Bilkent University

CS-319

Object-Oriented Software Engineering

Fall 2022/2023

Detailed Design Report

Group “erasmus muarrem”

Instructor: Eray Tüzün

TAs: İdil Hanhan, Muhammad Umair Ahmed,

Mert Kara, Emre Sülün

14/11/2022

Member Name	Student ID	Section
Korhan Kemal Kaya	21903357	02
Halil Alperen Gözeten	21902464	02
Hazal Buluş	21903435	02
Kürşad Güzelkaya	21902103	02
Süleyman Gökhan Tekin	21902512	02

1. Introduction

The software that will be developed by us will be provided for the use of Bilkent students, course coordinators, academic staff, department coordinators, and also incoming Erasmus/Exchange students to manage the Erasmus/Exchange Application process and streamline the needed communication. By using our software, the Bilkent students applying to the Erasmus/Exchange programs can track their applications, respond to offers, create pre-approval forms and course requests by uploading necessary documents, and submit/receive documents for the post-acceptance processes. In order to provide these functionalities, the department coordinators will be able to maintain the application process and manage the situations when students request to cancel/change their applications. They will also be able to approve course (elective) requests, pre-approval forms, and language competencies of the outgoing students while also guiding applicants through announcements. Course coordinators, as another actor ensuring the progress of the system, will be able to respond to course requests of the students and can request the final reports for the course (for some of them) taken in the mobility period. As another essential piece of the process, administrative staff will be able to manage the details related to the contracted partner universities and respond to document-related needs of accepted students and to course proposals by incoming students. Lastly, the incoming students, as another user type in our software, can propose a list of courses they wish to take and follow its status.

1.1 Purpose of the System

The main purpose of the system is to create a web-based application to digitize the different paperwork needed (course requests, pre-approval forms, acceptance letters, etc.) for students, educators, department managers, and academic staff during the Erasmus application process and manage the placement process in a more systematic manner. Through this, it's aimed to minimize the communication overhead and the time lost in this application process and make it more favorable compared to the existing system. Another goal of the system is to diminish the mailing and phone calls between students and academic personnel.

1.2 Design Goals

The web application that we are developing mainly focuses on three design goals: Usability, Maintainability, and Security. These design goals are chosen for the application to be more advantageous in terms of time efficiency and easiness and also more long-lasting than the current system. We believe these design goals are substantial not only for the users we

mentioned in the introduction but also for the Bilkent Computer Center (BCC) staff, who will (possibly) manage the application.

1.2.1 Usability

The purpose of implementing an Erasmus Application Management System is to facilitate the process for both students and academics involved in the process, such as department coordinators. Users of the system should benefit more from the ongoing system. Facilitation, as one of the main goals of the application, requires a usable and practical system for all its users. The system will include simple workflows and UI elements to improve the user experience. Since the purpose of the system is to minimize the use of paper transfer and online mailing, it will be designed so that all users can upload certain required documents. Regardless of the screen size, the font size and the size of the buttons will be easily visible on each screen. The font on each screen will be 14 pt, the dimensions of the buttons will never be less than 7% of the screen, and information tables will not be more than 5 columns so that the texts and any other properties will be easily readable by the user. Additionally, the user will always be steered toward the appropriate input type whenever input is necessary; for example, if a number type is required, the user will not be allowed to write a text. Also, the system will be as simple and straightforward as possible by implementing the UI according to the traditional methods common to many applications and familiar to many users. To find a student profile, users will be able to search for a specific student (such as in the placement list that consists of too many students), which will enable them to find the specific student they are looking for. This will allow users to access the information they want easier and faster, which will increase usability in general.

1.2.2 Maintainability

Considering the fact that this software may be deployed in real life, a maintainable software design and implementation is a must. Taking this situation into place, modern software engineering practices and maintainable software development methods are used while developing the application. As a result of the interviews we had with various academicians who took part in the application process and our student friends who had gone to Erasmus/Exchange programs before, we obtained a valid problem & application domain. Starting the implementation by drawing UML diagrams, such as Use-Case, sequence, and state, in line with the results of our requirement analysis is important for the maintainability of the project. These

diagrams not only show us the direction we will take during the implementation process but also strengthen and determine the structure of the project. In this way, the main thing that anyone who aims to maintain the project should do is to act in accordance with the diagrams, which increases the maintainability of the system. If the subject is considered more technically, an active and process GitHub repository, a well-commented code, and the well-documented technologies of React.js and Spring Boot, which are the frameworks to be used in the coding process, are among the factors that increase the maintainability of the system. Also, the 3-Layered-Architecture Style and patterns of OOP will ensure that the system is modifiable and thus maintainable.

1.2.3 Reliability

This application will be used for Erasmus/Exchange programs which have significant effects on students' academic lives. Also, this application will be used by academic personnel of a university. Hence, users' data should be kept safely, and unauthorized access should be prevented to make the application reliable. Information and documents of the users will be stored privately. The passwords of the users will be stored as encrypted. Users will use the activation code that is sent to their email by the application to change their passwords in case they forget. To protect the application from unexpected crashes, possible errors and exceptions will be handled. In addition to these security measures, several methods will be used to strengthen error handling. To eliminate errors, different testing practices, such as reliability testing, will be implemented while the system is being implemented and after the implementation is complete. It is aimed at creating a bug-free system by eliminating the detected errors.

2. High-level Software Architecture

2.1 Subsystem Decomposition

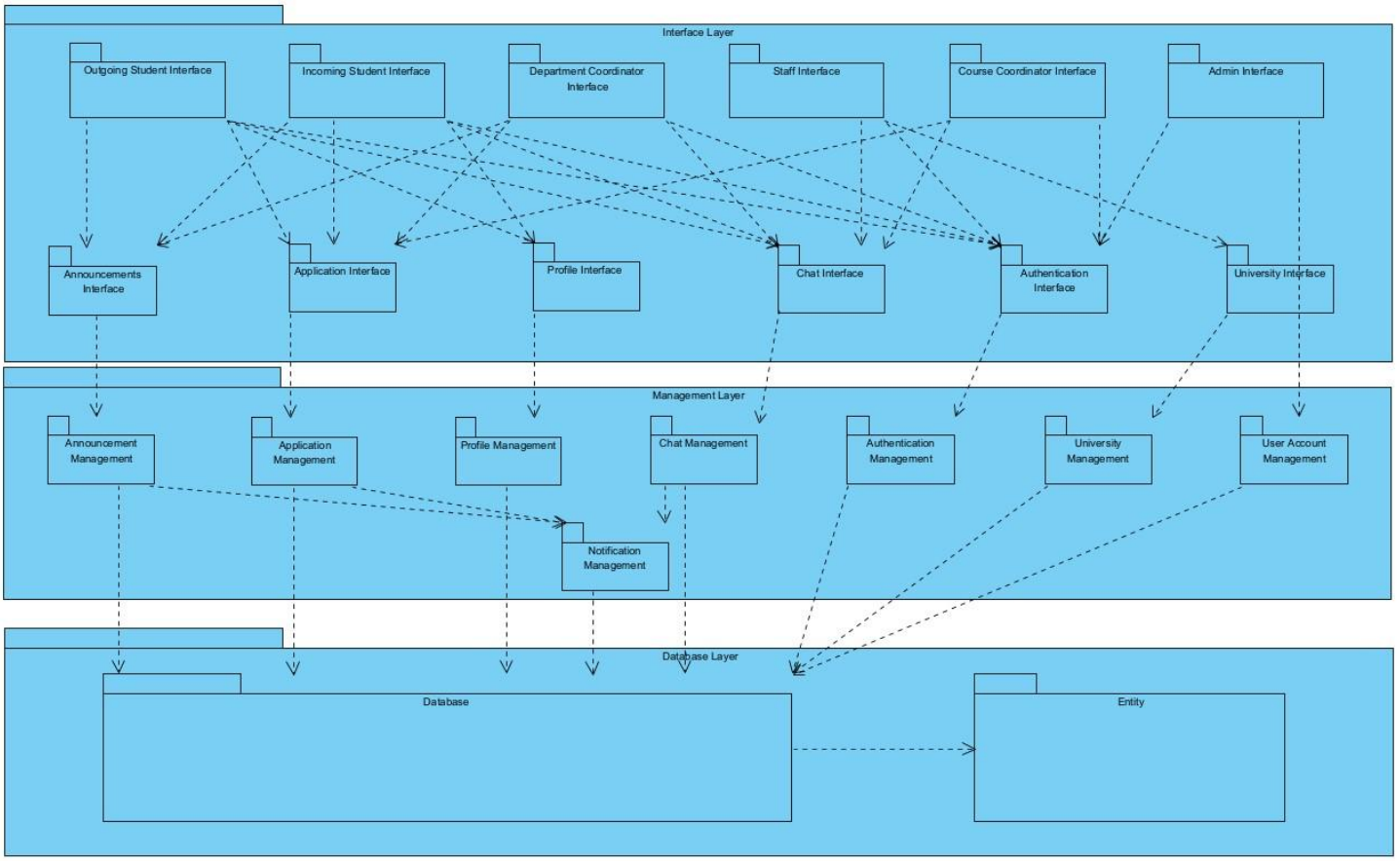

Figure 1: *Subsystem Decomposition of the System (click here for larger size)*

Because of the maintainability goal in our application, a 3-Layer Architecture style is chosen for our project. This architecture style, which separates applications into three logical and physical computing tires, is a widely used architecture choice. Due to the hierarchy, this architectural style creates, developers can make developments while the other parts remain the same.

