Bilkent University 

**CS319 Object-Oriented Software Engineering**

**Project Design Report Iteration 2**

**Section: 2**

**Team: ABKUY**

**Group 2A:**

**-Ahmet Şahin (21902702)**

**-Barış Tan Ünal (22003617)**

**-Kaan Berk Kabadayı (22002580)**

**-Uğur Can Altun (22002701)**

**-Yusuf Şenyüz (21903105)**

**Instructor:** Eray Tüzün

**Teaching Assistants:** Emre Sülün, İdil Hanhan, Mert Kara,   
Muhammad Umair Ahmed, Metehan Saçakçı

**11.12.2022**

**CONTENTS**

[**1. Introduction**](#_x5kxu5fmuome) **4**

[1.1. Purpose of the System](#_9hs258hqvkjo) 4

[1.2. Design Goals](#_73v5ydgaqise) 4

[1.2.1. Top Design Goals](#_ibvlsnsp7sbe) 4

[1.2.2. Further Design Goals](#_g5dedllogmr6) 5

[**2. High-Level Software Architecture**](#_g19dfkbvlzg1) **6**

[2.1. Subsystem Decomposition](#_7elf6q14fo5x) 6

[2.2. Hardware/Software Mapping](#_s5n0v8kfhyn) 7

[2.3. Persistent Data Management](#_olziabk6oibu) 8

[2.4. Access Control and Security](#_ed4sw0ku1143) 9

[2.5. Boundary Conditions](#_1oqroz15720n) 11

[2.5.1. Initialization](#_m7mjywev87s1) 11

[2.5.2. Termination](#_8qvp74fnnetx) 12

[2.5.3. Failure](#_vtfopq5vlul) 12

[**3. Low-level design**](#_4v0nj59mi3dl) **13**

[3.1. Object design trade-offs](#_1un817520kkl) 13

[3.1.1. Usability vs Functionality](#_kj5lyqf97ke6) 13

[3.1.2. Usability vs Security](#_ofrtihu5ltt6) 13

[3.2. Final Object Design](#_vxm3kyyroijd) 14

[3.2.1. Abstract User](#_3x4n23fc5y8p) 17

[3.2.2. Student](#_z2gh6fwn68j) 17

[3.2.3. Department Erasmus Coordinator](#_7usj8yp9u02e) 18

[3.2.4. FacultyAdminCommittee](#_o10suifdszzo) 18

[3.2.5. CourseCoordinator](#_9odw6i8dy6cv) 19

[3.2.6. Dean](#_alnedwufmyy6) 19

[3.2.7. DepartmentChair:](#_exyvwi29l7l6) 19

[3.2.8. AdminErasmusCoordinator](#_93jllw6tg7k2) 19

[3.2.9. InternationalStudentsOffice](#_g60ecmsrhm1k) 20

[3.2.10. LoginManager](#_x2vnfp3d5iau) 20

[3.2.11. Authenticator](#_kt4kq4kux04b) 20

[3.2.12. ErasmusManager](#_k8zkv9sinoqj) 20

[3.2.13. NotificationManager](#_dso0fz6zu74w) 21

[3.2.14. DocumentManager](#_3vsr98d5vhjf) 21

[3.2.15. FileType](#_7jmx2el3fnue) 22

[3.2.16. Form](#_b6w75ydrkxk5) 22

[3.2.17. PreApproval](#_rr0k6n6q8pdm) 22

[3.2.18. CouseTransfer](#_r5vlce1k26i) 23

[3.2.19. ToDoList](#_z261l1m952bp) 23

[3.2.20. Task](#_ayrgsvhwq4rx) 23

[3.2.21. CourseWishList](#_7a4zggrgfjjp) 24

[3.2.22. Wish](#_mf25remxqyie) 24

[3.2.23. Standing](#_2tfhdrmdmjd2) 25

[3.2.24. Course](#_g9i57gz9rou7) 25

[3.2.25. BilkentCourse](#_xoestorhbz42) 25

[3.2.26. HostCourse](#_iyru19sh97ao) 26

[3.2.27. ApprovedCourses](#_jcxg0fhtc13g) 26

[3.2.28. PlacementTable](#_nk0qgoopxyr2) 26

[3.2.29. Placement](#_ggndfmmw9c6a) 26

[3.2.30. Semester](#_1dmxotvkh190) 27

[3.2.31. PlacementManager](#_f17w3chqnrbs) 27

[3.2.32. WaitingBin](#_wvqpmag7r0la) 27

[3.2.33. Document](#_z7vrketfatlj) 28

[3.2.34. Syllabus](#_y5jjbqwwjcdv) 28

[3.2.35. Transcript](#_utuutpfruu3) 28

[3.2.36. ScoreTable](#_3tl0v1ist2rq) 28

[3.2.37. ScoreTableRow](#_tdp96egy6zxv) 29

[3.3. Layers](#_rsch9f2ssa7) 29

[3.3.1. User Interface Management Layer](#_frgkap95q7hv) 29

[3.3.1.1. NavBar](#_hf6zesdgr39z) 29

[3.3.1.2. ApplicationPage](#_ekkgbdc0516j) 30

[3.3.1.3. ContactsPage](#_luijdkgp884o) 30

[3.3.1.4. SendEmailBox](#_kpa3et610qb3) 30

[3.3.1.5. SendMessageBox](#_py00tmbykdtz) 30

[3.3.1.6. AnnouncementsPage](#_c2d0d36szsd9) 30

[3.3.1.7. EvaluationPage](#_lvs0e45qpb28) 30

[3.3.1.8. EvaluateUniversityBox](#_pbt0navc67mo) 31

[3.3.1.9. MessagePage](#_sfm8x6rk20p5) 31

[3.3.1.10. PersonalInfoBox](#_i37lo7f4bzi) 31

[3.3.1.11. MyProfilePage](#_raalywxzwg72) 31

[3.3.1.12. NotificationsPage](#_3xpfcssdyevz) 31

[3.3.1.13. ForumPage](#_xgewjqz9kuxv) 32

[3.3.1.14. SideBar](#_wajbbeecot5e) 32

[3.3.1.15. PlacementsPage](#_onl4f1ite648) 32

[3.3.1.16. ScoreTablePage](#_6s085kfmk9u1) 32

[3.3.1.17. ToDoListPage](#_sn5zbj9xjcde) 33

[3.3.1.18. HelpPage](#_51fe0i46fmyn) 33

[3.3.1.19. LoginPage](#_2cxxlsx012ib) 33

[3.3.1.20. RegisterPage](#_weefle4pkr) 33

[3.3.1.21. StudentChecklistPage](#_d8gp4780lq8t) 33

[3.3.1.22. WishlistPage](#_dmjvgt6f4uc9) 33

[3.3.1.23. AddWishPage](#_za9v5fcwfncg) 34

[3.3.1.24. AddNewCoursePage](#_pho8os0a93n) 34

[3.3.1.25. PreapprovalPage](#_evp3thncj2o9) 34

[3.3.1.26. LearningAgreementPage](#_rrp2a9rwarqm) 35

[3.3.1.27. DeadlinesPage](#_o1grh62wlln4) 35

[3.3.1.28. ChooseDeadlinePage](#_2qre7to0mgh) 35

[3.3.1.29. CourseTransferPage](#_ou8iyb64xfbj) 35

[3.3.1.30. TransferrableCoursePage](#_tkbjctp7ilah) 36

[3.3.1.31. TranscriptsPage](#_h452xf7dhmqq) 36

[3.3.2. Web Server Management Layer](#_k9rs3dj208zx) 37

[3.3.3. Data Management Layer](#_jhj91ckd4dwl) 38

[**4. Improvement Summary**](#_r6vk4ict3i2m) **40**

[4.1. Formatting](#_h6zd41oj2u1h) 40

[4.2. Safety Criteria and Security Heading](#_iwo2gisvi38i) 40

[4.3. Subsystem Decomposition](#_ddefeqfb63qv) 40

[4.4. Persistent Data Management](#_316pzbcb7st4) 40

[4.5. Hardware/Software Mapping](#_izgs4bdulwk0) 40

[4.6. Web Server Layer](#_gl8vshraeleu) 40

[4.7. Database Subsystem Diagram](#_o1gfu2n5ry5z) 40

[**5. Glossary and References**](#_andokvqs1bv4) **41**

# 

# 1. Introduction

## **1.1. Purpose of the System**

In our project, we created an extension for Erasmus App which is called Erasmus Application Manager. Erasmus Application Manager is a complementary system of the current Erasmus App of Bilkent. The main aim of this website is to decrease usage of mail and paper during the erasmus process, and make the process smooth for all users. Our extension does not contain the application part of the Erasmus Program, instead, our extension uses the “Score Table” uploaded by the student office.

Our project aims to combine the different parts of the Erasmus procedure and ease the whole process for all users with a semi-automated system. Each student's process will be much smoother by making it easier to track and manage their Erasmus operations for the coordinators. Our project provides a system, which has an algorithm, to place students to host organizations, and a system for students to choose a wishlist and prepare Pre Approval Form, and a system for students to track the approval process of courses. Also, our project provides a system for coordinators to track and manually interact with the Erasmus process and approve students’ forms.

## **1.2. Design Goals**

Our focus in this report is the detailed and extended version of the non-functional requirements from the analysis report. Our system should be user-friendly and maintainable to facilitate the tasks of the users. It should be also safe because properties of entities in our project are private and they should be kept safely. The system should perform high and effective performance. In addition, our system should be functional.

### 1.2.1. Top Design Goals

**End-User Criteria:**

**Functionality:**

Functionality is a key element in our design because currently Erasmus App is present in the Bilkent University however, that system mostly depends on the paperwork and users of Erasmus App are not satisfied with the application. Both students and coordinators expect better functionality. Our system allows students to choose a wishlist and prepare a Pre Approval Form easily. In addition, one important functionality of our system is that each user can track their tasks by using a to-do list. Our system solves the placement problem as well. Placement is not made manually, instead an algorithm places students to universities.

**Dependability Criteria:**

**Safety:**

Safety, along with other criteria is a prime concern for our Erasmus Application Manager project. Similar to the already existing Bilkent Erasmus/Exchange App, Bilkent students and staff can login to the system using their STARS credentials without needing to register. Since the email addresses exist in STARS, when a user enters their email address, our authorization system checks for that email’s existence and if the password is true. If that email - password combination is true, the authorization system sends an email to ensure two factor authentication. For these operations, we need a synchronization between our app and the existing database for login authentication and authorization. While implementing the login feature, it must be made sure that no SQL injection or Cross Site Scripting attacks are possible to protect the confidentiality of our program. The registration and user roles will only be made and modified by the Admin user at the beginning of each semester.

