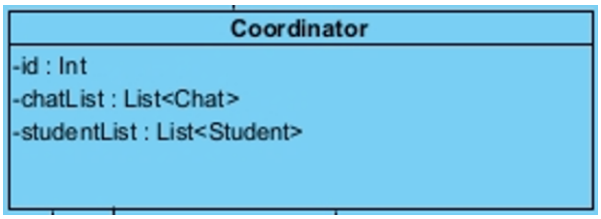
private List<Chat>chatList: All the chats of the student.

private String status: The status of the student.

private int coordinatorId: The id of the student's coordinator.

private bool isErasmusDone: A boolean variable for whether the student has completed their mobility period or not.

private University university: The university the student is placed in.



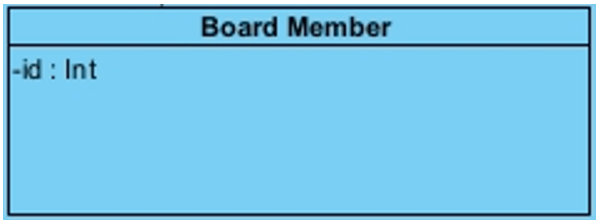
A class that represents the coordinator type of user.

Attributes:

private int id: Bilkent id of the coordinator

private List<Chat> chatList: List of chats the coordinator has represented as Chat objects.

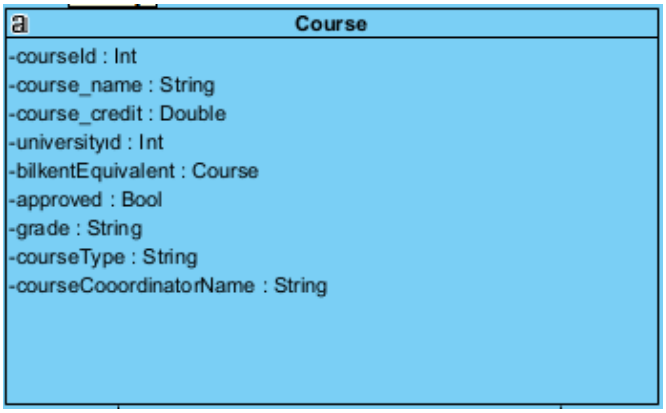
private List<Student> studentList: List of students represented as Student objects.



This class represents a faculty board member user.

Attributes:

private int id: Bilkent id of the board member.



This class represents courses in foreign universities.

Attributes:

private int courseId: Database id of the course

private String course_name: The name of the course, such as

private double course_credit:

private int universityId: Database id of the university student is going

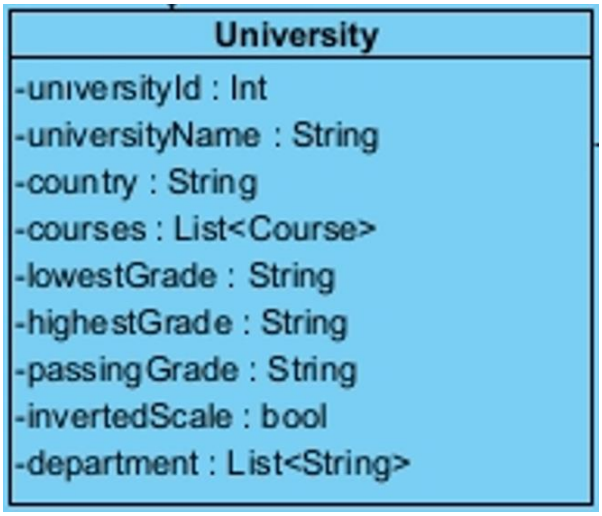
private Course bikentEquivalent:

private bool approved: A boolean variable for whether the course is approved by the coordinator or not.

private String grade: The grade the student received.

private String courseType: The course type, such as mandatory or elective.

private String courseCooordinatorName: The course coordinator's name for Bilkent University courses.