The first layer, which is the Interface Layer, is like the boundary object, and it contains UI elements in our project. Since there are six different user types in the application, we divided this layer accordingly. These different user types in the application will see different pages according to their roles. Every page has its own controller elements, so users can benefit from the functionalities of the application by using the specific UI elements on pages. Also, these pages are designed considering User Experience and simplicity.

Explanation of the interfaces of the Interface Layer:

- Outgoing Student Interface: Necessary interfaces and items that are going to be shown to Outgoing Students will be displayed in this interface.
- Incoming Student Interface: Necessary interfaces and items that are going to be shown to Incoming Students will be displayed in this interface.
- Department Coordinator Interface: Necessary interfaces and items that are going to be shown to the Department Coordinator will be displayed in this interface.
- Administrative Staff Interface: Necessary interfaces and items that are going to be shown to Administrative Staff will be displayed in this interface.
- Course Coordinator Interface: Necessary interfaces and items that are going to be shown to the Course Coordinator will be displayed in this interface.
- Admin Interface: Necessary interfaces and items that are going to be shown to Admins will be displayed in this interface.
- Announcements Interface: Necessary interfaces and items for displaying the announcements are in this interface.
- Request Interface: Necessary interfaces and items for making and displaying requests are in this interface.
- Profile Interface: Necessary interfaces and items for displaying and editing profiles are in this interface
- Chat Interface: Necessary interfaces and items for displaying messages and contacting other users are in this interface.
- Authentication Interface: Necessary interfaces and items for login and other authentication operations are in this interface
- Host University Interface: Necessary interfaces and items for displaying the information about the Host University are in this interface.
- Contracted University Interface: Necessary interfaces and items for displaying the information and making operations about the Contacted University are in this interface.
- Application Control Interface: Necessary interfaces and items for displaying the information, such as all users or adding new users, are in this interface.

The second layer, which is the Management Layer, is responsible for making the connection between the front end (User Interface) and the back end (database) of the application. With this layer, data in the database will be changed while users are using the application. Each one of the subsystems in this layer depends on the database and is responsible for managing functionality in the application.

2.2 Hardware/Software Mapping

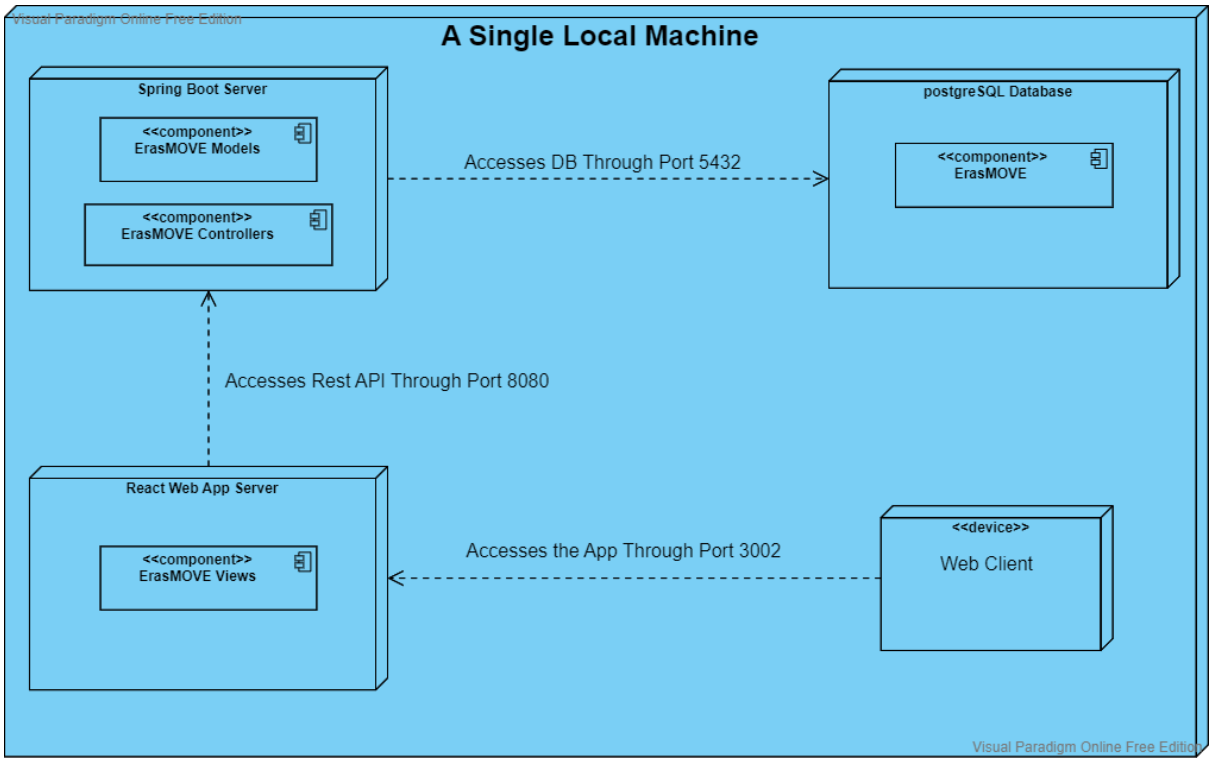

Figure 2: *Deployment Diagram*

The system that we are going to design will be a lightweight website with low consumption of the system resources and will not have a bare minimum as the hardware specification but will depend on the requirements of the browsers used. Computers, smartphones, and tablets with browsers will be capable of accessing and interacting with our website. For a computer that will access our system, the most basic requirements will be a monitor, keyboard, and mouse, together with a pre-installed proper-working browser. Suggested and commonly used browsers for macOS, Linux, Windows, and Android operating systems are Google Chrome (latest stable version: 107.0.5304.121), Mozilla Firefox (latest stable version: 107.0), and Microsoft Edge (latest stable version: 107.0.1418.56). To give a more concrete example of the hardware specification, the Chrome Web Browser on Windows 7 or later requires at least an i3 or equivalent processor and 128 MB of RAM, so our system will have analogous system requirements.

We've chosen to implement our front-end architecture using React since it provides performance, scalability, and excellent cross-platform support. Also, it's easy to adopt and has a small learning curve, and facilitates UI-focused designs, making it one of the best options for building an interactive user interface [1]. On the other hand, we chose Spring Boot Framework together with Java as the implementation language in order to deliver rapid and reliable

implementations. Spring Boot helps to complete most of the job through simple annotations when implementing endpoints, and it also provides automated queries to the database. Also, it has a small learning curve compared to other server-side frameworks, and the previous experiences of our team make it an even more profitable option. Most importantly, this framework makes it possible to provide the actual implementations of the architectural designs with different layers we did during the development process [2].

The Spring Boot application, PostgreSQL Database, and React Web server will run on the machines with the following hardware specifications:

- Intel Core i7-10750H processor
- 16GB DDR4 2667 MHz RAM
- 512 GB NVMe M.2 SSD
- Windows 10 Pro

  or
- Intel Core i5-10300H processor
- 8GB DDR4 2667 MHz RAM
- 512 GB NVMe M.2 SSD
- 1 TB 7200 RPM HDD
- Windows 11 Pro

Considering the nature of our application, any system having the minimum specifications of a raspberry pi and a minimum of 256MB ram will be enough for the deployment [3]. Hence, the above specifications are above the satisfactory level for our case.

2.3 Persistent Data Management

For developing the backend side of our web application, we will use the Spring Boot Framework together with the PostgreSQL Database, an open-source object-relational database. One of the main reasons for us to pick PostgreSQL as our database was Spring's great support and the current popularity of it as a couple in Spring RESTful Services. Since we plan to persist object-type data, PostgreSQL is again a favorable option for us because of its support for classes, objects, and inheritance, behaving similarly to an object-oriented database. To connect these two technologies, we will use the Spring Data JPA to provide a data access layer using Java Persistence API (JPA) [4]. This will provide an effortless implementation and improve the persistence code by only providing the repository interfaces and without writing complex SQL queries. The Spring Data JPA will provide automated implementations for the queries once

finder method signatures are written. Also, through the @Query annotation of Spring Data JPA, it's possible to write a custom query which will allow for even more fine-grained operations. Finally, for connecting to the database with our Spring Boot Application, we will use the "PostgreSQL Driver" (used for Java programs) of the Spring Boot Framework.