### 1.2.2. Further Design Goals

**End-User Criteria:**

**Usability:**

Our purpose in terms of usability is to provide a simple and user-friendly interface to the users. We designed our project to look simple and consistent. Simplicity is achieved by using user-friendly icons and options. Another quality of our user-friendly interface is intuitiveness. Users can easily understand the role of each button and can use our system without facing any issues. Our programs provide help icons on all pages so users can understand what to do. Furthermore, there are various users that will use our system such as students, erasmus coordinators, department coordinators. Our design prepares different user-interfaces for different users because responsibilities and tasks of each user are different. Our system has sidebar and navbar that contain the required icons for corresponding users. Each user accesses and views only his/her related sections and tasks. Another property of our system’s user-interface is that users can see their current status and other information in their profile.

**Dependability Criteria:**

**Performance:**

Having high-quality performance is essential for our system because the system must work properly even in the worst performing computers. To achieve this, all complex computations will be done in a cloud server through HTTP requests and the users will only see the resulting HTML page and will not have any performance issues. Users must login to the application within at most 5 seconds. Each page must load within at most 2 seconds. The placement algorithm must finish within at most 1 hour so placement data is available 1 hour later the score Table is uploaded. The website should be compatible with Google Chrome, Safari, Microsoft Edge Opera and Mozilla Firefox.

**Maintainability:**

One of our biggest concerns when developing this project was whether it could be used in the real Erasmus application process and whether it could provide service for a long time. These concerns can be addressed with good maintainability through good design that comes with applying Object Oriented Programming techniques when developing the project. Because the Erasmus schools/courses are subject to change, the project should be open to add/remove features through good maintainability.

# 2. High-Level Software Architecture

## **2.1. Subsystem Decomposition**

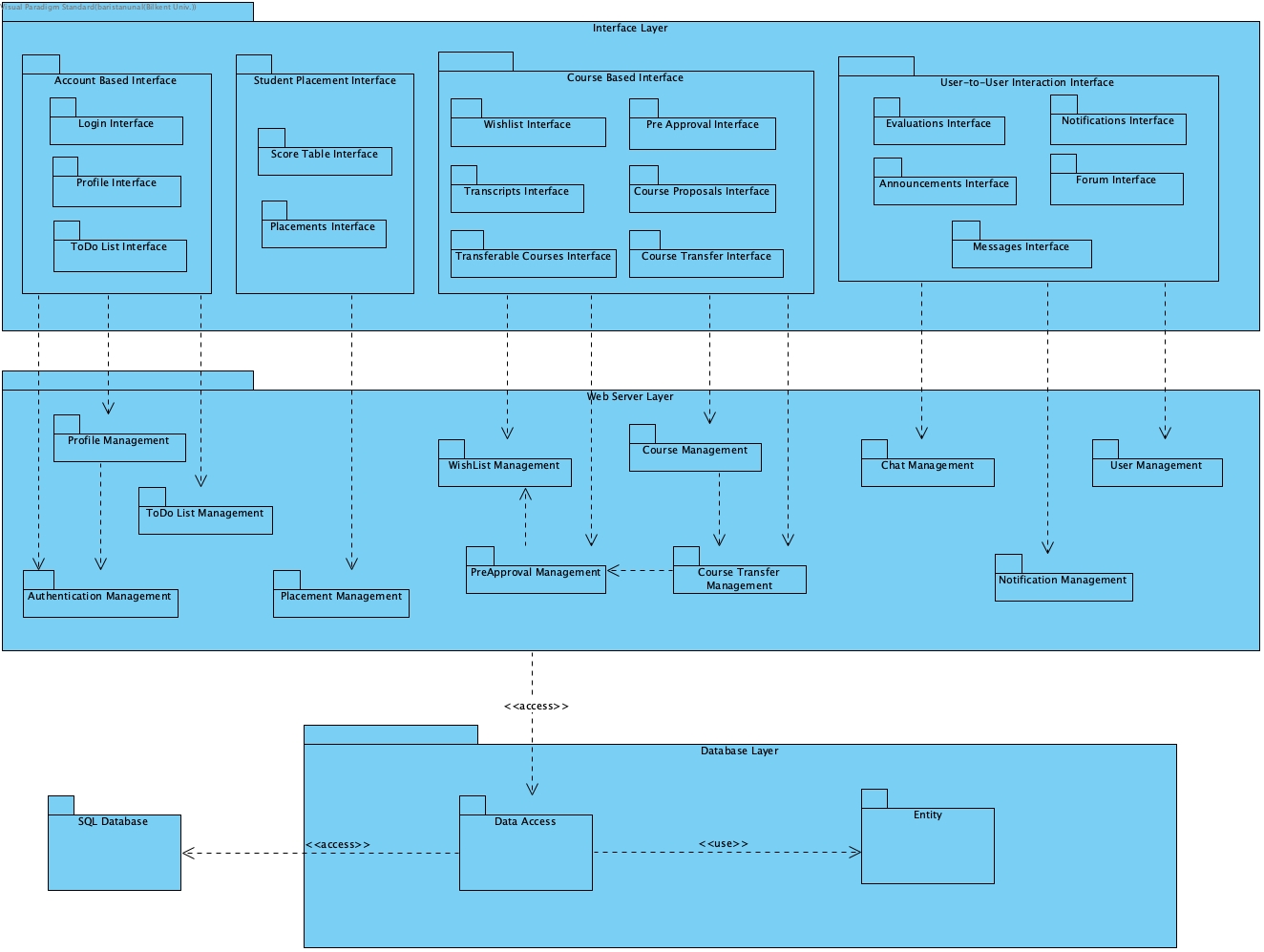


Fig. 1. Subsystem Decomposition Diagram

To achieve the maintainability level that is aimed for, a three-layer structure is used. In Figure 2.1, the overall software architecture can be seen. The Database Layer consists of Data Access, Entity and SQL Database subsystems. Data Access subsystems are created to provide an indirect access to the database so the implementation of the subsystems in the web service layer and entity subsystem can be independent from the type of database used. Data Access provides flexibility to other systems as the other systems can develop and adopt different technologies without worrying about the compatibility with the database.

The subsystems inside the Web Server Layer are involved in the operations done in the service, the back-end part of the program. Web Server Layer is partitioned into different subsystems according to the different features of the application to weaken the coupling in the software’s structure. Each subsystem is affiliated with the database and fetches the data that they do operations with from the database. Each subsystem represents a core functionality within the function. For example, Placement Management is involved with the student placement feature and PreApproval Management subsystem ensures the functionalities related with PreApproval process in the Erasmus program.

The User Interface Layer acts like a boundary object and contains subsystems related with the pages shown in the application. Each subsystem is affiliated with a page within the application and partitioned according to the pages inside the application. Every subsystem has its specific controller to ensure the user interaction between the system and the user. Users have many options such as clicking a button or writing an input to interact with the system. Each subsystem contains affiliated screens’ visual layout which are implemented to meet the project’s user friendly UI goal and their functionalities which interact with the controller component of the software architecture.

## **2.2. Hardware/Software Mapping**

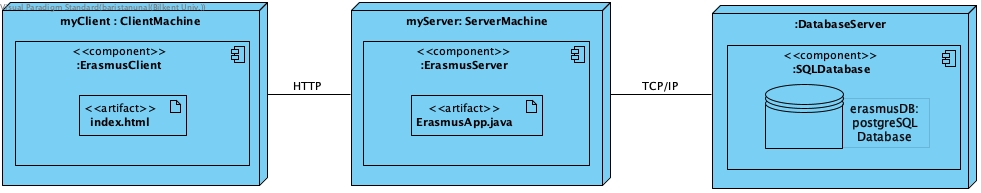


Fig. 2. Deployment Diagram

The Erasmus Application Manager does not require any specialized hardware components for successful execution as the application is a web extension to the existing Erasmus application website called “Erasmus+”. Therefore, web browsers are needed to run the application successfully and hardware systems should be sufficient enough to support web browsers. The Erasmus Application Manager can be accessed from personal computers and mobile phones as these gadgets can access web browsers, and web browsers are needed to access the Erasmus Application Manager. Keyboards, mouses and monitors can be used to support usage of the Erasmus Application manager in personal computers but are not necessary as most of the personal computers nowadays support monitor, keyboard and touchpad within its subsystems.

The Erasmus Application Manager uses HTML5, CSS3, and ECMAScript2020 (ES11) for its frontend. It uses React.js 18.2.0 as its framework and Node.js 18.12.1 for the JavaScript runtime environment. Furthermore, the web extension uses Spring Boot 2.7.5, Java JDK openjdk 17.0.5 2022-10-18, OpenJDK Runtime Environment Temurin-17.0.5+8 (build 17.0.5+8), OpenJDK 64-Bit Server VM Temurin-17.0.5+8 (build 17.0.5+8, mixed mode, sharing), and PostgreSQL 15.1 for the backend and database components of the application [1], [2].

The Erasmus Application Manager will be able to run on every browser that supports HTML5, which are the following:

* Google Chrome version 107
* Mozilla Firefox 106
* Safari version 16.1
* Microsoft Edge version 107
* Opera version 92

Furthermore, web browsers containing old versions like Internet Explorer 8 might be compatible due to APIs and libraries used in the application.

The deployment diagram is shown in figure 2. The deployment diagram contains 3 nodes. Nodes left-to-right contain the components: ErasmusClient, ErasmusServer and DatabaseServer, respectively. Frontend, backend and database of the application are deployed on different servers. ClientMachine and ServerMachine are connected by HTTP protocol and ServerMachine and DatabaseServer are connected by TCP/IP protocol.

For the ServerMachine and DatabaseServer, minimum hardware specifications are as follows:

* Processor: Intel Xeon 5600 Series or equal AMD CPU
* Memory: 16 GB RAM
* Operating System Disk: Dual HDD drives, 1 TB each
* Connection: Gigabit Ethernet
* Platform: All disk storage residing on single platform

## **2.3. Persistent Data Management**

The Erasmus Application Manager uses PostgreSQL for its database. PostgreSQL is chosen for the application’s database component because it is an object-relational database that helps integrate OOP principles into the software. Furthermore, PostgreSQL is a widely-used repository, which means that support revolving around PostgreSQL will be supplementary to the application’s development efforts. There is a need for communication between all users, so there will be a lot of data flow between different Users. Our application has to store necessary data for all types of users. Users' personal information, deadlines, application details of students, Bilkent courses, host university courses, previously approved courses, etc. are persistent data. This data should be able to be accessed by users in real-time. Hence we choose to store data in databases instead of using files. The necessary data of users will be stored in tables named User, Student, International Student Office, Course Coordinator, Task, etc. However, files with the extension ".pdf", ".XLSX", or ".DOCX" will be stored in folders according to our design choice. Their directory and name will be given by the program according to its type and who owns that file. For example, the Pre-approval form of a student whose id is 219027902 will be stored in “/PreApprovalForms/Erasmus” with the name “21902702\_PreApprovalForm”. Users will be able to alter the structure of the models inside the software, thus it is important to send HTTP requests such as GET, POST, DELETE, PATCH, and PUT, to the database dynamically [3]. We will use frameworks like Hibernate, and Java Persistence API (JPA). Spring Boot provides us with Object-Relational Mapping, so we are able to manage resources, use transaction strategies, and manage dependencies.

