Bilkent University

Department of Computer Engineering

CS319 Term Project

Erasmust: Bilkent's Erasmus and Exchange Application

Detailed Design Report

Group "BubbleZort" - 2C

Nisa Yılmaz - 22001849

Tolga Özgün - 22003850

Eylül Badem - 22003079

Yahya Eren Demirel - 21903026

Emirhan Büyükkonuklu - 22003049

Barış Yıldırım - 22003175

Instructor: Eray Tüzün

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

Contents

Design Report

1 Introduction

Erasmust is an Erasmus and Exchange tracking and managing application created by the team BubbleZort. Erasmust was created as a solution to Bilkent's way of handling Erasmus and Exchange programs. Our project is a website that aims to simplify the Erasmus/Exchange application process for students, coordinators, instructors and all other stakeholders.

Erasmust helps students in their process of creating their Erasmus or Exchange applications and tracking them from a single software, thus minimizing the reliance on emails. By simplifying the process in a single application with clear instructions, Erasmust saves time and resolves most of the confusion generated by the existing system.

If the student is eligible to join the program of their choice, Erasmust will enable them to complete other steps of the application process. By almost eliminating all emailing, Erasmust will enable an easier tracking of the application for everyone. Students will be able to view/edit their application. Other parties involved, such as instructors and coordinators, will be able to see the status of applications they are responsible for in real-time thus making the whole process a lot easier and faster.

1.1 Purpose of the System

The system is a web-based application that aims to facilitate the application processes of Bilkent University students to Erasmus-Exchange programs, and to reduce the workload on the personnel (such as coordinators, instructors) involved in these applications and processes by minimizing the manual communication in these application processes. Another purpose of the system is to contribute to the much more controlled and systematic operation of the processes.

1.2 Design Goals

Our application mainly focuses on two design goals: Security and Functionality. These design goals were chosen as a result of joint decision since preventing users' applications and personal information from being viewed/changed by third parties and providing functionality to its users, which is the main purpose of the application, are considered as the most important criteria.

1.2.1 Security

In this software design, there is some sensitive data flowing through. If this software is to be deployed, these sensitive data collections should be secured due to data privacy issues. We as the developer team do not want any unauthorized person to be able to edit or even cancel any sort of exchange/erasmus application of any candidate. We plan to use advanced security methodologies in order to maximize the safety of sensitive personal data such as other than using classical username/password authentication methodology, we also want to enable any user to have the option to use a two-factor authentication system in order to boost the security of any account. We are completely aware of the importance of data privacy and the need for a rigorous approach through the concept of security.

1.2.2 Functionality

Functionality is of obvious importance. Since this is its main purpose, we can only assume that our system works if it can save the potential user a significant amount of time. Therefore, it should save the user time, it should guide the user about what to do (with clear explanations and a clear interface), and should provide an environment that carries out the process of the system on a single platform, facilitates the follow-up of the process and the storage of the forms. In short, the system should provide most of the standard functionality and only leave the user the implementation of the parts that must be done manually. And even then, the user should be supported and guided by our system on what to do.