2.4 Access Control and Security

Securing the application and the data have great significance, as the application contains important information such as mail, phone number, ID, and CGPA of the users within. Therefore, each user has different authorization and verification levels. For example, the department coordinator can see the information of all outgoing students, while an outgoing student is not authorized to do so. In addition, external users will not be able to access the system as all users will be created and granted access by the system, which increases the security of the system and provides admin to control access of users. Passwords will be hashed to keep user information safe in case of any leak. Passwords will be kept in this form in the database. Furthermore, to increase security, the HTTPS protocol will be followed rather than the HTTP protocol.

2.4.1 Access Matrix

	Outgoing Student	Department Coordinator	Course Coordinator	Administrative Staff	Incoming Student
Login	x	x	x	x	x
Change Password	x	x	x	x	x
View Application List		x	x	x	
View Student Profile		x	x	x	
Cancel Application	x				
Create Course Request	x				
Create New Pre-Approval Form	x				

Request Acceptance Documents	**x**				
Edit an Announcement		**x**			
Add Language Proficiency		**x**			
View Application List		**x**	**x**	**x**	
Add an Announcement		**x**			
Respond to Pre-Approval Form			**x**		
Respond to Mandatory Course Request			**x**		
Respond to Elective Course Request			**x**		
Add a University				**x**	
Edit University Details				**x**	
View Document Requests				**x**	
Submit Document				**x**	
Cancel Placement	**x**				
Respond to Replacement Offer	**x**				

Upload Course Final Report	**x**				
View Announcement	**x**	**x**			
Send Replacement Offer		**x**			
Respond to Course Proposal				**x**	
Form Course Proposal					**x**
Edit Course Proposal					**x**
Submit Course Proposal					**x**
Upload Applications				**x**	

Figure 3: *Access Matrix*

2.5 Boundary Conditions

2.5.1 Initialization

The initialization process of the ErasMove application is going to start with running the XAMPP program to start the PostgreSQL database. Then, the com.erasmove.java file will be executed in the source file to make the backend accessible, enabling the REST API to function. The command "yarn start" will be used to launch the React project for the front-end web application. Initialization typically takes 4-5 minutes.

2.5.2 Termination

The opposite order of initialization should be used to correctly terminate the application in order to prevent termination failure. Start by terminating React Application by typing "ctrl + c" in the terminal. After that, quit the ErasMove Application.java program. Finally, use the "stop" button while the database is highlighted to terminate the PostgreSQL server.

2.5.3 Failure

The user will be taken to the login screen, and their login token will be terminated if an unexpected event results in a software problem. By doing this, we can avoid having invalid data or other issues corrupting the system's database. The system's security and reliability will improve as a result of this strategy. The project's goal is to have this kind of unexpected event happen infrequently and avoid degrading usability and user experience. Since we are utilizing a three-layered architecture, which makes it difficult to go down until the issue is in at least two layers, for the majority of failure scenarios, it is possible to recover the app directly into the stable state.

3. Low-level Design

3.1 Object Design Trade-Offs

Usability vs. Functionality: The system's ability to be easily used by everyone is one of its primary objectives. It is essential that the system be simple to use because the present method used by students and department coordinators at Bilkent University to manage the Erasmus and Exchange programs is complicated and time-consuming. The user would find it complicated if the system adopted an extensive number of functionalities because they would have to keep track of all the subsections and their individual functionalities. Therefore, in order to accomplish usability and make the system more favorable than the existing one, the program's features must be kept at a certain level. This will enable users to get what they need from the system without having to overcomplicate its functionalities.

Maintainability vs. Performance: Since we are using a 3-Layered architecture in the project, different layers cannot directly interact with deeper layers and must always communicate with the next layer. This separation reduces dependencies and provides significant maintenance benefits. However, this strict layering slightly reduces runtime performance because layers with different heights cannot be accessed directly. This situation results in relatively poor performance.

Reliability vs. Cost: Reliability is one of the factors we will concentrate on because most exceptions and errors will be handled by our application without causing any harm, and users will be informed with different cautions and messages for different types of errors. To accomplish this, it will require more time and the use of multiple test programs to find every possible error and properly handle them, increasing the cost of the project.

Usability vs. Security: Security is not one of the main purposes of this project.
While we wanted our application to be secure, we didn't want to sacrifice usability. Although adding necessary security steps helps to create a more secure platform, adding too many authentications damages the usability of the application. Security precautions are taken to protect your data, but these precautions are not extensive enough to affect your user experience and

usability; therefore, widespread use of certain protection methods will be limited in this application.