## **2.4. Access Control and Security**

The Erasmus Application Manager has a complex access control and security system to manage permissions of each stakeholder to ensure smooth functioning of application processes. There are many types of users with different tasks in the web extension, thus it is important to implement strict boundaries between the user's task and a detailed permission schema. The web extension can only be accessed via the credentials a user has on the Bilkent STARS system. When a user enters the web extension, the user’s credentials will be authenticated by the existing STARS database and the permission and UI layer of the matched user type will be provided to the user. Therefore, the UI layers, functions and permissions that users can reach are limited by its user type and access control is enforced. Furthermore, the Erasmus Application Manager fetches only the required data of the users to the frontend to enhance security. For example, sensitive data such as the password of the user is not fetched in unrelated pages and functions in the web extension.

|  | Student | Course Coordinator | Department Erasmus Coordinator | International Students Office | Administrative Erasmus Coordinator | Department Chair | Faculty Administration Committee | Dean |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Account Settings | X | X | X | X | X | X | X | X |
| View Notifications | X | X | X | X | X | X | X | X |
| Add Notification |  |  | X | X | X |  |  |  |
| Message Function | X | X | X | X | X | X | X | X |
| View/Use Forum | X | X | X | X | X | X | X | X |
| View Evaluations | X | X | X | X | X | X | X | X |
| Add Evaluation | X |  |  |  |  |  |  |  |
| View Announcements | X | X | X | X | X | X | X | X |
| Add Announcement |  |  | X | X | X |  |  |  |
| View /Use Contacts | X | X | X | X | X | X | X | X |
| View Application(s) | X |  |  |  |  |  |  |  |
| View/Use Coordinators Page |  |  | X |  |  |  |  |  |
| View/Use To Do List | X | X | X | X | X | X | X | X |
| View/Use Checklist | X |  |  |  |  |  |  |  |
| View/Use Wishlist | X |  | X |  |  |  |  |  |
| View /Use Pre Approval | X |  | X |  | X |  |  |  |
| View/Use Learning Agreement | X |  | X |  |  |  |  |  |
| View/Use Course Transfer | X |  | X |  |  | X | X | X |
| View/Use Course Proposals | X | X | X |  |  |  |  |  |
| View/Use Placements |  |  | X | X | X |  |  |  |
| View/Use Score Table |  |  |  | X |  |  |  |  |
| View/Use Transcripts |  |  | X | X |  |  | X |  |
| Add Deadlines |  |  |  |  | X |  |  |  |
| View Deadlines | X | X | X | X | X | X | X | X |
| View/Use Transferable Courses |  |  |  |  |  |  | X |  |

Fig. 3. User Permission Table

## **2.5. Boundary Conditions**

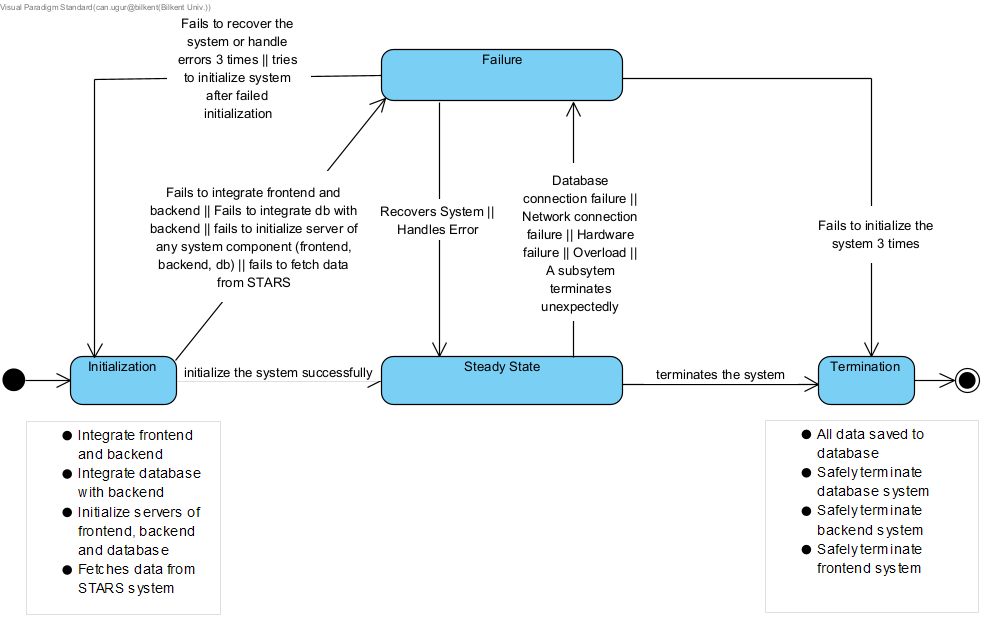


Fig. 4. Boundary Conditions State Diagram

### 2.5.1. Initialization

To initialize the application’s system; first the servers of frontend, backend and database components of the application are initialized. Then, the database is integrated with the backend and the backend is integrated with the frontend of the application. Finally, the required data to run the application successfully from the STARS system such as the roles of the users, credits and coordinators of the courses are fetched. The Erasmus Application Manager does not require any installation since the application is a web extension that runs on a web server. It runs 24/7 as long as the web server is stable and users can access it easily just with an internet connection and adequate authentication to enter to the platform. To access the web extension, users should have an account in Bilkent’s STARS system. If a user does not have an account inside Bilkent’s STARS system, it cannot access to the web extension in any form as the user needs to enter to the existing website, which requires an account in the STARS system, to access to the web extension. The logged user can access the web extension from different tabs and devices simultaneously. All data related with web extension are fetched from the database component of the application.

### 2.5.2. Termination

The Erasmus Application Manager works uniformly, the whole application terminates when a single subsystem of the system is terminated. Furthermore, the system terminates when the system fails to initialize again 3 times in case of failure. Admin can also terminate the system deliberately. All data created by the users of the program are saved to the database via the service layer to avoid data loss when an admin initiates termination of the system. Then, database, backend and frontend systems of the application are safely terminated to successfully terminate the application system safely.

### 2.5.3. Failure

The Erasmus Application Manager is currently hosted locally, failure of the hosted environment is notified to the developers. Failure of the system can happen due to database connection failure, network connection failure, hardware failure, overload or unexpected termination of a subsystem of the program. If the system is recovered or the generated errors are handled automatically, the system will turn to the stable state from the failure state. If the failure is not resolved by the recovery system or error handling system after 3 attempts, the system will be initialized again to restart the system. System can also enter the failure state if the system fails to enter the stable state in the initialization stage. In this case, the system will try to be initialized again. If the failure is not resolved by restarting after 3 attempts, the system will be terminated and developers will fix the problem manually. If the web extension is decided to be hosted in any cloud hosting service in the upcoming times, their procedure on failure will be followed. Furthermore, the service layer of the program has HTTP response codes for unsuccessful operations for the bugs created by the developers. Error logs explaining the problem will be stashed for debugging purposes.

# 3. Low-level design

## **3.1. Object design trade-offs**

### 3.1.1. Usability vs Functionality

This application is intended for usage of Students, Department Erasmus Coordinators, Administrative Erasmus Coordinators, Course Coordinators, Faculty Administration Committee, International Students Office, and BCC. Therefore, this application requires lots of different functions. This application requires higher functionality to provide users with accurate and efficient operation. However, the increase in the number of functions leads to decrease in usability for the users. There will be more choices for users to choose, more options, more buttons, more pages, this makes it difficult for users to take advantage of the functions.

### 3.1.2. Usability vs Security

This application is an extension to the current Erasmus Application, so accessing this application is as secure as the current Erasmus Application. However, in order to access our application users have to login to the Erasmus Application and click the button to access our extension. This decreases usability since it forces users to login to the current website. Without the secure login to our system, our system would be more usable. However, security is an essential non-functional property of our system because information of users is important and should be protected safely.