2 High Level Software Architecture

2.1 Subsystem Decomposition

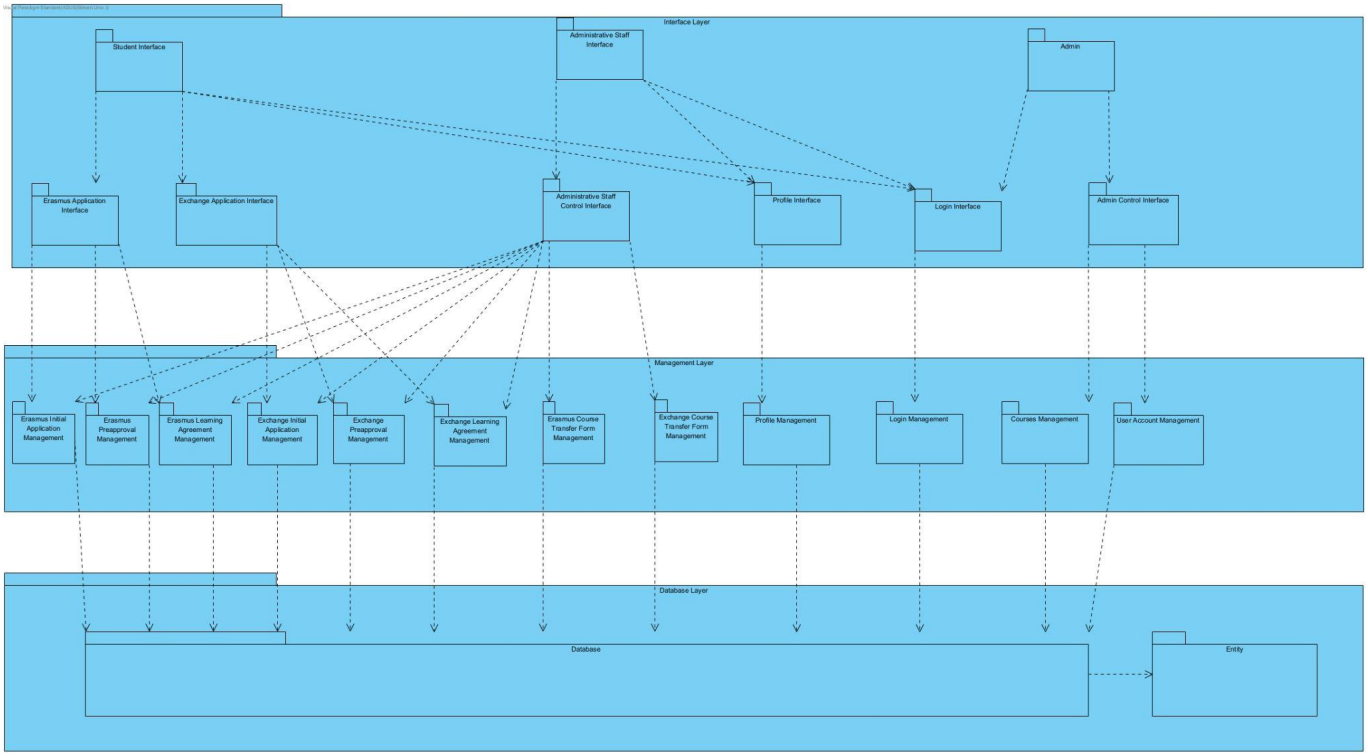

Fig. 1. Subsystem Decomposition Diagram of the System (Click here for full version)

2.2 Hardware/Software Mapping

Erasmust does not require any high-profile hardware to access. And it will be available for any device that has browsers like Google Chrome, Mozilla Firefox, Microsoft Edge. Devices include tablets, computers, smart phones.

At the front-end Erasmust uses HTML5, CSS3, Bootstrap and React. Aside from the fact that all of the front-end developers have experience in React, the reasons behind choosing React are; it offers an optimized development and its API is lightweight and drives fast performance as well as a stress-free development workflow. It is widely used so it has a huge community to get help. Spring is used at the back-end. The reasons behind using Spring are; It supports MVC design pattern so that it matches the objective of the course and potentially loose coupling by doing so. And like React it is popular so it is easy to get help when stuck.

Recommended server requirements are as the following; 16GB RAM, 400GB SSD and 8 core processor. This specification matches the setup of the host Ubuntu server, hosted in Godaddy with unlimited bandwidth. This setup is guaranteed to run Erasmust's services, with the support of Docker containers. Also, with the help of load balancing, more services can be installed to match exceeding load if requested. However, the scope of this project does not require a load balancing system, as the number of concurrent users are not expected to be higher than React, PostgreSQL or Spring Boot requirements. Therefore, a load balancing system is not installed by default.

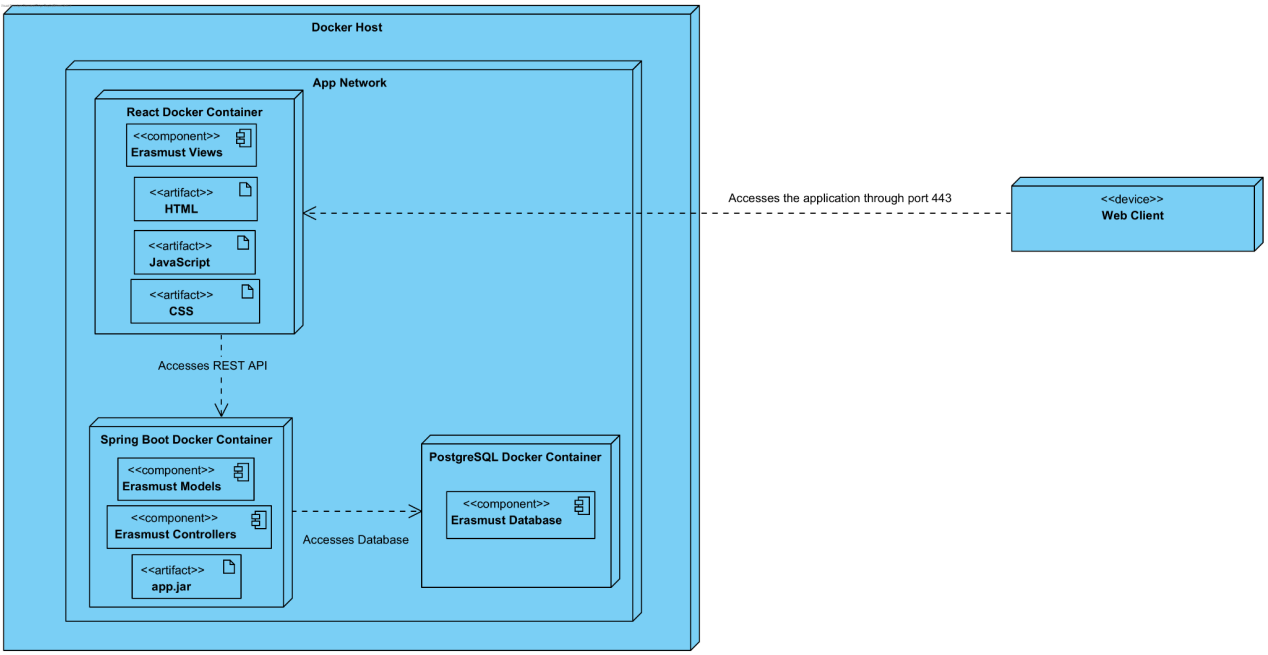

Fig. 2. Deployment Diagram of the System (Click here for full version)