3.2 Final Object Design

3.2.1 Interface Layer

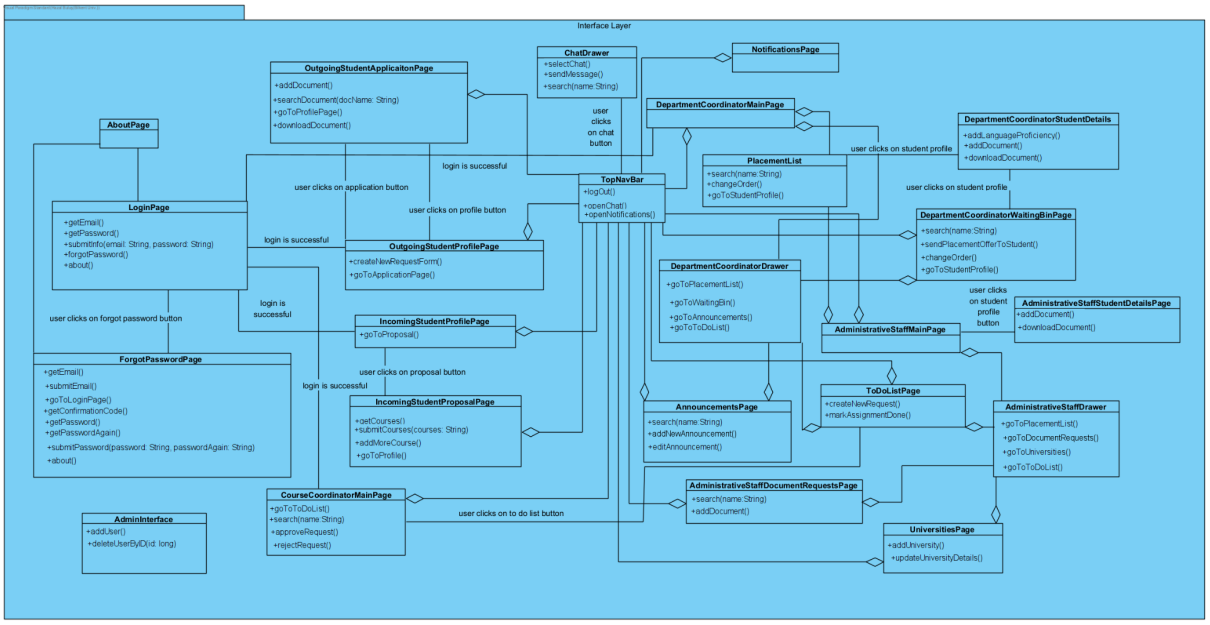

Figure 4: *Interface Layer Class Diagram(click here for larger size)*

3.2.2 Management Layer

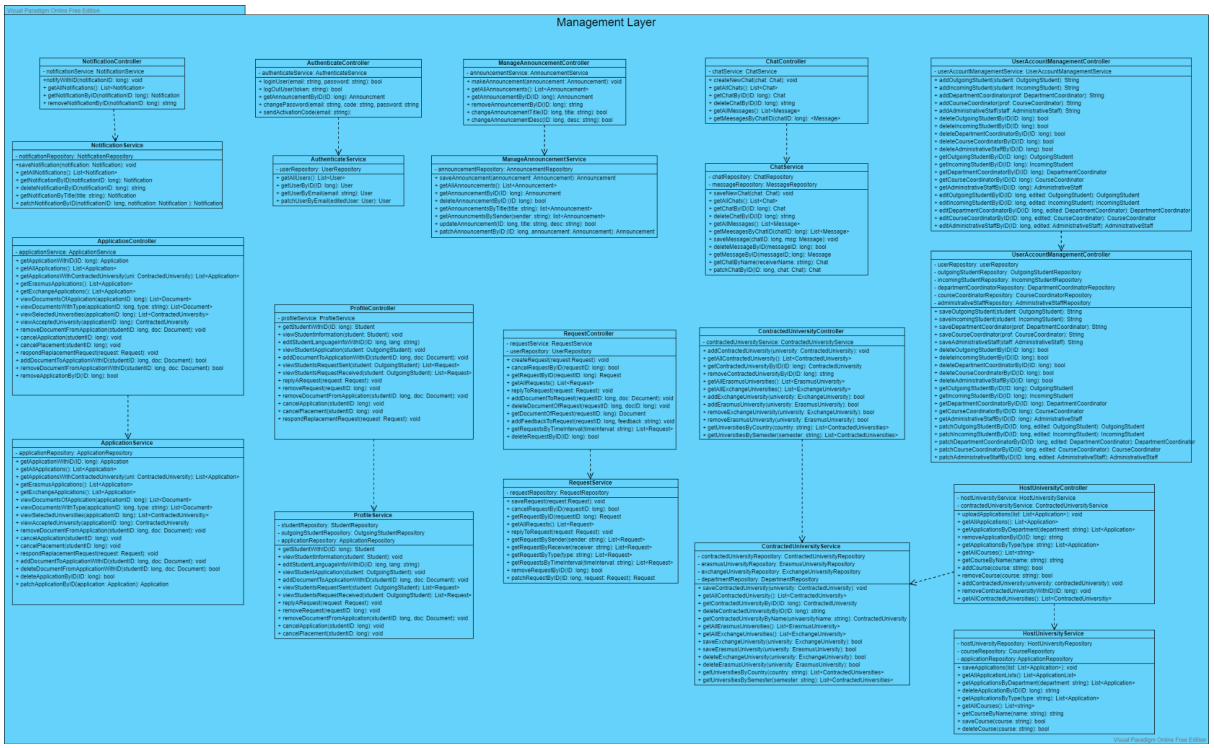

Figure 5: *Management Layer Class Diagram (Click here for larger size)*

3.2.3 Database Layer

3.2.3.1 Database

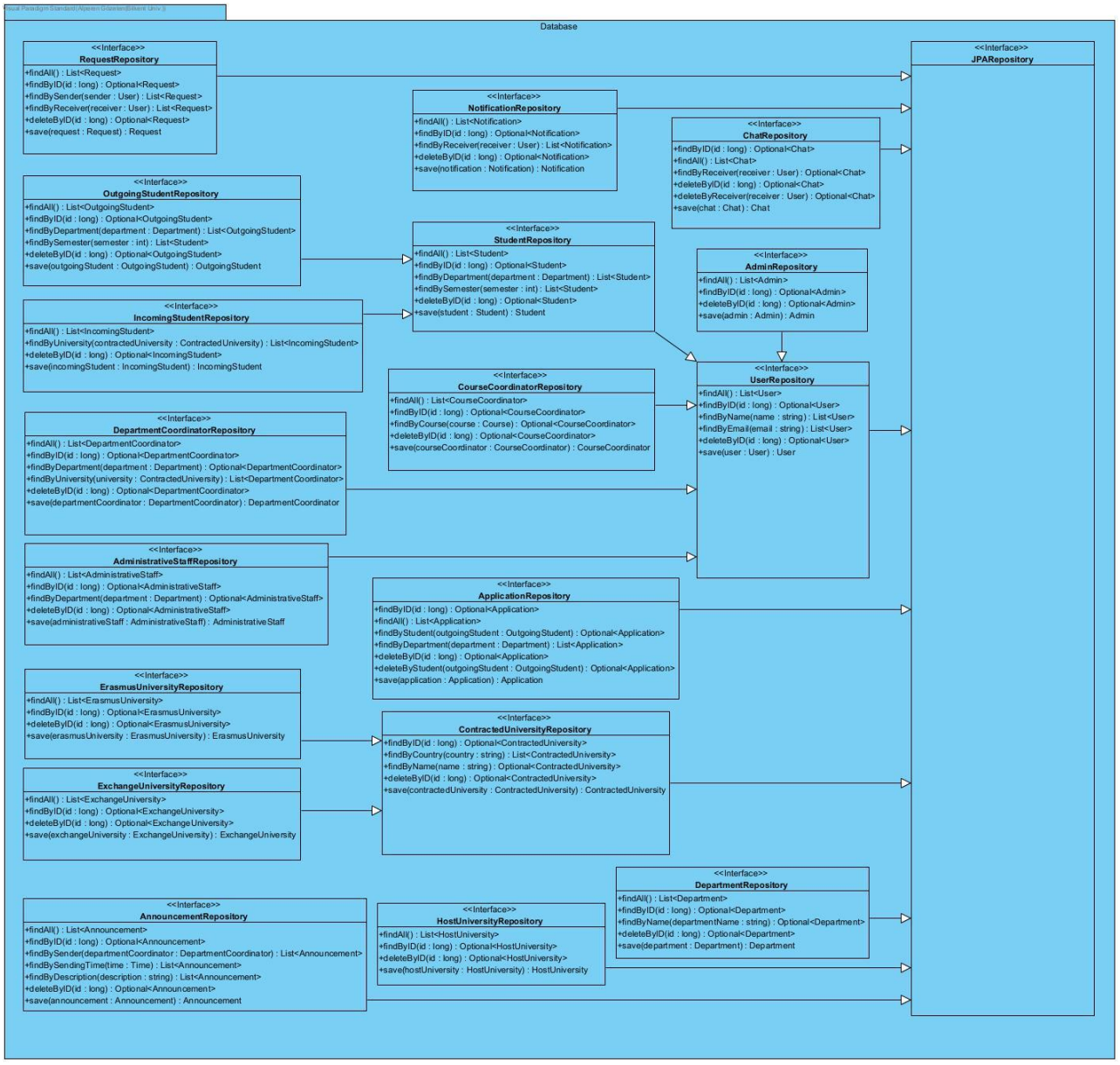

Figure 6: *Diagram of Interface Classes inside the Database Package as a part of Database Layer (Click here for larger size)*

3.2.3.2 Entity

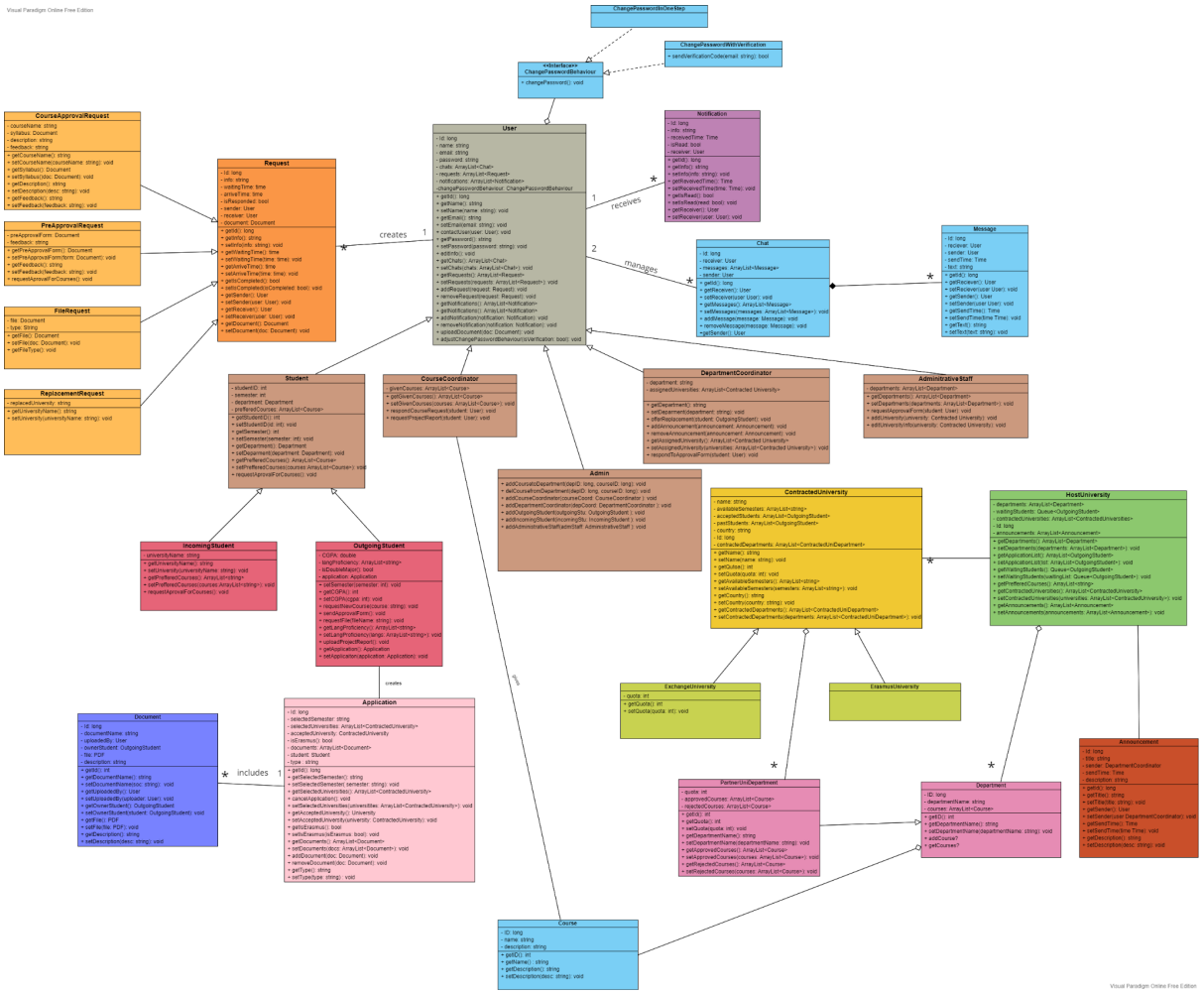

Figure 7: *Diagram of Classes of Entity Package in Database Layer (Click here for bigger size)*

3.2.3.3 Design Patterns

We implement our Model and Controller classes in a Spring Boot application and our View classes in a React web application to follow the Model-View-Controller (MVC) design pattern throughout the project. Since we are not restricted to views to test our backend code, their separation of concerns really benefited our development process. Additionally, this type of pattern utilization benefited during the backend or frontend maintenance of the program with regard to certain issues.