A class that indicate how universities stored in the system

Attributes:

private int universityId: Database id of the university

private String universityName: Name of the university as string

private String country: Name of the country in which the university is

private List<Course> courses: The list of approved courses in that university

private String lowestGrade: The lowest grade for the course

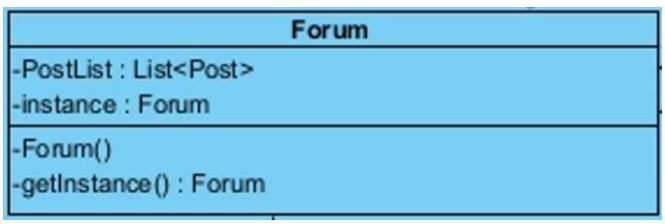
private String highestGrade: The highest grade for the course

private String passingGrade:

The minimum grade to pass the course

private bool invertedScale: It shows whether scale is ordered from the highest grade to the lowest grade, or the opposite

private List<String> department:



Class that represents Forum

Attributes:

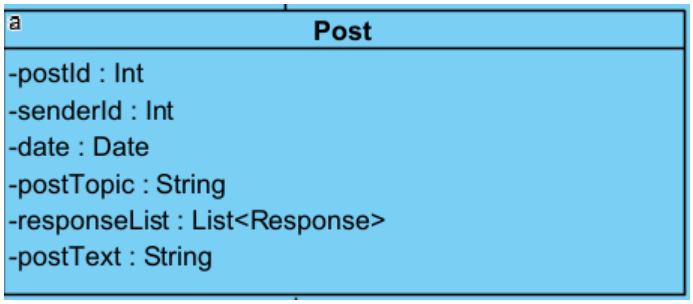
private List<Post> PostList: List of posts in the Forum represented as Post objects

private Forum instance: This attribute holds one instance of this class, in line with the Singleton design pattern.

Operations:

private Forum(): This private constructor initializes the one Singleton instance of this class.

public Forum getInstance(): This operation returns the one Singleton instance of this class.



Class that represents a single post on Forum

Attributes:

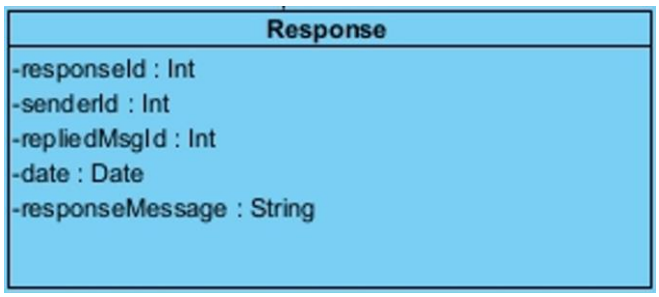
private int postId: Database id of a single post

private int senderId: Database id of the user who posts the post

private Date date: Date object representing the date on which the post is posted on the forum

private String postTopic: The subject of the post

private List<Response> responseList: A list consist of Response objects which represent replies to posts



Class that represents response to a post on Forum

Attributes:

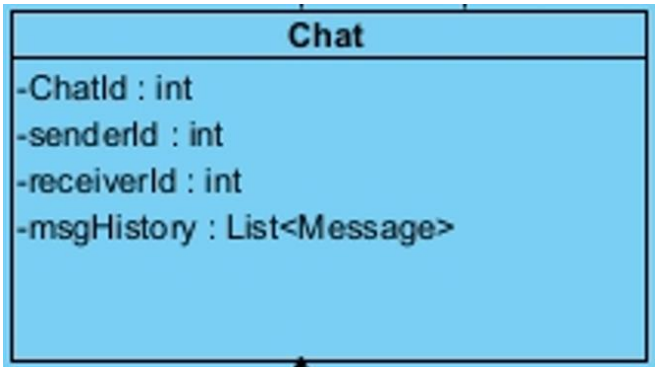
private int responseId: Database id of the response

private int senderId: Database id of the user who posted a reply

private int repliedMsgId: Database id of the post that is being replied with a Response object