## **3.2. Final Object Design**

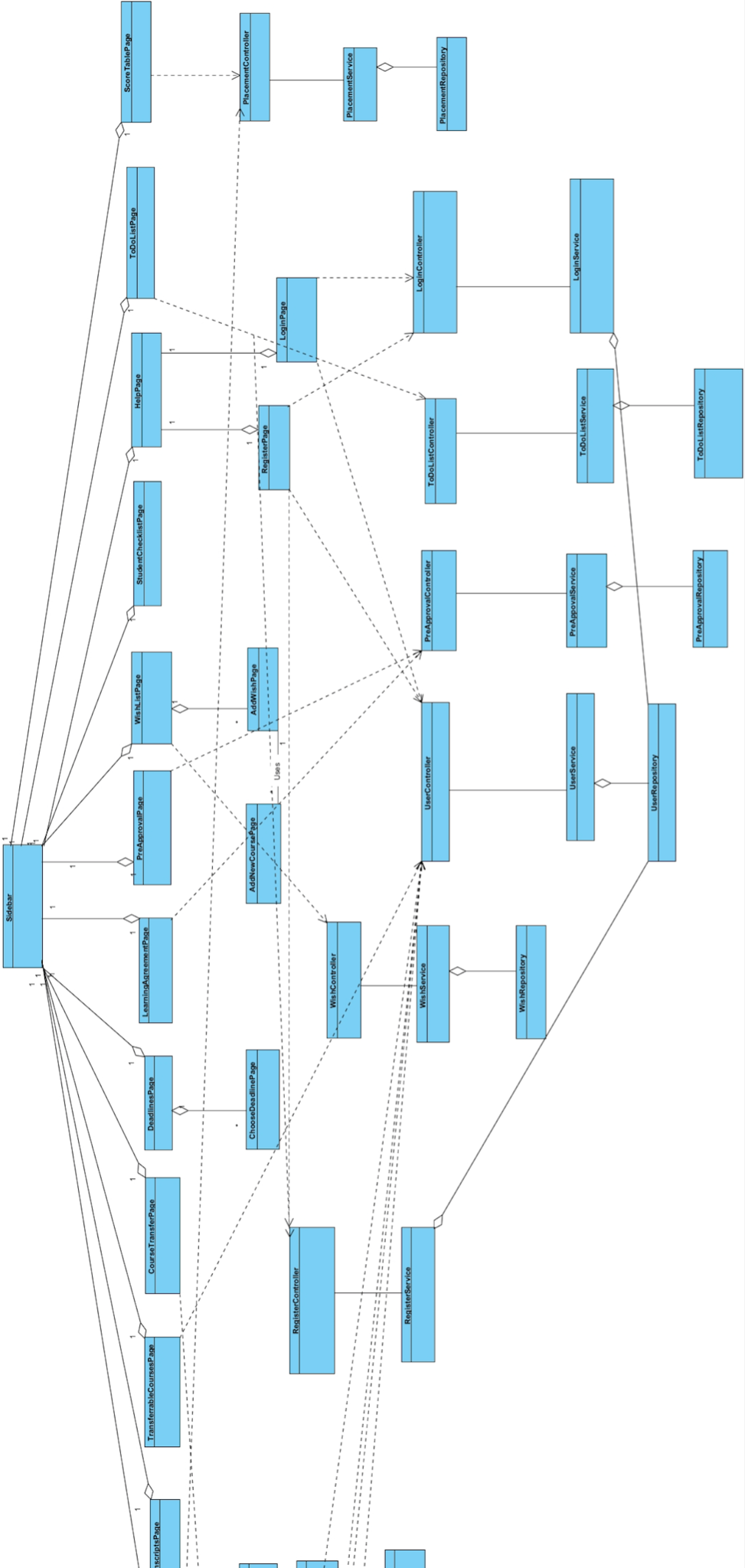


Fig. 5. Final Object Design Classes Right Side

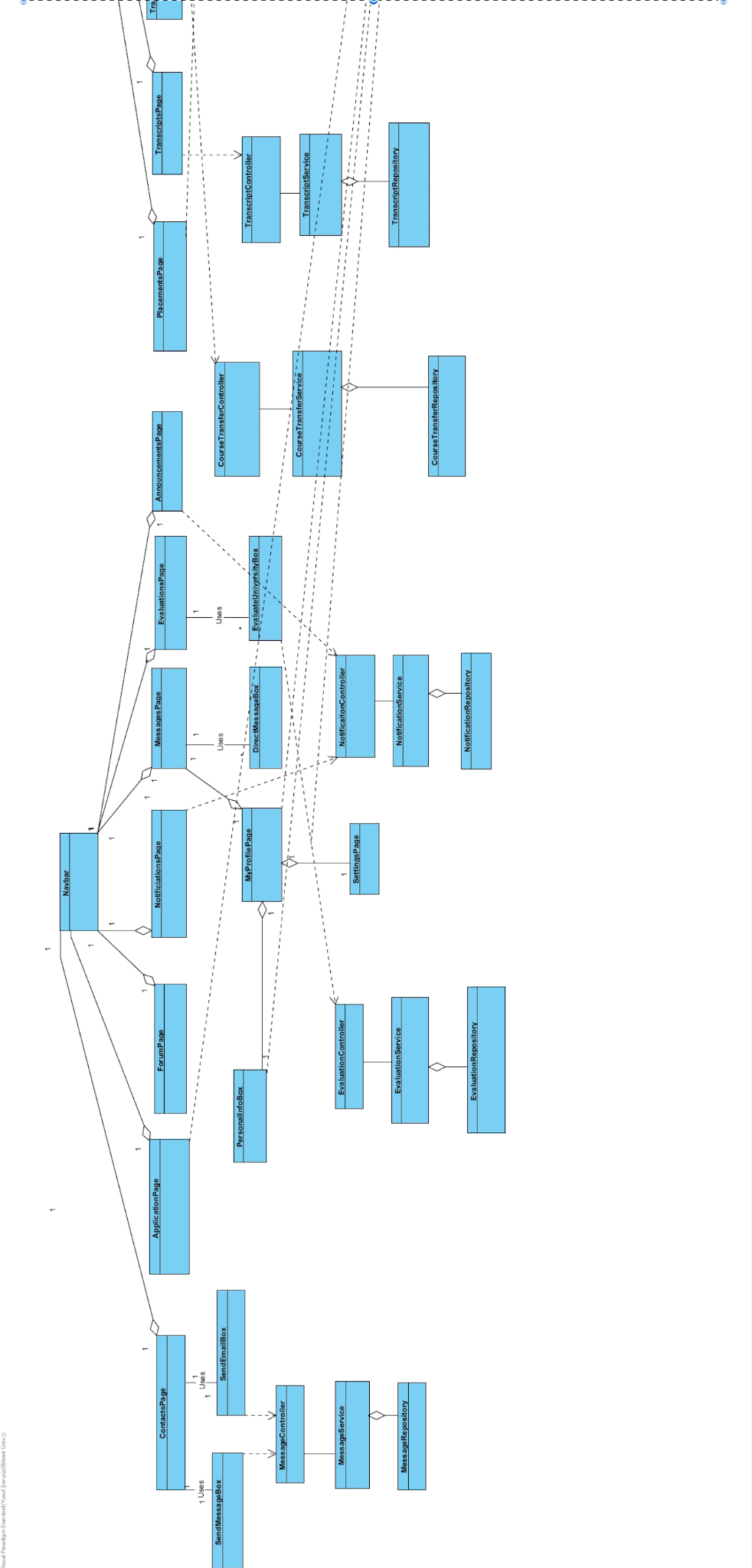


Fig. 6. Final Object Design Classes Left Side

### 3.2.1. Abstract User

**Explanation:**

User is an abstract class that captures the common behavior of the stakeholders of our software.

**Attributes:**

**private ToDoList toDoList :** Contains toDOList of the user. This attribute is used by users to manage their todo list according to their tasks.

**private String email:** Contains email of the user.

**private String firstName:** Contains first name of the user.

**private String lastName:** Contains last name of the user.

**private ErasmusManager erasmusManager:** Contains erasmusManager of the user. This attribute is used for required external functionalities such as notifying users and messaging between users.

### 3.2.2. Student

**Explanation:**

The student is a child class of the User class and is one of the major stakeholders in this program.

**Attributes:**

**private bool isPlaced:** It contains the result of whether the student has been placed at any host university.

**private String hostUniversity:** Contains name of host university if student is placed.

**private Transcript transcript:** Contains transcript of student. This attribute is used to retrieve all the courses the student has taken.

**private BilkentCourse[] passedCourses:** Contains passedCourses of students. This attribute is used to store all passed courses by students.

**private CourseWishList courseWishList:** Contains courseWishList of student. This attribute is used by the user to create and update his/her wishlist.

**private StudentInformation information:** Contains information of the user. This attribute is used to retrieve all necessary information such as ID and academicYear of the student.

**private Form[] forms:** Contains forms of user. This attribute is used for retrieving and examining course transfer form and Pre Approval Form.

**Methods:**

**public CourseTransfer createPreApp():** This method creates a Pre Approval form using the courseWishList attribute of the student if all courses are approved.

**public WishList createWishList():** This method creates a wishlist form and sends all unapproved courses to coordinators for approval.

**public void approvePlacement(Placement placement):** This method takes placement information as a parameter, and it is used for approving placement requests.

**public void uploadSyllabus(Syllabus syllabus):**This method takes syllabus information as a parameter, and it is used for uploading syllabus.

**public void informRegistrationFailure():** This method is used to inform coordinators in case a student fails to get all courses in his/her Pre Approval Form.

**public bool downloadLearningagreement():** This method is used to download Learning Agreement documents.

**public bool uploadLearningAgreement():** This method is used to upload a filled Learning Agreement document.

### 3.2.3. Department Erasmus Coordinator

**Explanation:**

DepartmentErasmusCoordinator is a child class of the User class and is one of the major stakeholders in the program.

**Attributes:**

**private PlacementManager placementManager:** Contains placementManager of Department Erasmus. This attribute is used to manage placement.

**Methods:**

**public PlacementTable placeStudents():** This method used to place students using placementManager.

**public bool matchPreAppTranscript(Transcript transcript, PreApp preApp):** This method is used for matching the Preapproval Form and Transcript uploaded by the International Students Office.

**public void finalizePreApp(PreApproval preApproval):** This method takes Pre Approval form as a parameter, and it is used to finalize Pre Approval of students whose Pre Approval form is passed as parameter.

**public void approveElectiveCourse(Wish wish):** This method takes wish, which includes counting courses in the host university to courses in Bilkent university , as a parameter. This method is used to approve whether a given host university course is countable as a Bilkent elective course.

**public void approveCourseTransfer(CourseTransfer courseTransfer):** This method is used to approve course transfer forms.

**public bool uploadLearningAgreement():** This method is used to upload a new Learning Agreement Form.

### 3.2.4. FacultyAdminCommittee

**Explanation:**

FacultyAdminCommittee is a child class of the User class and is one of the minor stakeholders.

**Methods:**

**public void approvePreApp(PreApproval preApproval):** This method takes the final version of the Pre Approval form of the student, and it is used to approve the final form of the Pre Approval form of students.

**public void approveCourseTransfer(CourseTransfer courseTransfer):** This method takes Course Transfer Exemption Form as a parameter, and it is used to approve course transfer form.

### 3.2.5. CourseCoordinator

**Explanation:**

The CourseCoordinator is one of the stakeholders in the program which is important for approving new must courses.

**Methods:**

**public void approveMustCourse(Wish wish):** This method takes wish, which includes counting courses in the host university to courses in Bilkent university, as a parameter. This method is used to approve whether a given host university course is countable as a Bilkent MUST course.

### 3.2.6. Dean

**Explanation:**

Dean is a child class of the User class and is one of the minor stakeholders in the program.

**Methods:**

**public void approveCourseTransfer(CourseTransfer courseTransfer):** This method takes Course Transfer Exemption Form as a parameter, and it is used to approve course transfer form after this form is approved by the Department Erasmus Coordinator.

### 3.2.7. DepartmentChair:

**Explanation:**

DepartmentChair is a child class of the User class and is one of the minor stakeholders in the program.

**Methods:**

**public void approveCourseTransfer(CourseTransfer courseTransfer):** This method takes Course Transfer Exemption Form as a parameter, and it is used to approve course transfer form after this form is approved by the Department Erasmus Coordinator.

### 3.2.8. AdminErasmusCoordinator

**Explanation:**

AdminErasmusCoordinator is a child class of the User class and is a stakeholder in the program.

**Methods:**

**public void viewPlacementTable(Placement Table):** This method takes a placement table as a parameter, and it is used to view the placement table.