Also, in our models, we use Strategy and Singleton design patterns. For example, there are two different behaviours for changing a password for user: one is change directly from account settings when you are logged in and the other one is change it from forget password page with verification emai. This two behaviour have a common changePassword method whose implementation is different for them. This change password behaviours build our Strategy

pattern. This kind of pattern choice made our code clean because of the absence of nested if blocks. Also, we use singleton design pattern in Host University class since the application can have only one host university HostUniversity class can have only one instance. This design pattern prevents our application from having different instances of doing the same business. It allows our memory to be used efficiently.

Lastly, we use Facade design pattern to connect backend to frontend. We use API endpoints which interacts with all object models. Fronted views only interact with these API endpoints to get information from backend. This pattern choice made our code clean and independent to each other so we don't need to update anything from frontend unless API changes even if backend changes.

3.2.4 Final Object Diagram

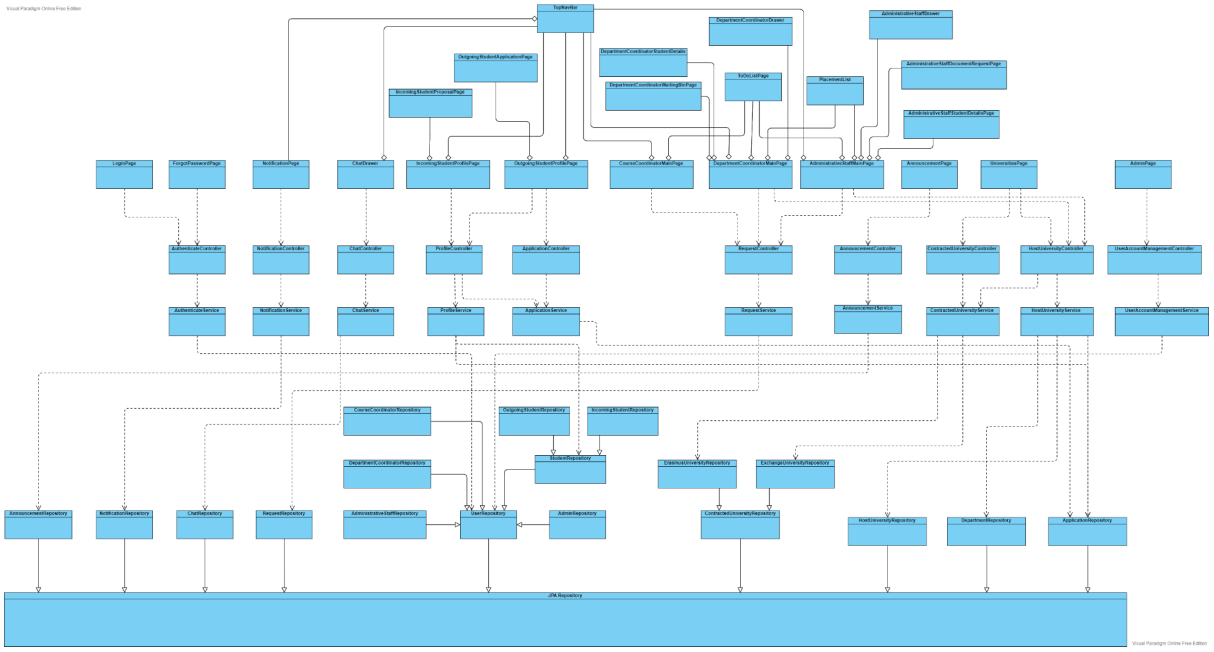

Figure 8: *Final Object Diagram (Click here for larger size)*

3.3 Packages

In our software, two types of packages are used. The first type of these two packages is the internal package, which is developed by us, and the other type of package is the external package, which is obtained by external APIs.

3.3.1 Internal Packages

These are the packages that are developed by us during the development, which include Model, Repository, Service, and Controller packages, and they have a hierarchy between them. The hierarchy mentioned here has a comparable logic to that of the Boundary-Control-Entity architectural model.

3.3.1.1 Model

This package contains the entities we specified in our class diagram. It has instance variables with getters, setters, and functions.

3.3.1.2 Repository

Repository interfaces are in this package for the models (entities) in the software. By using Spring Boot and Java Persistence API, the operations findByID and deleteByID, where ID is an instance of the model (entity), are predefined. So these operations are declared automatically without the need for a concrete class. Hence, they are interfaces.

3.3.1.3 Service

The major operations, such as Pre-Approval Form, File Request, Placement Cancellation, Profile, etc., in our application are in this package. For each of these major operations, we (usually) need an interface and a class. These classes implement the needed interfaces that define the needed method by using repository instance variables.

3.3.1.4 Controller

Controller objects for the services are in this package. In this package, the correct HTTP request endpoint for each service operation is generated. These HTTP request endpoints will serve to connect the backend and the frontend parts of the application.

3.3.1.5 View

View objects for the application ‘ErasMove’ are in this package. These objects are the JavaScript files that are implemented using the React framework of JavaScript. The server endpoints in the controller package are going to be called to fetch the data from the backend, and this data will be displayed in the front end.

3.3.2 External Packages

The following are the packages that we benefit from during the development of our system.

3.3.2.1 Spring Web

The Spring Web is an MVC framework that allows us to create restful web services by creating endpoints using different annotations like @Controller and @GetMapping, @RequestMapping and etc.

3.3.2.2 Spring Data JPA (internally uses Java Persistence API)

This package allows us to persist data to the database with our Java code using the Java Persistence API and reduces the boilerplate code that needs to be written for the queries by implementing them automatically.

3.3.2.3 Spring Boot PostgreSQL Driver

Through this API, we are able to connect our Java Code to the different tables inside the PostgreSQL databases and conduct queries on them.

3.3.2.3 Spring Boot OAuth2 Client

OAuth2 authorization framework is supported by Spring by providing a Client, which we will use for managing the authorization-related features of the users like login, logout, and reset password.

3.3.2.4 MaterialUI API

By using this API, a bunch of pre-designed UI elements in the React framework of JavaScript is accessible for our application. This provides a faster and more reliable development process with a flexible design in the user interface.

3.4 Class Interfaces

3.4.1 Interface Layer Explanation

All of the classes are implemented using JavaScript. Types of parameters are written to specify their intended types.

- **TopNavBar Class**

This class is seen on every page, and it directs the user to the chat or login page. Users can log out with the sign-out button in the TopNavBar.

Operations:

function logOut(): This function enables the user to log out and navigate the user back to the LoginPage class.

function openChat(): This function enables the user to open chat and redirect the user to ChatDrawer class.

function openNotifications(): This function enables the user to open the notifications drawer and redirect the user to the NotificationsPage class.

- **NotificationsPage Class**

This class will be responsible for displaying notifications that the user gets.

- **ChatDrawer Class**

This class will be responsible for displaying the chat drawer on the right side of each screen. Chat drawer enables users to send private messages to each other.

Operations:

function selectChat(): This function enables the user to select a chat.
function sendMessage(): This function enables the user to send a message to the selected chat.
function search(name:String): This function enables the user to search for a chat by the name of the contacted person.

- **PlacementList Class**

This class will be responsible for displaying the university placement list of outgoing students.

Operations:

function search(name:String): This function enables users to search for a student in the placement list by their name.
function changeOrder(): This function enables the user to change the order of the list.
function goToStudentProfile(): This function directs the user to the Student Profile Page.

- **LoginPage Class**

When users open the application, firstly, the LogInPage class appears. All users must have accounts in order to benefit from the features of the application.

Operations:

function getEmail(): This function will get the email input specified by the user.
function getPassword(): This function will get the password input specified by the user.
function about(): This function navigates users to the AboutPage class when users click on About button.
function forgotPassword(): This function navigates users to the ForgotPasswordPage class when users click the Forgot Password button.

- **ForgotPasswordPage Class**

This class will be responsible for displaying the necessary items needed to reset the password of the user.

Operations:

function getEmail(): This function will get the email input specified by the user.
function submitEmail(email:String): This function submits the input e-mail coming from the user, and it sends them to AuthenticateController.
function goToLoginPage(): This function directs the user to Login Page.
function getConfirmationCode(): This function will get the confirmation code that was sent user from the AuthenticateController and specified by the user.
function getPassword(): This function will get the password input specified by the user.
function getPasswordAgain(): This function will get the password input specified by the user a second time for confirmation of the password.
function submitPassword(password:String, passwordAgain:String): This function submits the input passwords coming from the user, and it sends them to AuthenticateController.
function about(): This function navigates users to the AboutPage class when users click on the About button.