private Date date: Date object representing on which date the reply is posted

private String responseMessage: Text body of the response



Class that represent a single chat

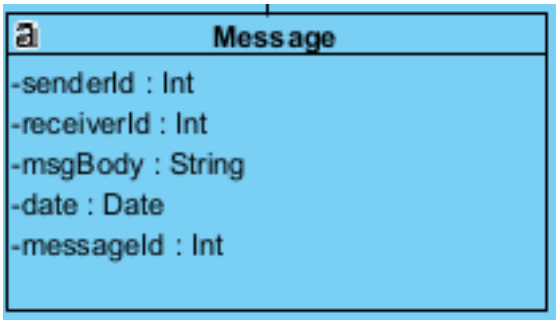
Attributes:

private int ChatId: Database id of the chat

private int senderId: Database id of the user who is the sender side of the chat.

private int receiverId: Database id of the user who is the receiver side of the chat

private List<Message> msgHistory: A list consisting of Message objects in the chat



A class that indicates messages in the system

Attributes:

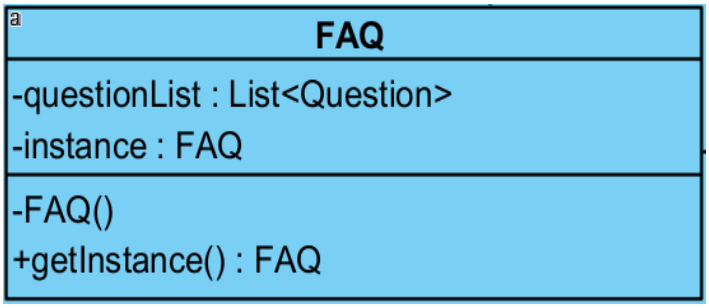
private int senderId: The id of the message sender

private int receiverId: The id of the message receiver

private String msgBody: String containing message content

private Date date: The date message is sent

private int messageId: Database id of the message



A class that illustrates FAQ system

Attributes:

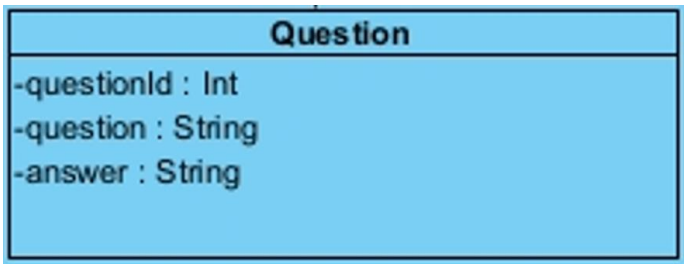
private List<Question> questionList: The list of question objects that will be stored in database

private FAQ instance: It is a single FAQ object that will be stored in database

Operations:

private FAQ(): This private constructor initializes the one Singleton instance of this class.

public FAQgetInstance(): This operation returns the one Singleton instance of this class.



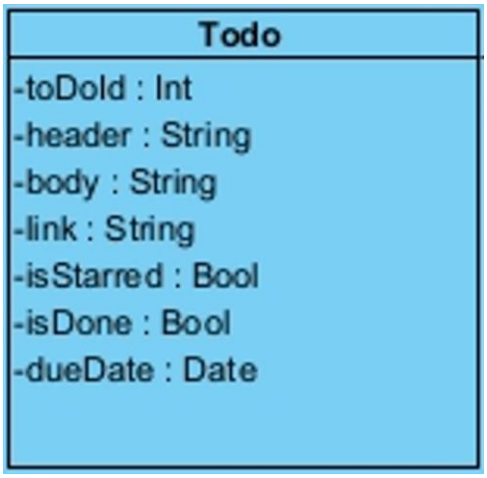
A class that represents Question of FAQ in the system.