**public void editNominationDeadline(int start, int end):** This method takes start and end date, and it is used to set/edit the deadline of the Nomination process according to given parameters.

### 3.2.9. InternationalStudentsOffice

**Explanation:**

InternationalStudentsOffice or the ISO is a child class of the User class and is one of the stakeholders in the program.

**Methods:**

**public void uploadScoreTable(DocumentManager docManager):** This method is used to upload the Score Table after the application period for Erasmus ends.

**public void uploadBoardingStudentTranscript(Transcript transcript, DocumentManager docManager):** This method is used to upload an official transcript of a student after the host university sends it to ISO.

### 3.2.10. LoginManager

**Explanation:**

LoginManager is a container class to make the authentication process abstract and usable by the User class without any problems.

**Attributes:**

**private Authenticator auth:** Contains auth of LoginManager. This attribute is used to authenticate users.

**Methods:**

**public void login():** This method is used to login to our extension application if the authentication process is successful.

### 3.2.11. Authenticator

**Explanation:**

The Authenticator class is used by the LoginManager class and is useful for handling the authentication process when the link to our program is clicked.

**Methods:**

**public void authenticate():** This method is used to authenticate the user.

### 3.2.12. ErasmusManager

**Explanation:**

This class is used by the User class and encapsulates the required external functionalities such as notifying users, messaging between users, downloading documents, and all other similar behaviors that should be abstracted out from the other classes.

**Attributes:**

**private NotificationManager notifier:** Contains notifier of ErasmusManager. This is used to notify other users.

**private DocumentManager docManager:** Contains docManager of ErasmusManager. This is used to manage documents.

**Methods:**

**public void proposePlacement(Placement placement):** This method takes placement as a parameter, and it is used to propose placement to students at the top of the waiting bin if there is vacancy.

**public void notifyStudentCourseApproval():** This method is used to notify students if a proposed host university course for counting for Bilkent University course is approved.

**public void notifyCourseCoordinator():** This method is used to notify Course Coordinator.

**public void notifyStudentPlacement():** This method is used to notify Student when Student is placed.

**public bool downloadDocument():** This method is used to download documents.

**public Transcript fetchTranscript():** This method is used to fetch Transcript.

**public void messageUser(User user, String message):** This method takes user and message context as a parameter, and it is used to send messages to other users.

**public SampleScoreTable createSampleScoreTable():** This method is used to create a Score Table.

### 3.2.13. NotificationManager

**Explanation:**

This class is used by the ErasmuManager class. This class provides some functionalities about the notification system such as notifying users and other similar behaviors that should be abstracted out from the other classes. There is one to many relationship between the ErasmusManager and NotificationManager so that a single ErasmusManager object can manipulate and use multiple NotificationManager objects.

**Methods:**

**public void notifyUser(User user, String notificationMessage):** This method takes 2 parameters: User that will have the notification and message that contains the notification prompt. Then, the corresponding user is notified.

### 

### 3.2.14. DocumentManager

**Explanation:**

This class is used by the ErasmuManager class. This class provides some functionalities x "about the document system such as downloading documents and other similar behaviors that should be abstracted out from the other classes. There is one to many relationship between the ErasmusManager and DocumentManager so that a single ErasmusManager object can manipulate and use multiple NotificationManager objects.

**Methods:**

**public bool uploadDocument(Document document):** This method takes a single parameter which is an Document that will be uploaded to the system. Return type of this method is bool. If the loading operation is successful this method returns **true**, otherwise; it returns **false**.

**public bool createDocument(Document document, FileType fileType):** This method takes 2 parameters which are a Document that will be created and type of that particular document.. Return type of this method is bool. If the creation operation is successful this method returns **true**, otherwise; it returns **false**.

**public bool downloadDocument(Document document, FileType fileType):** This method takes 2 parameters which are a Document that will be downloaded and type of that particular document. Return type of this method is bool. If the download operation is successful this method returns **true**, otherwise; it returns **false**

### 3.2.15. FileType

<<Enumeration>>

**Explanation:**

This class is an enumerable class and is useful for handling the problem of downloading documents in different formats

**Types:**

**PDF:** Whether the user wishes to download in PDF format

**DOCX:** Whether the user wishes to download in DOCX format

### 3.2.16. Form

**Explanation:**

Form class is an abstract class to capture the similar behavior of the Pre Approval Forms and the Course Transfer Exemption Forms.

**Attributes:**

**private bool isApproved:**Contains bool to check whether Form object is approved or not.

**private Wish[] wishes:** Contains array of Wishes. This is used to store all wishes in Form.

**private bool signedByDepCoordinator:** Contains bool to check whether Form object is signed by Department Coordinator.

### 3.2.17. PreApproval

**Explanation:**

PreApproval class is a child class of the Form class.

**Attributes:**

**private String preApprovalDeadline:** Contains deadline of Pre Approval process.

**Methods:**

**public bool createPreApprovalDoc():** This method is used to create PreApproval documents using wishes. Wishes is in wishArr, since Pre Approval is a child class of the Form class.

### 3.2.18. CouseTransfer

**Explanation:**

CourseTransfer class is a child class of the Form class.

**Attributes:**

**private bool signedByDean:**  Contains bool which states whether CourseTransfer form is signed by Dean.

**private bool signedByDepChair:** Contains bool which states whether CourseTransfer form is signed by Department Chair.

**private String[] passingGrades:** Contains array of passing grades which are corresponding to passing grades of wishes in wishArr. Wishes is in wishArr, since PreApproval is a child class of the Form class.

**Methods:**

**public bool createCourseTransfer():** This method is used to create Course Transfer Form.

### 3.2.19. ToDoList

**Explanation:**

ErasmusManager class contains this ToDoList class as its attribute. In other words, there is an aggregation relationship between the ToDoList class and ErasmusManager class. This class enables users to manage their tasks. Users can see their tasks and mark their tasks as done by using this class.

**Attributes:**

**private Task[] tasks:** Contains the array of tasks. This is used to store and manipulate all tasks.

**Methods:**

**private void addTask():** This method is used to add a particular task to the to do list of the particular user. There is no return type of this method. After using this class, an array of tasks changes. Array of tasks has an element.

**private void markTaskDone():** This method is used to mark a particular task as done in the to do list of the particular user. There is no return type of this method. After using this class, the array of tasks changes so that some tasks in the tasks[] become completed.

**private void updateTask():** This method does not add a new task to tasks[] instead this method changes the content of the specific task. There is no return type of this method. After using this class, the array of tasks are updated.

### 3.2.20. Task

**Explanation:**

ToDoList class contains the array of this Task class as its attribute. In other words, there is a composition relationship(strong type of aggregation) between the ToDoList class and Task class. This class is the building block of the ToDoList class.

**Attributes:**

**private String content:** This content attribute is the content of the particular task. Type of attribute is String.

**private String deadline:** This content attribute is the deadline of the particular task. Type of attribute is String. It specifies the due date of the corresponding task. Task can be marked as done in the specific deadline.

### 3.2.21. CourseWishList

**Explanation:**

CourseWishList aggregates the Wish class and contains a list of Wishes.

**Attributes:**

**private Wish[] wishes:** Contains array of wishes of CourseWishList.

**private bool isCompleted:** Contains bool to check whether all wishes in CourseWishList.

**private int totalCredit:** Cointans total credit of CourseWishList.

**private PreApprovalManager preappManager:** Contains preappManager of CourseWishList.

**Methods:**

**public bool checkCredit(int low, int high):** This method is used to check total credit of CourseWishList according to restrictions which are set by the Department Erasmus Coordinator.

**public bool checkPrerequisites():** This method is used to check prerequisites. // Bune beyler neyin neyini checkliyor, yani parameter falan niye yok

**public Wish sendApprovalRequestOfMustCourse(): Method to send approval request of MUST course.**

**public Wish sendApprovalRequestOfElectiveCourse(): Method to send approval request of elective course.**

### 3.2.22. Wish

**Explanation:**

Wish class and is contained by the class of CourseWishlist class. Wish is one of the 5-6 wishes a student must submit before forming a valid Pre Approval Form. A wish contains the class student wishes to take in the host university, the corresponding Bilkent course to transfer it, the intent of taking this course, and the syllabus of the course in the host university.

**Attributes:**

**private String intent:** This is the content of the intent. Student states that the course in the host university is similar to the course in Bilkent university and justifies its reasons.

**private Standing standing:** This attribute determines the status of

**private Syllabus syllabus:** Syllabus is the syllabus of the course in the host university. This attribute is necessary for the courses that were not previously accepted.

**private Map<BilkentCourse, HostCourse> courseMap:** This contains the pair of Bilkent University course and Host University course which student …

### 3.2.23. Standing

<<Enumeration>>

**Explanation:**

This class is an enumerable class and is useful for indicating the status of standing such as PENDING, SUCCESSFUL, UNSUCCESSFUL.

**Types:**

**PENDING:** This is used when the wish hasn't been approved or rejected yet. Confirmation is not finalized.

**SUCCESSFUL:** This is used when the wish is valid/acceptable and as a result of this, that corresponding wish is approved.

**UNSUCCESSFUL:** This is used when the wish is invalid/unacceptable and as a result of this, that corresponding wish is not approved.

### 3.2.24. Course

**Explanation:**

Course class is an abstract class useful for capturing the behavior of HostCourse and BilkentCourse. This class is contained by the Wish class.

**Attributes:**

**private int ECTSCredit:** Contains the ECTS credit of Course.

**private String name:** Contains name of Course.

**private String courseCode:**  Contains the course code.

**Methods:**

**public String getECTS():** This method is used to get ECTS of Course.

**public void setECTS(String ECTS):** This method takes ECTS as a parameter and sets the ECTS of Course.

### 3.2.25. BilkentCourse

**Explanation:**

This class is a child class of the Course class and contains the information about whether the course is to be taken as an elective course and the prerequisite BilkentCourses of this specific BilkentCourse.

**Attributes:**

**private BilkentCourse[] prerequisites:** Contains prerequisites of BilkentCourse.

**private bool isElective:**  Contains bool to check whether this BilkentCourse is elective course or MUST course.

### 3.2.26. HostCourse

**Explanation:**

This class is a child class of the Course class and contains the information about whether the course has been approved earlier, the name of the host university, and what are the BilkentCourses that a HostCourse can correspond to.

**Attributes:**

**private bool isApproved:** Contains bool to check whether HostCourse is approved before.

**private String hostName:** Contains name of host university.

**private BilkentCourse[] correspondingBilkentCourses:**  Contains Bilkent courses which are approved to count as HostCourse.

### 3.2.27. ApprovedCourses

**Explanation:**

This class acts as an interface between the Approved Courses Database and this program.