- **AboutPage Class**

This class will be responsible for displaying the story and necessary information about the application as a read-only text.

- **OutgoingStudentProfilePage Class**

This class is the main page for the outcoming student users. When users successfully log in, they are directed to this page.

Operations:

function createNewRequestForm(): This function enables the user to create a new request.
function goToApplicationPage(): This function directs the user to the Application Page.

- **OutgoingStudentApplicationPage Class**

This class provides necessary items for an outgoing student to track their application process and add or download a document.

Operations:

function addDocument(): This function enables users to add a document to their profile.
function searchDocument(docName: String): This function enables the user to search a document by its name.
function goToProfilePage(): This function directs the user to Profile Page.
function downloadDocument(): This function enables the user to download a document that is uploaded to their application.

- **IncomingStudentProfilePage Class**

This class is the main page for incoming student users. When users successfully log in, they are directed to this page.

Operations:

function goToProposal(): This function directs the user to the IncomingStudentProposalPage class.

- **IncomingStudentProposalPage Class**

This class provides necessary items for incoming student to track their proposal and add courses.
Operations:
function getCourses(): This function will get the course names input by the user.
function submitCourses(courses:String): This function submits the input courses coming from the user, and it sends them to HostUniversityController.
function addMoreCourse(): This function enables the user to add more courses than 3 course.
function goToProfile(): This function directs the user to Profile Page.

- **CourseCoordinatorMainPage Class**

This class is the main page for the course coordinator users. When users successfully log in, they are directed to this page.

Operations:

function goToToDoList(): This function directs the user to To Do List Page.
function search(name:String): This function enables the user to search a request by the student name who sent the request.
function approveRequest(): This function enables the user to approve the coming request.

function rejectRequest(): This function enables the user to reject the coming request.

- **DepartmentCoordinatorMainPage Class**

This class is the main page for the Department Coordinator type of users. When users successfully log in, they are directed to this page.

- **DeparmentCoordinatorDrawer Class**

This class is the sidebar menu for the Department Coordinator actor interface in the project. It provides ease for users to switch between different pages.

Operations:

function goToPlacementList(): This function directs the user to the placement list page.
function goToWaitingBin(): This function directs the user to the waiting bin page.
function goToAnnouncements(): This function directs the user to the announcements page.
function goToToDoList(): This function directs the user to the To-Do list page.

- **DepartmentCoordinatorWaitingBinPage Class**

This class is responsible for displaying content for the management of the waiting list. It provides necessary items for searching, sending a student placement offer, and directing the user to the student profile.

Operations:

function search(name:String): This function enables users to search for a student in the waiting list by their name.
function sendPlacementOfferToStudent(): This function enables the user to send a placement offer to a student in the waiting list. This function can be used only if there is empty quota.
function changeOrder(): This function enables users to change the order of the list.
function goToStudentProfile(): This function directs the user to the Student Profile Page.

- **DepartmentCoordinatorStudentDetails Class**

This class is for Department Coordinator type of users only. They can display the profile page of selected students, and this class is responsible for displaying necessary items for adding and downloading documents and adding a language proficiency.

Operations:

function addLanguageProficiency(): This function enables users to add any language proficiency to student's profile according to their test results.
function addDocument(): This function enables the user to add a document to the profile of the student.
function downloadDocument(): This function enables the user to download a document from the profile of the student.

- **AnnouncementsPage Class**

This class will display content for the management of an announcement.

Operations:

function search(name:String): This function enables the user to search an announcement by its name.
function addNewAnnouncement(): This function enables the user to add a new announcement.
function editAnnouncement(): This function enables the user to edit an existing announcement.

- **ToDoListPage Class**

This class will display content for the management of a To-Do list.

Operations:

function createNewRequest(): This function enables the user to create a new request.
function markAssignmentDone(): This function enables the user to mark an assignment as done.

- **AdministrativeStaffMainPage Class**

This class is the main page for administrative staff users. When users successfully log in, they are directed to this page.

- **AdministrativeStaffDocumentRequestPage Class**

This class is responsible for displaying necessary items to request related actions, adding documents, and searching documents by name.

Operations:

function search(name:String): This function enables the user to search for a request by its name.
function addDocument(): This function enables the user to add a requested document.

- **AdministrativeStaffDrawer Class**

This class is the sidebar menu for the Administrative Staff actor interface in the project. It provides ease for users to switch between different pages.

Operations:

function goToPlacementList(): This function directs the user to the Placement List Page.
function goToDocumentRequests(): This function directs the user to the Document Requests Page.
function goToUniversities(): This function directs the user to the Universities Page.
function goToToDoList(): This function directs the user to the To Do List Page.

- **AdministrativeStaffStudentDetailsPage Class**

This class is for Administrative Staff type of users only. They can display the profile page of selected students, and this class is responsible for displaying necessary items for adding and downloading documents.

Operations:

function addDocument(): This function enables the user to add a document to the profile of the student.
function downloadDocument(): This function enables the user to download a document from the profile of the student.

- **UniversitiesPage Class**

This class will be responsible for displaying the details of contracted universities, adding a new university, or updating details of the existing university.

Operations:

function addUniversity(): This function enables the user to add a new university to the system.

function updateUniversityDetails(): This function enables the user to update the details of an existing university in the system.

- **AdminInterface Class**

This class will be responsible for displaying necessary items that are needed to add a user to the system and to delete a user from the system (for Admin).

Operations:

function addUser(): This function enables the admin to add a new user to the system.
function deleteUserByID(id: long): This function enables the admin to delete an existing user by its id from the system.

3.4.2 Management Layer Explanation

In the management layer, we have two objects for each major operation in our system, such as ManageRequest, ManageDocument, ManageProfile, Chat, ContractedUniversity, HostUniversity, etc. These two objects are Controller and Service Objects. Service objects access the needed data from Repository objects that are discussed on the database layer. Controller objects use Services to create endpoints for outside access to the system. For example, GET, POST, and PATCH requests are created in those objects for each major operation.

Generally, most of the service objects have the following methods in common: **getAll(): List<Object>:** Returns the list of all objects that are instances of the class Object.
getByID(ID: long): List<Object> : Returns the object whose ID is specified as list. Since ID is unique among all classes, either a 0 or 1 object will be returned. Hence, we decided to return as a list since dealing with empty lists is easier than dealing with null objects.
deleteByID(ID: long): boolean: This function returns true if the ID specified is on the repository after deleting the object with that ID.
patchByID(ID: long, newObj: Object): Object: This function edits the object on the repository whose id is given.
save(obj: Object) : Object: This function creates a new instance of an object in the repository.

In controller objects, to perform functional requirements in the user interface and make necessary changes in the database, we defined some methods. Controller objects depend on service objects to do these wanted changes. Some examples of the methods we defined in control objects are:

addDocumentToApplicationWithID(studentID: long, doc: Document): void: adds a given Document to the student's application. It takes the Document object and studentID as parameters, and it performs necessary changes in the ApplicationRepository of the related student by using ProfileService of the ProfileController.

respondReplacementRequest(requestID: long, request: Request): void: updates the Request with the responded Request. It takes requestID and Request as parameters and makes related changes in the RequestRepository by using RequestService and RequestController.

createRequest(request:Request): void: creates a new request. It takes Request as a parameter and makes related changes in the RequestRepository by using RequestService and RequestController.

editStudentLanguageInfoWithID(ID: long, lang: string): void: adds given language info to the student's profile. It takes the studentID and lang as parameters, and it performs necessary changes in the StudentRepository by using ProfileService of the ProfileController.

3.4.3 Database Layer Explanation

All of the below interfaces are implemented using Java and Spring Boot Framework; every interface will extend the generic interface called JpaRepository [5].