2.3 Persistent Data Management

Erasmust's back-end is implemented using Spring Framework and Java. Internal and external packages were selected in accordance with this system. The implementation

of the database was carried out by using Hibernate, one of the frameworks offered by Java as open source, and ORM (Object Relational Mapping) operations. With the HQL (Hibernate Query Language) offered by Hibernate, the SQL implementation was carried out in an object-oriented manner, and the development process of the project was accelerated by using the ready-made functions offered by the JPA (Java Persistence API) to the developers. At this point, the reason why Hibernate-optimized JpaRepository was chosen instead of base CrudRepository is that JpaRepository offers paging and sorting algorithms besides CRUD operations. PostgreSQL is the database used to save the information in the system and make it available again when necessary. The reason for using SQL instead of NoSQL is to keep the Java objects created in the system in connection with each other in the system and to facilitate the tracking of the attributes of the objects. The reason why PostgreSQL is preferred over other database systems is that the system is used by many popular software companies, it allows working with complex data types, and one of the most important reasons is that the members of the group have previous experience with this database system. These tools, which are integrated into the back-end of Erasmust, also provide various conveniences when communicating with the front-end. Working integrated with Spring Framework, JPA (Java Persistence API) also enables various Http requests to be handled. Allowing Java classes to communicate with Annotations and the tools in it, Spring provides data transfer between backend and frontend with annotations such as @GetMapping and @PostMapping to handle CRUD operations. Hibernate, JPARepository and PostgreSQL were used in the database implementation of the project, as it facilitates the communication of the back-end and the front-end, contains ready-made search methods and is a popular tool by many companies.

2.4 Access Control and Security

The application processes lots of sensitive personal data such as users' name, surname, email, phone number, student ID and application password which should be stored and processed securely. In order to ensure whether the user authentication process is completed without any security and privacy issues or not, the application includes multi-layered authentication. First layer includes a password-based

authentication which is enabled by letting the user pick a password for her/himself after registering for the application. Second layer includes a two-factor authentication which a randomly generated six figure integer is to be sent to the users' email each time users try to log in to the application. Since students and each staff member has different task descriptions and levels of authorization, there is also a token authentication system that checks the level of authorization and then authenticates the users to related parts of the application. There is also AES256 Military Grade Password Encryption to be used with salt in order to encrypt all sorts of sensitive personal data and compare the login data with stored data which would boost the safety of the data stored for any account. In order to prevent the possible use of the same computer with the same session by any unauthorized person, there is a preset session time which after that time interval, terminates the session of the user and logs off the account back to the login page.

2.4.1 Access Matrix

Admin users should have all permissions in case data needs to be modified manually.

	Admin	Student	Erasmus Coordinator	Exchange Coordinator	Course Coordinator
Login	x	x	x	x	x
Add User	x				
Edit User	x	x(self)	x(self)	x(self)	x(self)
Delete User	x				
Change Password	x	x	x	x	x
Create/Edit/ Delete Erasmus Application	x	x			
View Erasmus Application Details	x	x(self)	x(all)		x(all)

Create/Edit/ Delete Exchange Application	x	x			
View Exchange Application Details	x	x(self)		x(all)	x(all)
Create/Edit/ Delete Preapproval Form	x	x			
View Preapproval Form	x	x(self)	x(all)	x(all)	x(all)
Approve /Reject Preapproval Form	x		x	x	x
Create/Edit/ Delete Learning Agreement	x				
View Learning Agreement	x	x	x	x	
Sign Learning Agreement	x		x	x	
Create/Edit/ Delete/View Course Transfer Form	x		x	x	

Table 1. Access Matrix

2.5 Boundary Conditions

2.5.1 Initialization

During the initialization stage of the Erasmust servers, all systems will go through planned tests using Jenkins software. Then, the Docker containers will be built and deployed to Dockerhub. After that, the host will pull the latest image to start the server. The data will be stored in the local storage of the server, however a scheduled data backup job will be run every day at 4:00 AM TRT (Turkish Regional Time). This will allow for an easy revert in case of emergency. During startup, the server will output a static HTML page of maintenance. This is the hard failure state (see 2.5.3 Failure).

2.5.2 Termination

Erasmust does not have any low-power or regression state, in which some subsystems are closed. Therefore, the system is built on the precondition that all subsystems should be active and running to prevent any unwanted circumstances. In the case of a termination of a subsystem, other subsystems will be notified. Firstly, the crash log will be saved into a predefined log storage area, and the Docker volumes will be saved into a different storage area, to prevent overriding onto safe data. Lastly, the whole system will go into the failure state, and will try to reboot afterwards. The latest updates will not be saved into a database, however will be stored as a different file. The admins can review these logs to determine if the new data is safe to push into the database, and will push to the database either directly or after making some changes.

2.5.3 Failure

In the case of an occurrence of failure in any subsystem, the user's session is ended by the system to prevent a loss of data. User is prompted by this event by a pop-up and forwarded to the splash page. This event is also persisted into the disk so that the admins will be able to retrieve the data back. Also, any expected maintenance will be announced to all clients beforehand with a closable notification so that all users will

act with the knowledge of the maintenance. This will allow them to save their data. Also, a data persistence layer will be added for planned maintenance by taking a snapshot of the system before going into maintenance. This will allow the system operators to roll-back to a working state with ease in case of emergency. In the case of system failure, the system administrators will be mailed or messaged (depending on their preferences) so that they can debug the issue and load the latest stable build. As Spring, PostgreSQL and React hold detailed logs, preventing data loss will be easier for admins. The system will try to reboot, and will perform some tests through Jenkins to check if it is stable to reboot. Then it will rebuild its Docker containers and run. If the tests fail, or any new exceptions are generated, then the system administrators will be notified with a high severity tag, and the system will go into maintenance mode. This maintenance mode will only serve a static HTML page where it will notify all clients that the server is in maintenance.

3 Low Level Design

3.1 Object Design Trade-offs

Security vs Usability: Erasmust aims to protect sensitive user information as well as prevent people from accessing other accounts and affecting others' Erasmus/Exchange applications therefore there will be a 2 Factor Authentication system to ensure secure login. Although 2FA increases security of the system it may decrease usability. In Erasmust, every time a user wants to log in, they will have to enter the code sent to their mail address which will decrease the usability. But since security is a greater concern than usability and 2FA alone will not be extremely detrimental to usability, this will not create a significant problem for the users.