**Attributes:**

**private Map<HostCourse,BilkentCourse>:** Contains map of HostCourse and BilkentCourse, where HostCourse is countable as BilkentCourse.

**Methods:**

**public void addNewCourse():** Method is used to add a new course.

**public void removeCourse():**  Method is used to remove existing courses.

### 3.2.28. PlacementTable

**Explanation:**

This class contains the placements on the Erasmus/Exchange programme

**Attributes:**

**private List<Placement> placements:** List of Placement objects

**Methods:**

**public void addPlacement():** This method is used to add a new placement to the already existing placement.

**public void removePlacement():** This method is used to remove a placement from the already existing placement table.

### 3.2.29. Placement

**Explanation:**

This class is contained by the Placement class. There is a strong type of relationship between the Placement and Placement table which is called decomposition. A placement contains the relevant information of a student’s placement including the student’s name, id, name of the placed university as well as the duration preferred (fall/spring).

**Attributes:**

**private String studentName:** This attribute demonstrates the name of the student in the specific placement row.

**private String placedUniversity:** This attribute is the name of the university that the student is placed in.

**private int studentId:** This attribute is the ID of the student.

**private double totalPoints:** This attribute is the total points that a student has.

**private Semester durationPreferred:** This attribute is the semester that duration will occur such as fall/spring. This attribute is represented by the enumeration **Semester class**.

### 3.2.30. Semester

<<Enumeration>>

**Explanation:**

This class is an enumerable class to choose between mobility periods, the FALL and SPRING semesters.

**Types:**

**FALL:** Denotes the mobility semester of the student as the fall semester

**SPRING:** Denotes the mobility semester of the student as the spring semester

### 3.2.31. PlacementManager

**Explanation:**

This class contains the useful functions for implementing the placement algorithms and waiting bin algorithms.

**Attributes:**

**private PlacementTable placements:** PlacementTable object containing the placements made in the semester.

**private WaitingBin waitingBin:** WaitingBin object to fill after placements are done.

**Methods:**

**public void placeStudents():** Start the placement algorithms and fill the WaitingBin with remaining students.

### 3.2.32. WaitingBin

**Explanation:**

This class contains the placements on the Erasmus/Exchange programme.

**Attributes:**

**private List<Student> waitingStudents:** List of Student objects in the waiting bin.

**Methods:**

**public Student pickTopStudent(Student student):** This method is used to pick the top student in the waiting bin.

**public void removeStudent(Student student):** This method is used to remove a student from the waiting bin.

**public void addStudent():** This method is used to add a student to the waiting bin.

**public void changeRankOfStudent(Student student, int order):** This method is used to change the rank of a student in the waiting bin.

### 3.2.33. Document

**Explanation:**

This class is the parent class of the other specific document types such as **Syllabus class, Transcript class, ScoreTable class**.

**Attributes:**

**private String content:** This attribute shows the content that a document has. All document types have this attribute because it is a common attribute.

### 3.2.34. Syllabus

**Explanation:**

This class contains the relevant information contained in a Syllabus.

**Attributes:**

**privaString courseName:** Course name from the host university

**String courseCode:** Course code from the host university

### 3.2.35. Transcript

**Explanation:**

This class contains the relevant information contained in a Bilkent Student’s Transcript.

**Attributes:**

**private Map<BilkentCourse, bool> transcript:** Map of BilkentCourse to a boolean, whether the course has been taken or not by the student.

### 3.2.36. ScoreTable

**Explanation:**

This class represents the Score Table the International Students Office uploads to the system at the start of semester.

**Attributes:**

**private List<ScoreTableRow> scoreTableRows:** List of ScoreTableRow’s containing the relevant information of the score table.

### 3.2.37. ScoreTableRow

**Explanation:** This class represents each score table row and its relevant information such as student info, total point gathered and school preferences.

**Attributes:**

**private double totalPoints:** The total points gathered by the Bilkent student out of 100

**private String studentName:** Name of the Bilkent student

**private String studentLastName:** Last name of the Bilkent student

**private List<String> preference:** List of Host University preferences made by the student total of 5

## **3.3. Layers**

### 3.3.1. User Interface Management Layer

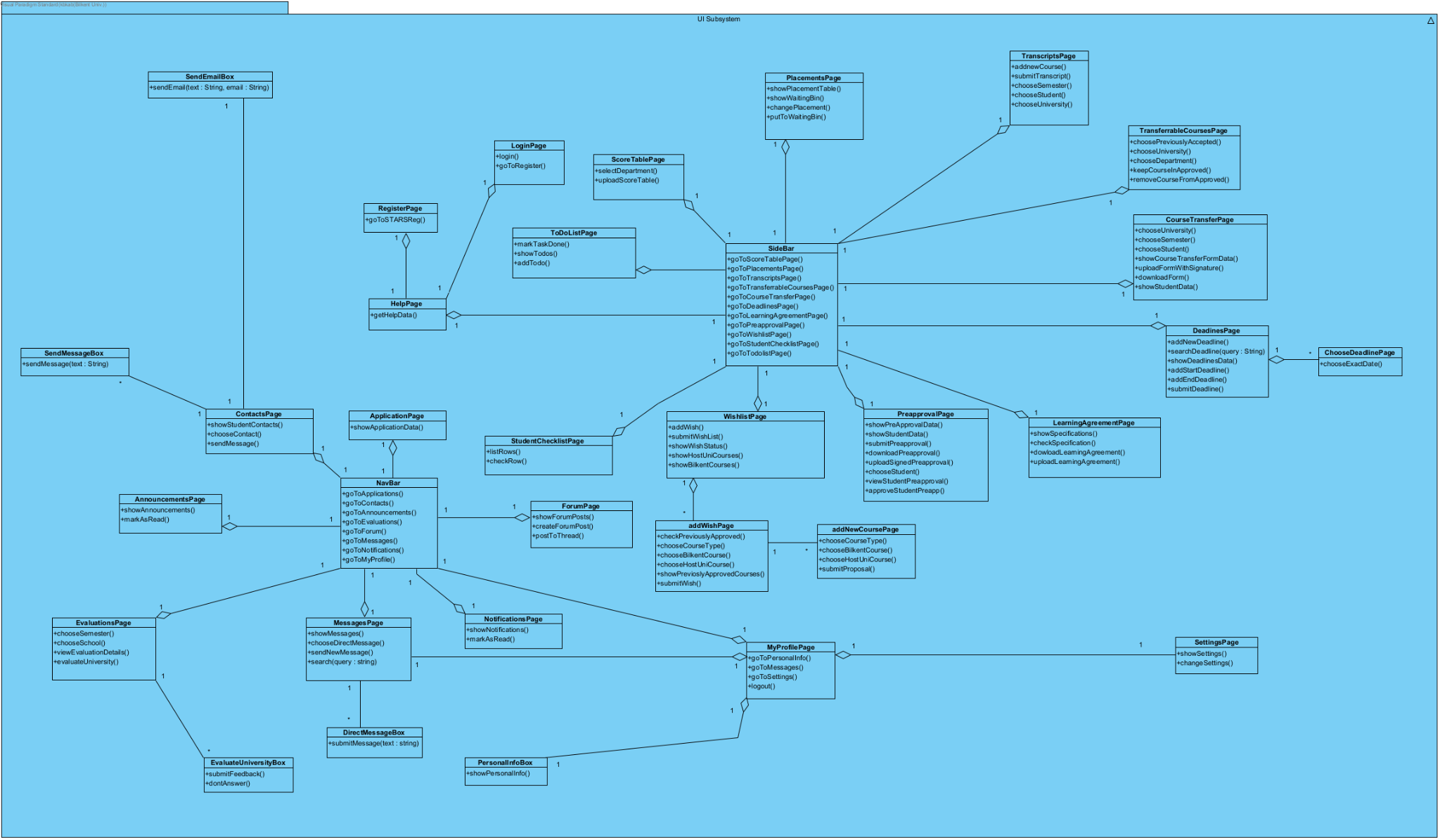


Fig. 7. User Interface Subsystem Diagram

This layer acts as the boundary object between the end-user and application layer. All of the classes in this layer are HTML files. Multiple classes communicate with the API service in the backend. The User Interface Management Layer contains dynamic components that change and work independently.