- RequestRepository Interface

This interface is a Spring Repository and is used to persist data related to the Request entities by providing different functionalities for queries.

function findAll(): This function is going to return the list of all Request entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the Request entities with matching id. Since the given ids are unique, it returns the container object Optional<Request>, which will contain null if there's no match, the Request object otherwise.

function findBySender(sender: User): This function is used to create a query to search for the Request entities with matching sender object (of User type) of the Request, which implicitly calls "equals" method on the User type objects. The function returns a list of all matched User objects, and the list will be empty if there's no matching User object.

function findByReceiver(receiver: User): This function is used to create a query to search for the Request entities with matching receiver object (of User type) of the Request, which implicitly calls "equals" method on the User type objects. The function returns a list of all matched User objects, and the list will be empty if there's no matching User object.

function deleteByID(id: long): This function searches for the Request entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<Request>, which will contain null if there's no match, the Request object otherwise.

function save(request: Request): This function saves the Request entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- NotificationRepository Interface

This Spring Repository interface is used to provide different functionalities (post, get, update, delete) for the Notification object to interact with the database.

function findAll(): This function is going to return the list of all Notification entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the Notification entities with matching id. Since the given ids are unique, it returns the container object Optional<Notification>, which will contain null if there's no match, the Notification object otherwise.

function findByReceiver(receiver: User): This function is used to create a query to search for the Notification entities with matching receiver object (of User type) of the Notification, which implicitly calls equals method on the objects. It returns the container object Optional<Notification>, which will contain null if there's no match, the Notification object otherwise.

function deleteByID(id: long): This function searches for the Notification entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<Notification>, which will contain null if there's no match, the Notification object otherwise.

function save(notification : Notification): This function saves the Notification entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- ChatRepository Interface

This interface is a Spring Repository and is used to persist data related to the Chat entities by providing different functionalities for queries.

function findAll(): This function is going to return the list of all Chat entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the Chat entities with matching id. Since the given ids are unique, it returns the container object Optional<Chat>, which will contain null if there's no match, the Chat object otherwise.

function findByReceiver(receiver: User): This function is used to create a query to search for the Chat entities with matching receiver object (of User type) of the Chat, which implicitly calls "equals" method on the User type objects. It returns the container object Optional<Chat>, which will contain null if there's no match, the Chat object otherwise.

function deleteByID(id: long): This function searches for the Notification entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<Notification>, which will contain null if there's no match, the Notification object otherwise.

function save(chat : Chat): This function saves the Notification entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- OutgoingStudentRepository Interface

This Spring Repository interface is used to provide different functionalities (post, get, update, delete) for the OutgoingStudent object to interact with the database.

function findAll(): This function is going to return the list of all OutgoingStudent entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the OutgoingStudent entities with matching id. Since the given ids are unique, it returns the container object Optional<OutgoingStudent>, which will contain null if there's no match, the OutgoingStudent object otherwise.

function findByDepartment(department : Department): This function is used to create a query to search for the OutgoingStudent entities with matching department object (of Department type) of the OutgoingStudent, which implicitly calls equals method on the objects. It returns the list of all OutgoingStudent objects with the matching department (by implicitly calling the equals method on the Department object), and the list will be empty if there's no matching OutgoingStudent object.

function findBySemester(semester : int): This function is used to create a query to search for the OutgoingStudent entities with matching attribute semester, which is of int type. The function

returns a list of all matched OutgoingStudent objects, and the list will be empty if there's no matching OutgoingStudent object.

function deleteByID(id: long): This function searches for the OutgoingStudent entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<OutgoingStudent>, which will contain null if there's no match, the OutgoingStudent object otherwise.

function save(outgoingStudent : OutgoingStudent): This function saves the OutgoingStudent entity as the parameter to the database. Updates the object in the database if the id already exists, creates a new one otherwise.

- IncomingStudentRepository Interface

This interface is a Spring Repository and is used to persist data related to the IncomingStudent entities by providing different functionalities for queries.

function findAll(): This function is going to return the list of all IncomingStudent entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the IncomingStudent entities with matching id. Since the given ids are unique, it returns the container object Optional<IncomingStudent>, which will contain null if there's no match, the IncomingStudent object otherwise.

function findByUniversity(contractedUniversity : ContractedUniversity): This function is used to create a query to search for the IncomingStudent entities with matching contractedUniversity object (of ContractedUniversity type) of the IncomingStudent, which implicitly calls "equals" method on the ContractedUniversity type objects. It returns the list of all IncomingStudent objects with the matching department, and the list will be empty if there's no matching IncomingStudent object.

function deleteByID(id: long): This function searches for the IncomingStudent entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<IncomingStudent>, which will contain null if there's no match, the IncomingStudent object otherwise.

function save(incomingStudent : IncomingStudent): This function saves the IncomingStudent entity as the parameter to the database. Updates the object in the database if the id already exists, creates a new one otherwise.

- StudentRepository Interface

This Spring Repository interface is used to provide different functionalities (post, get, update, delete) for the Student object to interact with the database.

function findAll(): This function is going to return the list of all Student entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the Student entities with matching id. Since the given ids are unique, it returns the container object Optional<Student>, which will contain null if there's no match, the Student object otherwise.

function findByDepartment(department : Department): This function is used to create a query to search for the Student entities with matching department object (of Department type) of the Student, which implicitly calls "equals" method on the Department type objects. It returns the list of all Student objects with the matching department, and the list will be empty if there's no matching Student object.

function findBySemester(semester : int): This function is used to create a query to search for the Student entities with matching attribute semester, which is of int type. The function returns a list of all matched Student objects, and the list will be empty if there's no matching Student object.

function deleteByID(id: long): This function searches for the Student entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<Student>, which will contain null if there's no match, the Student object otherwise.

function save(student : Student): This function saves the Student entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- AdminRepository Interface

This interface is a Spring Repository and is used to persist data related to the Admin entities by providing different functionalities for queries.

function findAll(): This function is going to return the list of all Admin entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the Admin entities with matching id. Since the given ids are unique, it returns the container object Optional<Admin>, which will contain null if there's no match, the Admin object otherwise.

function deleteByID(id: long): This function searches for the Admin entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the

database and returns container object Optional<Admin>, which will contain null if there's no match, the Admin object otherwise.

function save(admin : Admin): This function saves the Admin entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- CourseCoordinatorRepository Interface

This Spring Repository interface is used to provide different functionalities (post, get, update, delete) for the CourseCoordinator object to interact with the database.

function findAll(): This function is going to return the list of all CourseCoordinator entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the CourseCoordinator entities with matching id. Since the given ids are unique, it returns the container object Optional<CourseCoordinator>, which will contain null if there's no match, the OutgoingStudent object otherwise.

function findByCourse(course : Course): This function is used to create a query to search for the CourseCoordinator entities with matching course object (of Course type) of CourseCoordinator, which implicitly calls "equals" method on the Course type objects. It returns the container object Optional<Course>, which will contain null if there's no match, the Course object otherwise.

function deleteByID(id: long): This function searches for the CourseCoordinator entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<CourseCoordinator>, which will contain null if there's no match, the CourseCoordinator object otherwise.

function save(courseCoordinator : CourseCoordinator): This function saves the CourseCoordinator entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- DepartmentCoordinatorRepository Interface

This interface is a Spring Repository and is used to persist data related to the DepartmentCoordinator entities by providing different functionalities for queries.

function findAll(): This function is going to return the list of all DepartmentCoordinator entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the DepartmentCoordinator entities with matching id. Since the given ids are unique, it returns the container object Optional<DepartmentCoordinator>, which will contain null if there's no match, the DepartmentCoordinator object otherwise.

function findByDepartment(department : Department): This function is used to create a query to search for the DepartmentCoordinator entities with matching course object (of Department type) of the DepartmentCoordinator, which implicitly calls "equals" method on the Department type objects. It returns the container object Optional<DepartmentCoordinator>, which will contain null if there's no match, the found DepartmentCoordinator object otherwise.

function findByUniversity(university : ContractedUniversity): This function is used to create a query to search for the DepartmentCoordinator entities with matching university object (of ContractedUniversity type) of the DepartmentCoordinator, which implicitly calls "equals" method on the ContractedUniversity type objects. It returns the list of all DepartmentCoordinator objects with the matching department, and the list will be empty if there's no matching DepartmentCoordinator object.

function deleteByID(id: long): This function searches for the DepartmentCoordinator entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<DepartmentCoordinator>, which will contain null if there's no match, the DepartmentCoordinator object otherwise.

function save(departmentCoordinator : DepartmentCoordinator): This function saves the DepartmentCoordinator entity as the parameter to the database. Updates the object in the database if the id already exists, creates a new one otherwise.

- AdministrativeStaffRepository Interface

This Spring Repository interface is used to provide different functionalities (post, get, update, delete) for the AdministrativeStaff object to interact with the database.

function findAll(): This function is going to return the list of all AdministrativeStaff entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the AdministrativeStaff entities with matching id. Since the given ids are unique, it returns the container object Optional<AdministrativeStaff>, which will contain null if there's no match, the found AdministrativeStaff object otherwise.

function findByDepartment(department : Department): This function is used to create a query to search for the AdministrativeStaff entities with matching course object (of Department

type) of the AdministrativeStaff, which implicitly calls “equals” method on the Department type objects. It returns the container object Optional<AdministrativeStaff>, which will contain null if there’s no match, the found AdministrativeStaff object otherwise.

function deleteByID(id: long): This function searches for the AdministrativeStaff entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<AdministrativeStaff>, which will contain null if there’s no match, the AdministrativeStaff object otherwise.

function save(administrativeStaff : AdministrativeStaff): This function saves the AdministrativeStaff entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- UserRepository Interface

This interface is a Spring Repository and is used to persist data related to the User entities by providing different functionalities for queries.

function findAll(): This function is going to return the list of all User entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the User entities with matching id. Since the given ids are unique, it returns the container object Optional<User>, which will contain null if there’s no match, the found User object otherwise.

function findByName(name : string): This function is used to create a query to search for the User entities with matching attribute name, which is of string type. The function returns a list of all matched User objects, and the list will be empty if there’s no matching User object.

function findByEmail(email : string): This function is used to create a query to search for the User entities with matching attribute email, which is of string type. The function returns a list of all matched User objects, and the list will be empty if there’s no matching User object.

function deleteByID(id: long): This function searches for the User entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<User>, which will contain null if there’s no match, the User object otherwise.

function save(user : User): This function saves the User entity as the parameter to the database. Updates the object in the database if the id already exists, creates a new one otherwise.