Functionality vs Usability: Erasmust's main goal is to digitalize all steps of Erasmus and Exchange application process. To achieve this, the system must provide many different functionalities to the users. As the number of different functionalities increase, the usability may decrease. For Erastmust's case, this does not create a problem since the majority of users are capable of using complex and functional systems, such as SRS.

3.2 Final Object Design

3.2.1 Interface Layer

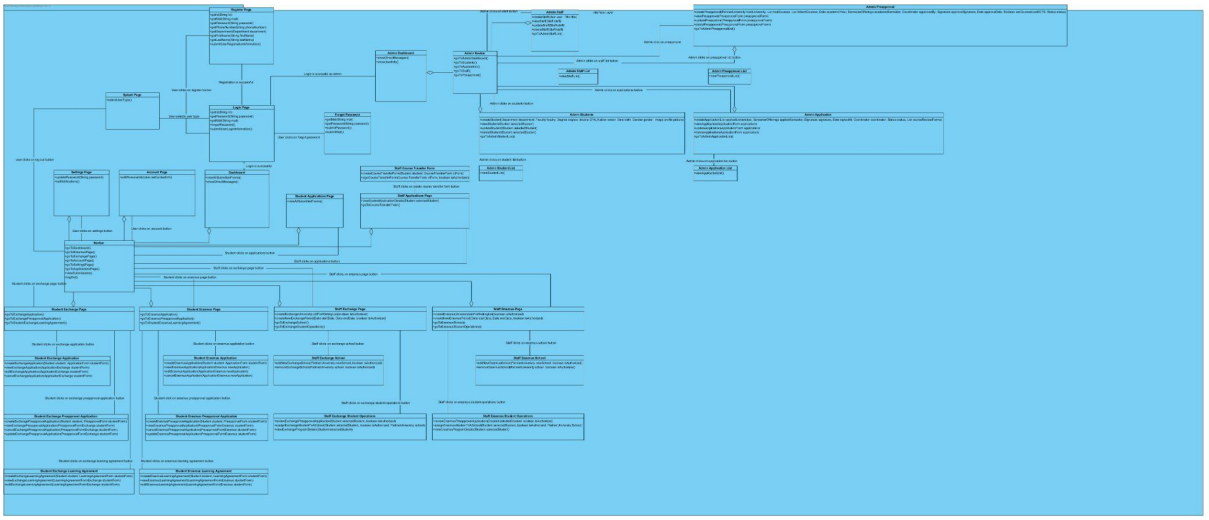

Fig. 3. Interface Layer of the System (Click here for full version)

3.2.2 Management Layer

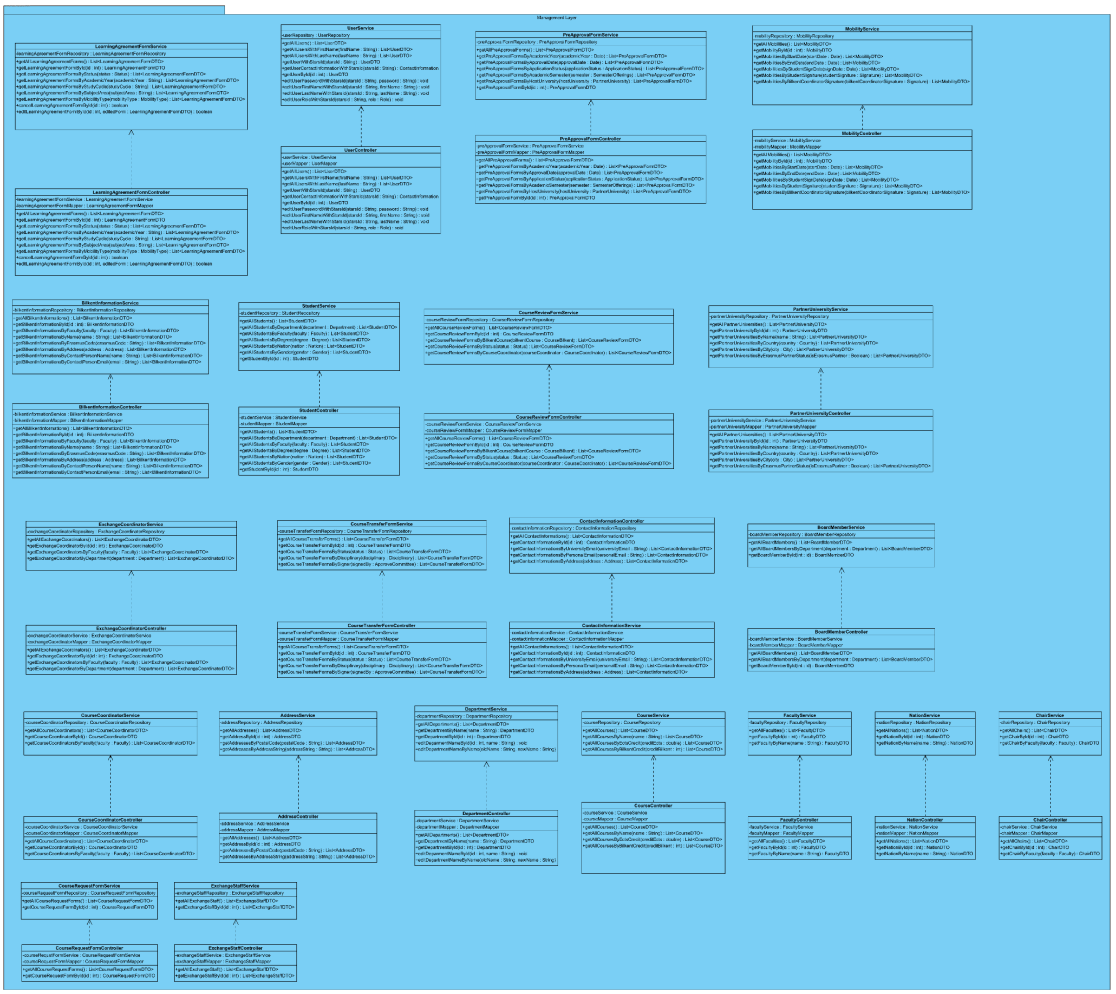

Fig. 4. Management Layer of the System (Click here for full version)