### 3.3.1.1. NavBar

**Explanation:** Navbar class is used to navigate between other pages.

**Methods:**

**public void goToApplications():** This method is used to go to the Application page.

**public void goToContacts():** This method is used to go to ContactsPage.

**public void goToAnnouncements():**This method is used to AnnouncementsPage.

**public void goToEvaluations():**This method is used on EvaluationsPage.

**public void goToForum():**This method is used to go to ForumPage.

**public void goToMessages():** This method is used to go to MessagesPage.

**public void goToNotifications():** This method is used to go to NotificationsPage.

**public void goToMyProfile():** This method is used to go to MyProfilePage.

### 3.3.1.2. ApplicationPage

**Explanation:** This class is used by students to see their applications.

**Methods:**

**public void showApplicationData():** This method is used to see applications.

### 3.3.1.3. ContactsPage

**Explanation:**This class is used to see contacts.

**Methods:**

**public void showStudentContacts():** This method is used to show student’s contacts.

**public void chooseContact():** This method is used to choose contact.

**public void sendMessage():** This method is used to send messages.

### 3.3.1.4. SendEmailBox

**Explanation:** This class is used to see and send email.

**Methods:**

**public void sendEmail(text:String, email:String):** This method is used to send email.

### 3.3.1.5. SendMessageBox

**Explanation:** This class is used to use and send messages.

**Methods:**

**public void sendMessage(text:String):** This method is used to send message

### 3.3.1.6. AnnouncementsPage

**Explanation:** This class is used to see announcements.

**Methods:**

**public void showAnnouncements():** This method is used to show announcements.

**public void markAsRead():** This method is used to mark announcements as read.

### 3.3.1.7. EvaluationPage

**Explanation:** This class is used to see evaluation forms and evaluate other schools.

**Methods:**

**public void chooseSemester():** This method is used to choose a semester to evaluate.

**public void chooseSchool():** This method is used to choose school to evaluate.

**public void viewEvaluationDetails():** This method is used to view Evaluation Details.

**public void evaluateUniversity():** This method is used to evaluate university.

### 3.3.1.8. EvaluateUniversityBox

**Explanation:** This class is used to evaluate selected schools.

**Methods:**

**public void submitFeedBack():** This method is used to submit feedback.

**public void dontAnswer():** This method is used if users do not want to give feedback.

### 3.3.1.9. MessagePage

**Explanation:** This class is used to handle messages.

**Methods:**

**public void showMessages():** This method is used to show messages.

**public void chooseDirectMessage():** This method is used to choose contact to send direct messages.

**public void sendNewMessage():** This method is used to send new messages.

**public void search(query: String):** This method is used to search old messages.

### 3.3.1.10. PersonalInfoBox

**Explanation:** This class is used to see personal information.

**Methods:**

**public void showPersonalInfo():** This method is used to show personal information.

### 3.3.1.11. MyProfilePage

**Explanation:** This method is used to see profile pages.

**Methods:**

**public void goToPersonalInfo():** This method is used to go to PersonalInfoBox.

**public void goToMessages():** This method is used to MessagesPage.

**public void goToSettings()** This method is used to go to SettingPage

**public void logout():**  This method is used to logout.

### 3.3.1.12. NotificationsPage

**Explanation:** This class is used to see notifications.

**Methods:**

**public void showNotifications():** This method is used to show notifications.

**public void markAsRead():** This method is used to mark notification as read.

### 3.3.1.13. ForumPage

**Explanation:** This class is used to see forum pages.

**Methods:**

**public void showForumPosts():** This method is used to show forum posts.

**public void createForumPost():** This method is used to create forum posts.

**public void postToThread():** This method is used to post to thread.

### 3.3.1.14. SideBar

**Explanation:** This method is used to see the sidebar.

**Methods:**

**public void goToScoreTablePage():** This method is used to go to scoreTablePage.

**public void goToPlacementsPage():** This method is used to go to placementsPage.

**public void goToTranscriptsPage():** This method is used to go to TranscriptsPage.

**public void goToTransferrableCoursePage():** This method is used to go to TransferrableCoursePage.

**public void goToCourseTransferPage():** This method is used to go to CourseTransferPage

**public void goToDeadlinesPage():** This method is used to go to DeadlinesPages.

**public void goToLearningAgreementPage():** This method is used to go to LearningAgreementPage.

**public void goToPreApprovalPage():** This method is used to go to PreApprovalPage.

**public void goToWishlistPage():** This method is used to go to WishListPage.

**public void goToStudentChecklistPage():** This method is used to go to StudentChecklistPage.

**public void GoToTodolistPage():** This method is used to go to TodolistPage.

### 3.3.1.15. PlacementsPage

**Explanation:** This class is used to see the placement page.

**Methods:**

**public void showPlacementTable():** This method is used to show placement tables.

**public void showWaitingBin():** This method is used to show the waiting bin.

**public void changePlacement():** This method is used to change placement.

**public void putToWaitingBin():** This method is used to put students to the waiting bin.

### 3.3.1.16. ScoreTablePage

**Explanation:** This class is used to see the score table page.

**Methods:**

**public void selectDepartment():** This method is used to select departments for the score table.

**public void uploadScoreTable():** This method is used to upload a score table.

### 3.3.1.17. ToDoListPage

**Explanation:** This class is used to see the todo list page.

**Methods:**

**public void markTaskDone():** This method is used to mark tasks as done.

**public void showTodos():**This method is used to show components in todo list.

**public void addTodo():**This method is used to add a new todo component.

### 3.3.1.18. HelpPage

**Explanation:** This class is used to see the help page.

**Methods:**

**public void getHelpData():** This method is used to get help data.

### 3.3.1.19. LoginPage

**Explanation:** This class is used to see the login page.

**Methods:**

**public void login():** This method is used to login.

**public void goToRegister():** This method is used to register.

### 3.3.1.20. RegisterPage

**Explanation:** This class is used to see the register page.

**Methods:**

**public void goToSTARSReg():** This method is used to go to the STARS registration page.

### 3.3.1.21. StudentChecklistPage

**Explanation:** This class is used to see checklist pages.

**Methods:**

**public void listRows()**: This method is used to list rows.

**public void checkRow():** This method is used to check row.

### 3.3.1.22. WishlistPage

**Explanation:** This class is used to see a wishlist page.

**Methods:**

**public void addWish():** This method is used to add a new wish.

**public void submitWishList():** This method is used to wishlist.

**public void showWishStatus():** This method is used to show wish status.

**public void showHostUniCourses():** This method is used to show host university courses.

**public void showBilkentCourses():** This method is used to show bilkent courses.

### 3.3.1.23. AddWishPage

**Explanation:** This class is used to see pages to add a new wish.

**Methods:**

**public void checkPreviouslyApproved():** This method is used to check whether an added wish is previously approved or not.

**public void chooseCourseType():** This method is used to choose course type.

**public void chooseBilkentCourse():** This method is used to choose a bilkent course.

**public void chooseHostUniCourse():** This method is used to choose a host university course.

**public void showPreviouslyApprovedCourses():** This method is used to show previously approved courses.

**public void submitWish():** This method is used to submit courses.

### 3.3.1.24. AddNewCoursePage

**Explanation:** This class is used to see a page to add a new course.

**Methods:**

**public void chooseCourseType():** This method is used to choose the type of new course.

**public void chooseBilkentCourse():** This method is used to choose a bilkent course.

**public void chooseHostUniCourse():** This method is used to choose a host university course.

**public void submitProposal():** This method is used to submit proposals.

### 3.3.1.25. PreapprovalPage

**Explanation:** This class is used to see the Pre Approval page.

**Methods:**

**public void showPreApproval():** This method is used to show Pre Approval form.

**public void showStudentData():** This method is used to show student data.

**public void submitPreapproval():** This method is used to submit Pre Approval form

**public void downloadPreappral():** This method is used to download Pre Approval.

**public void uploadSignedPreapproval():** This method is used to upload signed Pre Approval forms.

**public void chooseStudent():** This method is used to choose students.

**public void viewStudentPreapp():** This method is used to view student Pre Approval form.

**public void approveStudentPreapp():** This method is used to approve student’s Pre Approval.

### 3.3.1.26. LearningAgreementPage

**Explanation:** This class is used to see the learning agreement page.

**Methods:**

**public void showSpecifications():** This method is used to show specifications.

**public void checkSpecification():** This method is used to check specifications

**public void downloadLearningAgreement():** This method is used to download learning agreements.

**public void uploadLearningAgreement():** This method is used to upload learning agreements.

### 3.3.1.27. DeadlinesPage

**Explanation:** This class is used to see the deadline page.

**Method:**

**public void addNewDeadline():** This method is used to add new deadlines.

**public void searchDeadline(query: String):** This method is used to search deadlines using a given query.

**public void showDeadlinesData():** This method is used to show data of deadlines.

**public void addStartDeadline():** This method is used to add start time for deadline.

**public void addEndDeadline():** This method is used to add end time for deadline

**public void submitDeadline():** This method is used to submit deadlines.

### 3.3.1.28. ChooseDeadlinePage

**Explanation:** This class is used to see pages for choosing the deadline page.

**Method:**

**public void chooseExactDate():** This method is used to choose exact dates.

### 3.3.1.29. CourseTransferPage

**Explanation:** This class is used to see the course transfer page.

**Method:**

**public void chooseUniversity():** This method is used to choose university.

**public void chooseSemester():** This method is used to choose the semester.

**public void chooseStudent():** This method is used to choose the student.

**public void showCourseTransferFormData():** This method is used to show course transfer form’s data.

**public void uploadFormWithSignature():** This method is used to upload a form with signature.

**public void downloadForm():** This method is used to download a form.

**public void showStudentData():** This method is used to show student’s data.

### 3.3.1.30. TransferrableCoursePage

**Explanation:** This class is used to see the transferable course page.

**Method:**

**public void choosePreviouslyAccepted():** This method is used to choose previously accepted courses.

**public void chooseUniversity():** This method is used to choose university.

**public void chooseDepartment():** This method is used to choose a department.

**public void keepCourseInApproved():** This method is used to keep course in approved ones.

**public void removeCourseFromApproved():** This method is used to remove a course from approved ones.

### 3.3.1.31. TranscriptsPage

**Explanation:** This class is used to see the page of transcripts.

**Method:**

**public void addnewCourse():** This method is used to add a new course.

**public void submitTranscript():** This method is used to submit a transcript.

**public void chooseSemester():** This method is used to choose a semester.

**public void chooseStudent():** This method is used to choose a student.

**public void chooseUniversity():** This method is used to choose university.

### 3.3.2. Web Server Management Layer

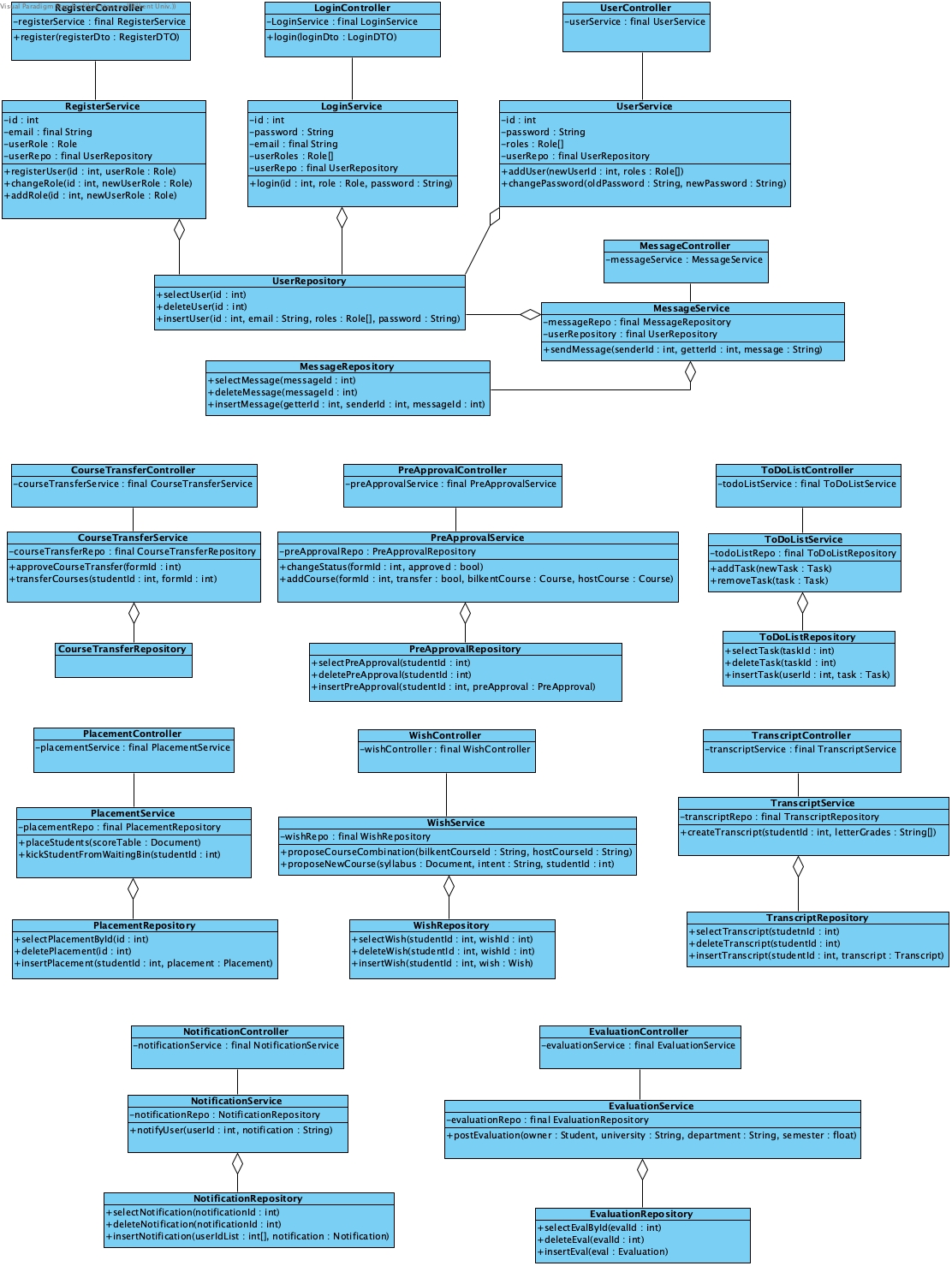


Fig. 8. Web Design Layer (Controller and Service)   
and its connections with Repository classes