- ApplicationRepository Interface

This Spring Repository interface is used to provide different functionalities (post, get, update, delete) for the Application object to interact with the database.

function findAll(): This function is going to return the list of all DepartmentCoordinator entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the DepartmentCoordinator entities with matching id. Since the given ids are unique, it returns the container object Optional<DepartmentCoordinator>, which will contain null if there's no match, the DepartmentCoordinator object otherwise.

function findByStudent(outgoingStudent : OutgoingStudent): This function is used to create a query to search for the DepartmentCoordinator entities with matching course object (of Department type) of the DepartmentCoordinator, which implicitly calls "equals" method on the Department type objects. It returns the container object Optional<DepartmentCoordinator>, which will contain null if there's no match, the found DepartmentCoordinator object otherwise.

function findByDepartment(department : Department): This function is used to create a query to search for the Student entities with matching department object (of Department type) of the Student, which implicitly calls "equals" method on the Department type objects. It returns the list of all Student objects with the matching department, and the list will be empty if there's no matching Student object.

function findByUniversity(university : ContractedUniversity): This function is used to create a query to search for the DepartmentCoordinator entities with matching university object (of ContractedUniversity type) of the DepartmentCoordinator, which implicitly calls "equals" method on the ContractedUniversity type objects. It returns the list of all DepartmentCoordinator objects with the matching department, and the list will be empty if there's no matching DepartmentCoordinator object.

function deleteByID(id: long): This function searches for the DepartmentCoordinator entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<DepartmentCoordinator>, which will contain null if there's no match, the DepartmentCoordinator object otherwise.

function save(departmentCoordinator : DepartmentCoordinator): This function saves the DepartmentCoordinator entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- ExchangeUniversityRepository Interface

This interface is a Spring Repository and is used to persist data related to the ExchangeUniversity entities by providing different functionalities for queries.

function findAll(): This function is going to return the list of all ExchangeUniversity entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the ExchangeUniversity entities with matching id. Since the given ids are unique, it returns the container object Optional<ExchangeUniversity>, which will contain null if there's no match, the ExchangeUniversity object otherwise.

function deleteByID(id: long): This function searches for the ExchangeUniversity entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<ExchangeUniversity>, which will contain null if there's no match, the ExchangeUniversity object otherwise.

function save(exchangeUniversity : ExchangeUniversity): This function saves the ExchangeUniversity entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- ErasmusUniversityRepository Interface

This Spring Repository interface is used to provide different functionalities (post, get, update, delete) for the ErasmusUniversity object to interact with the database.

function findAll(): This function is going to return the list of all ErasmusUniversity entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the ErasmusUniversity entities with matching id. Since the given ids are unique, it returns the container object Optional<ErasmusUniversity>, which will contain null if there's no match, the ErasmusUniversity object otherwise.

function deleteByID(id: long): This function searches for the ErasmusUniversity entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<ErasmusUniversity>, which will contain null if there's no match, the ErasmusUniversity object otherwise.

function save(erasmusUniversity : ErasmusUniversity): This function saves the ErasmusUniversity entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- ContractedUniversityRepository Interface

This interface is a Spring Repository and is used to persist data related to the ContractedUniversity entities by providing different functionalities for queries.

function findAll(): This function is going to return the list of all ContractedUniversity entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the ContractedUniversity entities with matching id. Since the given ids are unique, it returns the container object Optional<ContractedUniversity>, which will contain null if there's no match, the ContractedUniversity object otherwise.

function findByCountry(country : string): This function is used to create a query to search for the ContractedUniversity entities with matching attribute country, which is of string type. The function returns a list of all matched ContractedUniversity objects, and the list will be empty if there's no matching ContractedUniversity object.

function findByName(name : string): This function is used to create a query to search for the ContractedUniversity entities with matching attribute name, which is of string type. Since the given names to the universities should be unique, it returns the container object Optional<ContractedUniversity>, which will contain null if there's no match, the ContractedUniversity object otherwise.

function deleteByID(id: long): This function searches for the ContractedUniversity entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<ContractedUniversity>, which will contain null if there's no match, the ContractedUniversity object otherwise.

function save(contractedUniversity : ContractedUniversity): This function saves the ContractedUniversity entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- DepartmentRepository Interface

This Spring Repository interface is used to provide different functionalities (post, get, update, delete) for the Department object to interact with the database.

function findAll(): This function is going to return the list of all Department entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the Department entities with matching id. Since the given ids are unique, it returns the container object Optional<Department>, which will contain null if there's no match, the Department object otherwise.

function findByName(departmentName : string): This function is used to create a query to search for the Department entities with matching attribute name, which is of string type. Since the given names to the universities should be unique, it returns the container object Optional<Department>, which will contain null if there's no match, the Department object otherwise.

function deleteByID(id: long): This function searches for the Department entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<Department>, which will contain null if there's no match, the Department object otherwise.

function save(department : Department): This function saves the Department entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- HostUniversityRepository Interface

This interface is a Spring Repository and is used to persist data related to the HostUniversity entities by providing different functionalities for queries.

function findAll(): This function is going to return the list of all HostUniversity entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the HostUniversity entities with matching id. Since the given ids are unique, it returns the container object Optional<HostUniversity>, which will contain null if there's no match, the HostUniversity object otherwise.

function deleteByID(id: long): This function searches for the HostUniversity entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<HostUniversity>, which will contain null if there's no match, the HostUniversity object otherwise.

function save(hostUniversity : HostUniversity): This function saves the HostUniversity entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- AnnouncementRepository Interface

This Spring Repository interface is used to provide different functionalities (post, get, update, delete) for the Announcement object to interact with the database.

function findAll(): This function is going to return the list of all Announcement entities saved in the database. The function will return an empty list if the database is empty.

function findByID(id : long): This function is used to create a query to search for the Announcement entities with matching id. Since the given ids are unique, it returns the container object Optional<Announcement>, which will contain null if there's no match, the Announcement object otherwise.

function findBySender(departmentCoordinator : DepartmentCoordinator): This function is used to create a query to search for the Announcement entities with matching sender object (of DepartmentCoordinator type) of the Announcement, which implicitly calls "equals" method on the DepartmentCoordinator type objects. The function returns a list of all matched Announcement objects, and the list will be empty if there's no matching Announcement object.

function findBySendingTime(time : Time): This function is used to create a query to search for the Announcement entities with matching time object (of Time type) for the sendTime attribute of the Announcement, which implicitly calls "equals" method on the Time type objects. The function returns a list of all matched Announcement objects, and the list will be empty if there's no matching Announcement object.

function findByDescription(description : string): This function is used to create a query to search for the Announcement entities with matching attribute description, which is of string type. The function returns a list of all matched Announcement objects, and the list will be empty if there's no matching Announcement object.

function deleteByID(id: long): This function searches for the Announcement entity existing in the database with the matching id. Since the given ids are unique, it deletes the object from the database and returns container object Optional<Announcement>, which will contain null if there's no match, the Announcement object otherwise.

function save(announcement : Announcement): This function saves the Announcement entity as the parameter to the database. Updates the object in the database if the id already exists and creates a new one otherwise.

- JpaRepository Interface

This repository is a generic repository interface as a part of Spring Data JPA, and all of our interface classes will extend this interface in order to conduct auto-implemented queries to the database tables.

4. References

[1] A. Deshpande, “7 advantages of reactjs for Building Interactive User Interfaces,” *Clarion Tech*. [Online]. Available: https://www.clariontech.com/blog/7-advantages-of-reactjs-for-building-interactive-user-interfaces. [Accessed: 27-Nov-2022].

[2] A. A. K. W. at B. Agile, Author: and W. at B. Agile, “Pros and cons of using Spring Boot,” *Insights*, 08-Jun-2022. [Online]. Available: https://bambooagile.eu/insights/pros-and-cons-of-using-spring-boot/. [Accessed: 27-Nov-2022].

[3] “R/java - system requirements for a Spring boot app?,” *reddit*. [Online]. Available: https://www.reddit.com/r/java/comments/givind/system_requirements_for_a_spring_boot_app/. [Accessed: 29-Nov-2022].

[4] “Spring Data JPA,” *Spring*. [Online]. Available: https://spring.io/projects/spring-data-jpa. [Accessed: 27-Nov-2022].

[5] “Interface jparepository,” *JpaRepository (Spring Data JPA Parent 3.0.0 API)*, 18-Nov-2022. [Online]. Available: https://docs.spring.io/spring-data/data-jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html. [Accessed: 28-Nov-2022].