3.2.3 Database Layer

3.2.3.1 Database

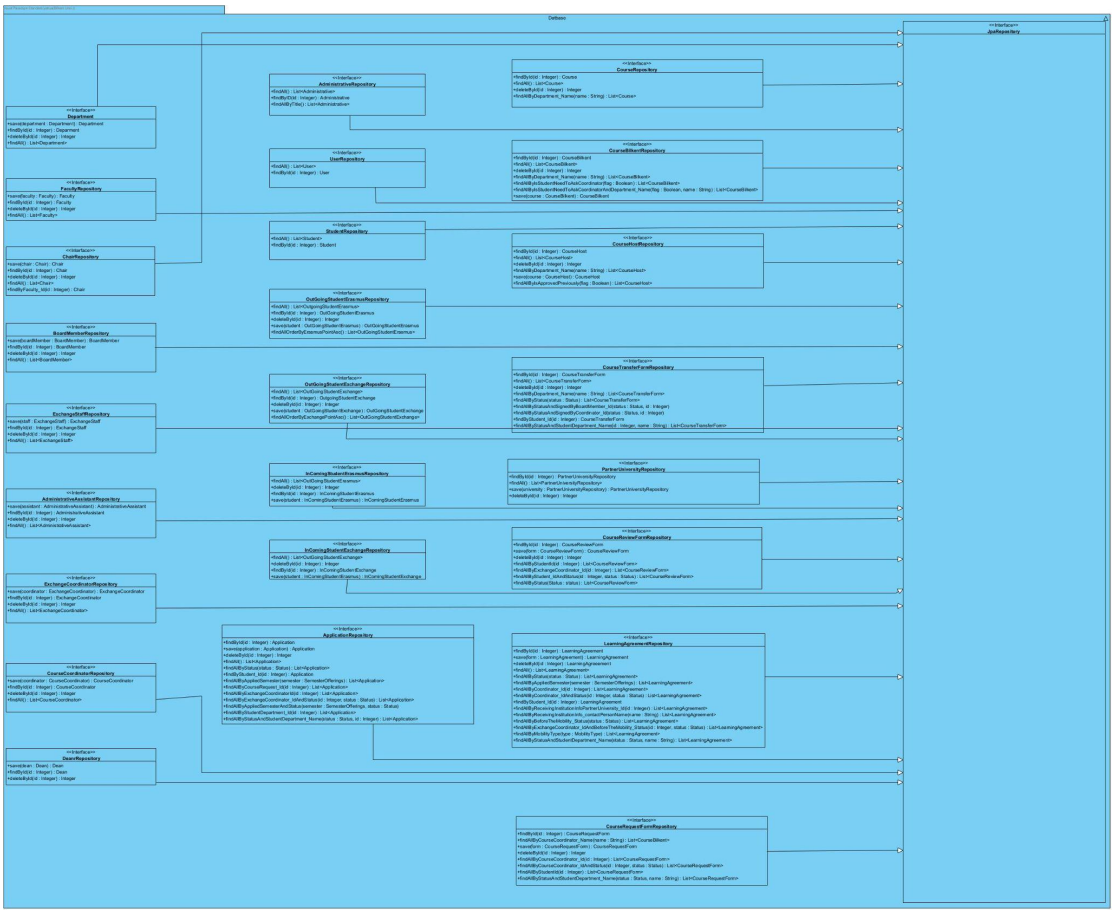

Fig.5. Database Layer of the System (Click here for full version)