Attributes:

private int questionId: unique id for a question to store in database

private String question: String containing what is the question

private String answer: String containing answer of the question



A class that indicate ToDo in the system

Attributes:

private int toDoInt: Unique ToDo id store in database

private String header: String including the title of ToDo

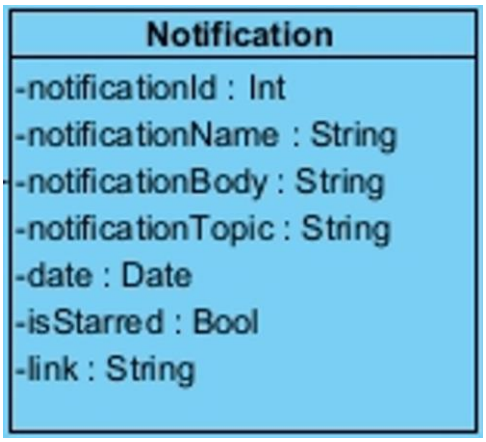
private String body: String including actual content of ToDo

private String link:

private bool isStarred: a boolean to store if the ToDo has star or not

private bool isDone: a boolean to store if the ToDo is done or not

private Date dueDate: Due date of ToDo, if it is user created ToDo then user can define a due



A class representing the notifications in the system

Attributes:

private int notificationId: Unique notification id stored in database

private String notificationName: Heading of the notification

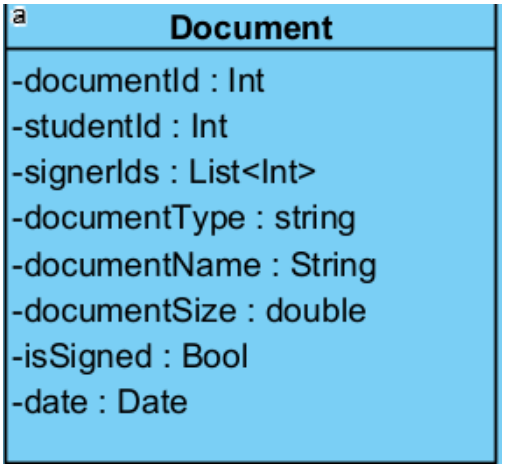
private String notificationBody: The actual content of notification

private String notificationTopic: The subject of the notification such as, messages, course approval

private Date date: The date of the notification when the user received it.

private bool isStarred: boolean that store if the notification has star or not

private String link: The link that includes corresponding interface user should reach when clicked on the notification.



A class that represents the documents in the system.

Attributes:

private int documentId: unique id of document to store in database

private int studentId: student id that exist in all documents

private List<int> signerIds: id integer list of the user that has signing authority

private String documentType: the type of document to store in database

private String documentName: document name that is defined by the user to store in database

private double documentSize: the size of the document

private bool isSigned: A boolean if the document is signed or not

private Date date: The date object which represents the date on which the file is uploaded

4. Glossary & References

[1] By: IBM Cloud Education, “What is three-tier architecture,” *IBM*. [Online]. Available: https://www.ibm.com/cloud/learn/three-tier-architecture#:~:text=Three%2Dtier%20architecture%20is%20a,associated%20with%20the%20application%20is. [Accessed: 29-Nov-2022].

[2] “Singleton,” *Refactoring.Guru*. [Online]. Available: https://refactoring.guru/design-patterns/singleton. [Accessed: 29-Nov-2022].

[3] “Template method design pattern,” *GeeksforGeeks*, 18-Oct-2021. [Online]. Available: https://www.geeksforgeeks.org/template-method-design-pattern/. [Accessed: 29-Nov-2022].

[4] “Web Browser System Requirements Overview,” *Ghacks*, 21-May-2012. [Online]. Available: https://www.ghacks.net/2012/05/21/web-browser-system-requirements-overview/. [Accessed: 28-Nov-2022].

[5] “What are the hardware and software requirements to run a website?,” *Quora*. [Online]. Available: https://www.quora.com/What-are-the-hardware-and-the-software-requirements-to-run-a-website/answer/Rohit-K-Choudhary?ch=10&oid=101070104&share=4cf084b1&srid=lilzw&target_type=answer. [Accessed: 28-Nov-2022].