### 3.3.3. Data Management Layer

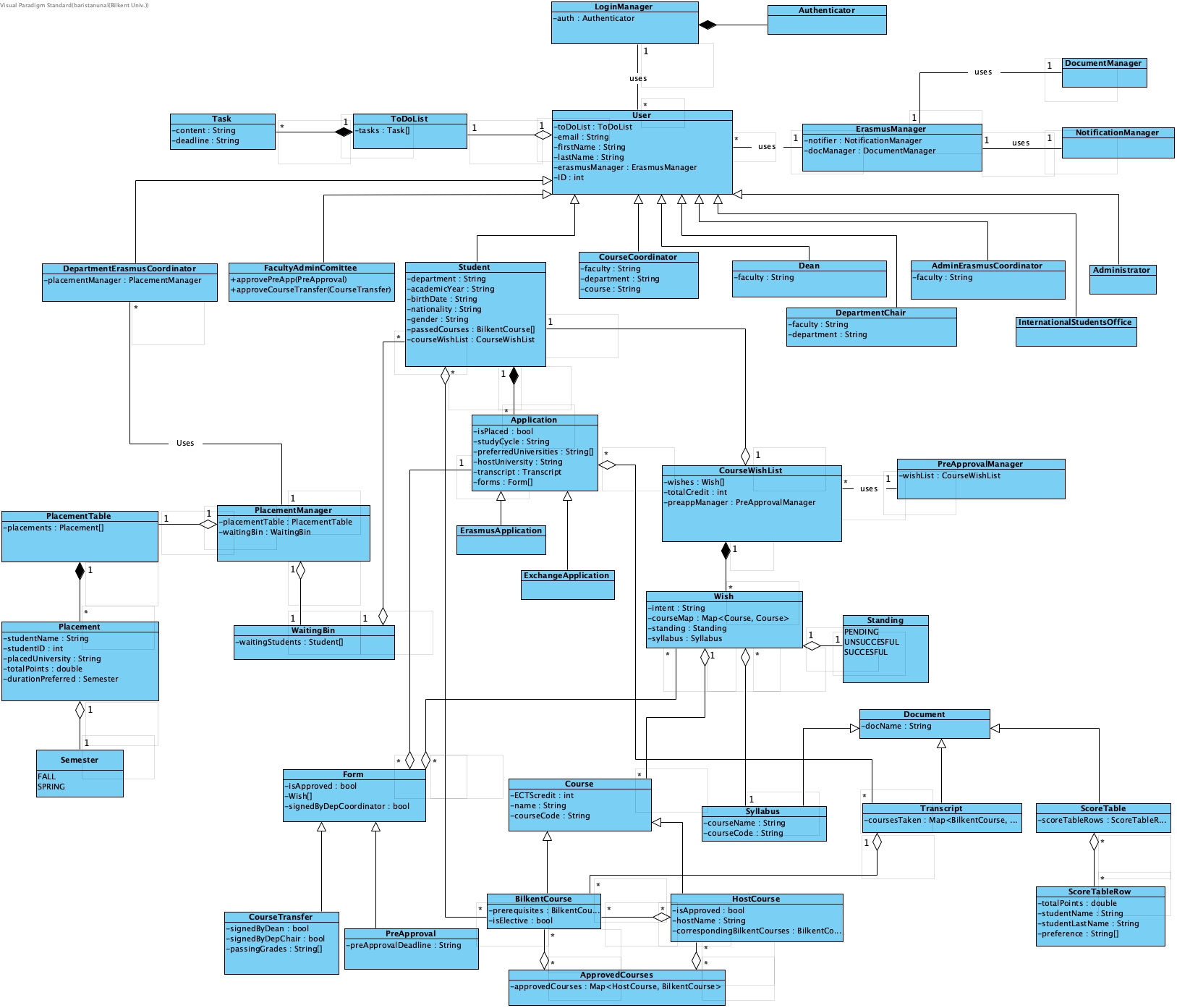


Fig. 9. Entity Class Diagram

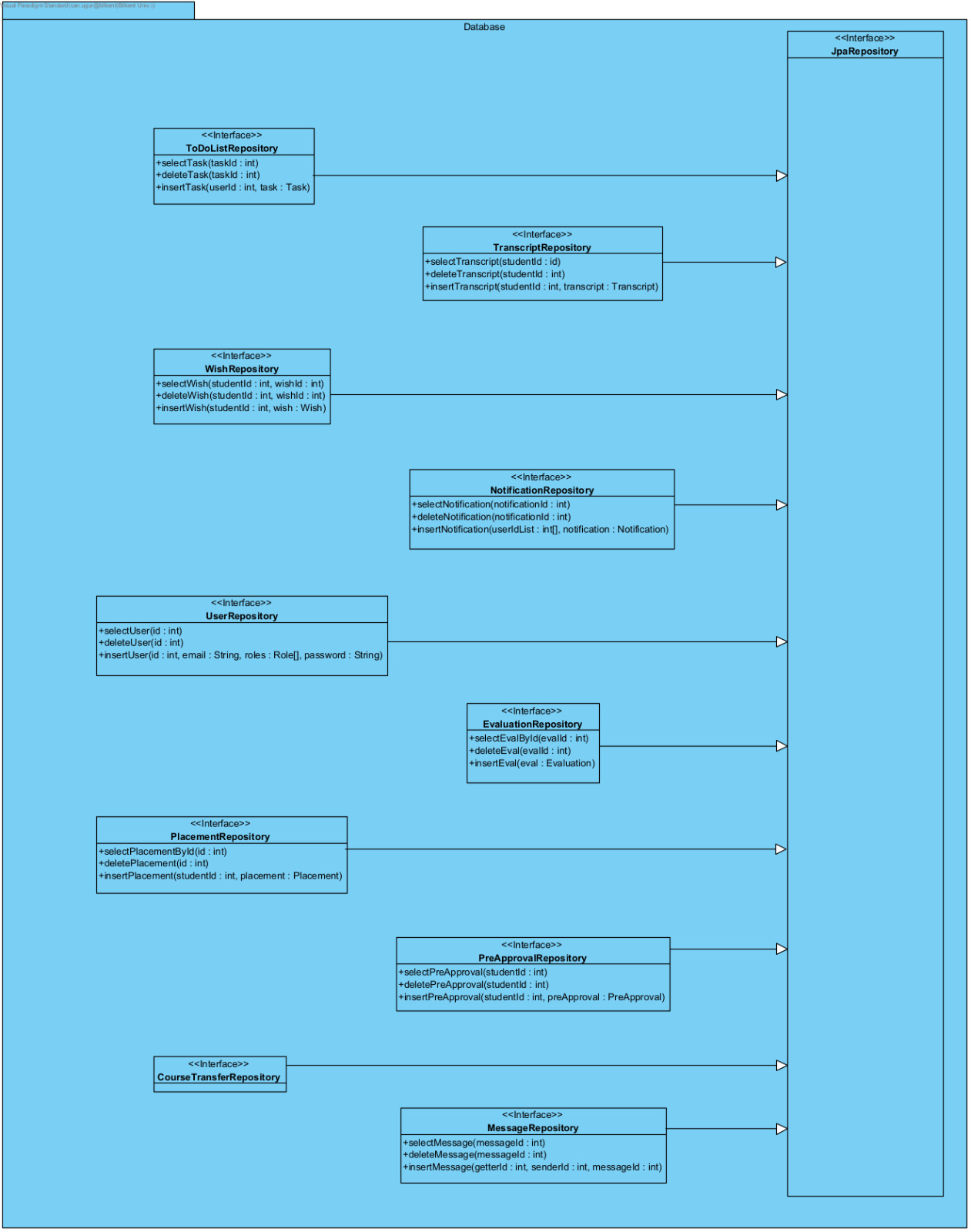


Fig. 10. Database Subsystem Diagram

# 4. Improvement Summary

## 4.1. Formatting

* Added page numbering,

## 4.2. Safety Criteria and Security Heading

* Fixed wording and explanation of how a session is authenticated and authorized by the system

## **4.**3. **Subsystem Decomposition**

* Added “Account Based Interface”, “Student Placement Interface”, “Course Based Interface” and “User-to-User Interaction Interface” packages for grouping purposes.
* Updated the connections according to the new package structure.

## 4.4. Persistent Data Management

* Added information about what data should be persistent.
* Explained where to store these persistent data. Some of them such as personal information in the database some of them such as PDFfiles in the local folders.

## **4.**5. **Hardware/Software Mapping**

* Added hardware specifications for the ErasmusServer and DatabaseServer

## 4.6. Web Server Layer

* Made the types of all ID’s int instead of float.

## 4.7. Database Subsystem Diagram

* Added delete and insert operations to repository classes.

# 

# 5. Glossary and References

[1] “Latest releases,” *Adoptium*. [Online]. Available: <https://adoptium.net/temurin/releases/>. [Accessed: Nov, 27 2022].

[2] “E.1 release 15.1,” PostgreSQL Documentation, Nov, 10 2022. [Online]. Available: <https://www.postgresql.org/docs/15/release-15-1.html>. [Accessed: Nov, 27 2022].

[3] “HTTP request methods - http: MDN,” *HTTP | MDN*. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>. [Accessed: Nov 27, 2022].