3.2.3.2 Entity

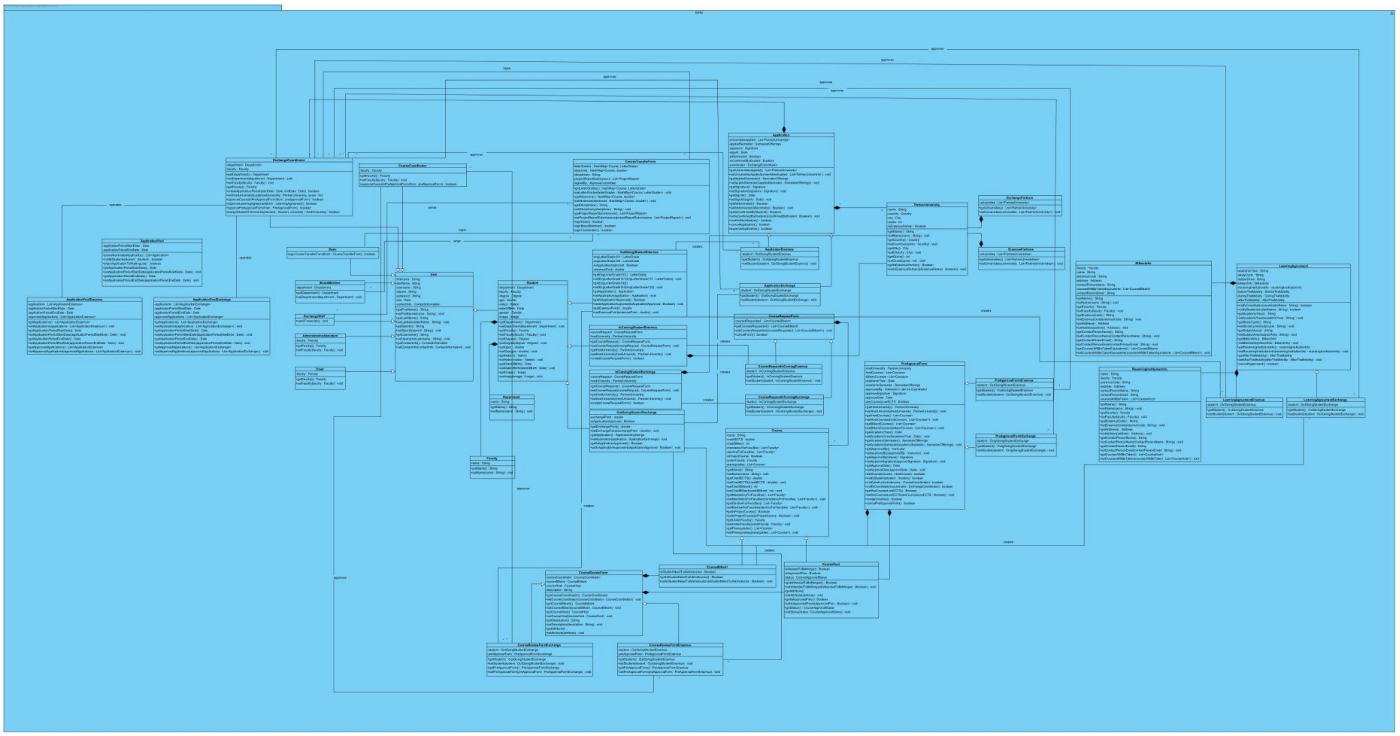

Fig. 6. Entity Diagram of the System (Click here for full version)

3.3 Packages

While implementing Erasmust, it basically uses two different packages. The internal one is to classify the implementation of the internal logic, and the second one is to implement maven dependencies' into Spring Boot.

3.3.1 Internal Packages

In this package, where the java classes in the Spring project are classified, sub-packages such as model, enum, dto, mapper, repository, service, controller packages are kept. The purpose of these packages is to integrate structures such as Entity, Controller and Boundary Objects into the system in a more functional way.

3.3.1.1 Model

These classes, which were initialized to Spring with the @Entity annotation, were created to create database tables with the help of PostgresSQL, integrate the OOP structure into the system and perform ORM operations. Necessary attributes are

assigned to the objects in the system with this class structure. In addition, with Lombok, which is an external package, structures such as getters, setters, constructor were implemented in these classes with clean code logic.

3.3.1.2 Enum

These classes were created in order to prevent the constants used in the system from being created repeatedly, to limit certain attributes, and to implement the authorization algorithm in the external package Spring Security.

3.3.1.3 Dto

The reason for the integration of DTO classes, which stands for Data Transfer Object, into the system is to ensure that only the required fields of a class are transferred between services during data transfer, thus limiting the transferred information. In addition, the controller classes communicate with the front-end and DTO objects, preventing the database from being directly exposed to external manipulation.

3.3.1.4. Mapper

The mapstruct dependency, which is used to convert dtos used for communication between Front-end and Back-end, into required objects easily and quickly, was used in the implementation of mapper interfaces. Mapper interfaces were created to provide developers with ease of writing and to comply with clean-code logic.

3.3.1.5 Controller

Controller classes are the place where service classes are injected into the system with dependency injection, and it is the class where http requests are handled. In addition, the methods in this class take dto objects as input and return types are also dto objects. Thanks to Restful APIs, back-end and front-end communicate over this class.

3.3.1.6 Repository

Thanks to these interfaces with crud operations, sorting and paging algorithms offered by JpaRepository, the system is in communication with the database and the necessary queries are called over these interfaces depending on the situation.

3.3.1.7 Service

Business logic is implemented in these classes where repository interfaces and mapper interfaces are integrated into the system with dependency injection. These classes allow the database to communicate with the controller classes.

3.3.2 External Packages (Dependencies)

External packages are included in the system for ease of integration by installing the dependencies available for use in the Erasmust project.

3.3.2.1 PostgreSQL

This dependency has been integrated into the system to organize the created database using Java, to ensure that the information coming to the system is organized, recorded and reused when necessary.

3.3.2.2 Spring Web

This dependency allows us to create a restful web application with a spring boot project.

3.3.2.3 Spring Data JPA

This dependency performs the ORM operations of the model classes with the annotations it provides and enables the entities to be introduced to Spring.

3.3.2.4 Spring Security

With this dependency, the security of the system is ensured. Spring Security, which uses JWT in its internal implementation, is used to authenticate and authorize users to log in to the system by giving certain roles to users. By integrating Spring Security

into the system, the information of the users is encrypted in a salted way and the data security of these users is kept under guarantee.

3.3.2.5 Lombok

This dependency provides convenience on the developer side. It is used to easily implement functions such as getter, setter, constructor and logger, and the purpose is to comply with the clean-code logic.

3.3.3.6 Mapstruct

This dependency is used to facilitate the conversion of dto objects and entity objects.

3.4 Class Interfaces

3.4.1 Interface Layer Explanation

Splash

First thing users encounter is Splash Page. At this page users select their User Type then they are forwarded to LoginPage.

Operations:

Function selectUserType(): This function receives User Type selected from user and forwards them to LoginPage.

LoginPage

A login page for the program. Then user is forwarded to Dashboard if they are a registered user.

Operations:

Function getId(String id): Receives user's id input.

Function getPassword(String password): Receives user's password input.

Function getMail(String mail): Receives user's mail input.

Function forgotPassword(): Forwards user to ForgotPassword Page.

Function submitUserLoginInformation(): Submits all of the information above for approval.

RegisterPage

Operations:

Function getId(String Id): Receives user's id input. Prompts user if there is anything wrong with input, and asks for them to enter a valid one.

Function getMail(String mail): Receives user's mail input. Prompts user if there is anything wrong with input, and asks for them to enter a valid one.

Function getPassword(String password): Receives user's password input. Prompts user if there is anything wrong with input, and asks for them to enter a valid one.

Function getPhoneNumber(String phoneNumber): Receives user's phone number input. Prompts user if there is anything wrong with input, and asks for them to enter a valid one.

Function getDepartment(Department department): Receives user's department input. Prompts user if there is anything wrong with input, and asks for them to enter a valid one.

Function getFirstName(String firstName): Receives user's first name input.

Function getLastName(String lastName): Receives user's last name input.

Function submitUserRegistrationInformation(): Submits all of the information above for registering user.

ForgotPasswordPage

Operations:

Function getMail(String mail): Gets the user input for mail.

Function submitMail(): Submits the mail for resetting password. Prompts if the mail is not registered to the system.

Function getNewPassword(String password): Gets the user input for password. Prompts user if password is not met with requirements.

Function submitNewPassword(): Submits the mail for resetting password.

Dashboard

Operations:

Function viewAllSubmittedForms(): Shows all of the previous forms submitted by student.

Function showDirectMessages(): Shows direct messages forwarded to the user if any.

AccountPage:

This page contains the user's personal information.

Operations:

Function editPersonalInfo(User setContactInfo): This function changes contact information of the user.

SettingsPage:

This page contains information about notification specifications. Users can change their password from this page as well.

Operations:

Function updatePassword(String password): Changes the password but prompts if the user has entered a password that is not met with requirements.

Function setNotifications(): Changes notification specifications to user's desire.

Navbar:

A navigation bar that is available through all of the pages in the system. Forwards different types of user's to different pages

Operations:

Function goToDashboard(): Forwards user to dashboard. This function is not affected by user type, so all of the user types are forwarded to the dashboard.

Function goToErasmusPage(): Forwards students to StudentErasmusPage, staff to StaffErasmusPage.

Function goToExchangePage(): Forwards students to StudentExchangePage, staff to StaffExchanePage.

Function goToAccountPage(): Forwards user to AccountPage. This function is not affected by user type, so all of the user types are forwarded to AccountPage.

Function goToSettingsPage(): Forwards user to SettingsPage. This function is not affected by user type, so all of the user types are forwarded to SettingsPage.

Function goToApplicationsPage(): Forwards students to StudentApplicationsPage, staff to StaffApplicationsPage.

Function viewSubmissions(): When clicked on the icon, students can see their submission in a pop-up status of their forms like, Preapproval, Learning Agreement etc.

Function logOut(): Logs out the user and forwards them to Splash.

StudentApplicationsPage

Operations:

Function viewAllSubmittedForms(): Shows students all of their previous Applications and their approval.

StaffCourseTransferPage

Operations:

Function createCourseTransferForm(Student student, CourseTransferForm ctForm, boolean isAuthorized): Enables staff to create a Course Transfer Form for the specifically selected student, if the staff has the proper authorization for it.

Function signCourseTransferForm(CourseTransferForm ctForm, boolean isAuthorized): Enables staff to sign the previously created Course Transfer Form of the previously selected student, if the staff has the proper authorization of it.

StaffApplicationsPage

Only staff can see this page but not all of the types of staff can do the operations listed below.

Operations:

Function viewStudentApplicationDetails(Student selectedStudent): Shows selectedStudent's application details.

Function goToCourseTransferForm(): Forwards staff to courseTransferForm Page.

StudentExchangePage

Operations:

Function goToExchangeApplication(): Forwards student to StudentExchangeApplication page.

Function goToExchangePreapprovalApplication(): Forwards student to ExchangePreapprovalApplication page.

Function goToStudentExchangeLearningAgreement(): Forwards student to ExchangeLearningAgreement page.

StudentExchangeApplication

Operations:

Function createExchangeApplication(Student student, ApplicationForm studentForm) : This function can create an Exchange Application with student's input to attributes that exist in Application Form.

Function viewExchangeApplication(ApplicationExchange studentForm) : This function can show the student the existing Exchange application.

Function editExchangeApplication(ApplicationExchange studentForm) : This function enables the student to edit the existing Exchange application.

Function cancelExchangeApplication(ApplicationExchange studentForm) : This function discards the existing Exchange Application that user edits.

StudentExchangePreapprovalApplication

Operations:

Function createExchangePreapprovalApplication(Student student, PreapprovalForm studentForm) : This function can create an Exchange Preapproval Application with student's input to attributes that exist in Preapproval Form.

Function viewExchangePreapprovalApplication(PreapprovalFormExchange studentForm) : This function can show the student the existing Exchange Preapproval Application.

Function cancelExchangePreapprovalApplication(PreapprovalFormExchange studentForm) : This function discards the existing Exchange Preapproval Application that user edits.

Function updateExchangePreapprovalApplication(PreapprovalFormExchange studentForm) : This function enables the student to update the existing Exchange Preapproval Application.

StudentExchangeLearningAgreement

Operations:

Function createExchangeLearningAgreement(Student student, LearningAgreementForm studentForm) : This function can create an Exchange Learning Agreement Form with student's input to attributes that exist in LearningAgreement Form.

Function viewExchangeLearningAgreement(LearningAgreementFormExchange studentForm) : This function can show the student the existing Exchange Learning Agreement Form.

Function editExchangeLearningAgreement(LearningAgreementFormExchange studentForm) : This function enables the student to edit the existing Exchange Learning Agreement Form.

StudentErasmusPage

Operations:

Function goToErasmusApplication(): Forwards student to StudentErasmusApplication page.

Function goToErasmusPreapprovalApplication(): Forwards student to ErasmusPreapprovalApplication page.

Function goToStudentErasmusLearningAgreement(): Forwards student to ErasmusLearningAgreement page.

StudentErasmusApplication

Operations:

Function createErasmusApplication(Student student, ApplicationForm studentForm) : This function can create an Erasmus Application with student's input to attributes that exist in Application Form.

Function viewErasmusApplication(ApplicationErasmus newApplication) : This function can show the student the existing Erasmus application.

Function editErasmusApplication(ApplicationErasmus newApplication) : This function enables the student to edit the existing Erasmus application.

Function cancelErasmusApplication(ApplicationErasmus newApplication) : This function discards the existing Erasmus Application that user edits.

StudentErasmusPreapprovalApplication

Operations:

Function createErasmusPreapprovalApplication(Student student, PreapprovalForm studentForm) : This function can create an Erasmus Preapproval Application with student's input to attributes that exist in Preapproval Form.

Function viewErasmusPreapprovalApplication(PreapprovalFormErasmus studentForm) : This function can show the student the existing Erasmus Preapproval Application.

Function cancelErasmusPreapprovalApplication(PreapprovalFormErasmus studentForm) : This function discards the existing Erasmus Preapproval Application that user edits.

Function updateErasmusPreapprovalApplication(PreapprovalFormErasmus studentForm) : This function enables the student to update the existing Erasmus Preapproval Application.

StudentErasmusLearningAgreement

Operations:

Function createErasmusLearningAgreement(Student student, LearningAgreementForm studentForm) : This function can create an Erasmus Learning Agreement Form with student's input to attributes that exist in LearningAgreement Form.

Function viewErasmusLearningAgreement(LearningAgreementFormErasmus studentForm) : This function can show the student the existing Erasmus Learning Agreement Form.

Function editErasmusLearningAgreement(LearningAgreementFormErasmus studentForm) : This function enables the student to edit the existing Erasmus Learning Agreement Form.

StaffExchangePage

Operations:

Function createExchangeUniversityListfForWaitingList(boolean isAuthorized) : This function is used by Exchange Coordinator, and creates a list of universities for the students that are in waiting list.

Function createNewExchangePeriod(Date startDate, Date endDate, boolean isAuthorized) : Used at the beginning of the term by Exchange Coordinator, this function sets the Exchange Period.

Function goToExchangeSchool(): Forwards staff to StaffExchangeSchool page.

Function goToExchangeStudentOperations(): Forwards staff to StaffExchangeStudentOperations page.

StaffExchangeScool

Operations:

Function addNewExchangeSchool(PartnerUniversity newSchool, boolean isAuthorized) : This function adds a new Exchange Partner to PartnerUniversityList by staff specified attributes to PartnerUniversity.

Function removeExchangeSchool(PartnerUniversity school, boolean isAuthorized): This function enables staff to remove the existing Exchange Partner from Partner University List.

StaffExchangeStudentOperations

Operations:

Function reviewExchangePreapprovalApplication(Student selectedStudent, boolean isAuthorized) : This function shows the staff Exchange Preapproval Application of the specified student.

Function assignExchangeStudentToASchool(Student selectedStudent, boolean isAuthorized, PartnerUniversity school) : Used by Exchange Coordinator, this function assigns the specified student to the specified school.

Function viewExchangeProgramDetails(Student selectedStudent): This function shows the specified student's program details to staff members.

StaffErasmusPage

Operations:

Function createErasmusUniversityListForWaitingList(boolean isAuthorized) : This function is used by Erasmus Coordinator, and creates a list of universities for the students that are in waiting list.

Function createNewErasmusPeriod(Date startDate, Date endDate, boolean isAuthorized) : Used at the beginning of the term by Erasmus Coordinator, this function sets the Erasmus Period.

Function goToErasmusSchool(): This function forwards staff to StaffErasmusScool page.

Function goToErasmusStudentOperations(): This function forwards staff to StaffErasmusStudentOperations page.

StaffErasmusStudentOperations

Operations:

Function reviewErasmusPreapprovalApplication(Student selectedStudent, boolean isAuthorized) : This function shows the staff Erasmus Preapproval Application of the specified student.

Function assignErasmusStudentToASchool(Student selectedStudent, boolean isAuthorized, PartnerUniversity school) : Used by Erasmus Coordinator, this function assigns the specified student to the specified school.

Function viewErasmusProgramDetails(Student selectedStudent) : This function shows the specified student's program details to staff members.

StaffErasmusSchool

Operations:

Function addNewErasmusSchool(PartnerUniversity newSchool, boolean isAuthorized) : This function adds a new Erasmus Partner to PartnerUniversityList by staff specified attributes to PartnerUniversity.

Function removeErasmusSchool(PartnerUniversity school, boolean isAuthorized) : This function enables staff to remove the existing Erasmus Partner from Partner University List.

AdminDashBoard

Operations:

Function showDirectMessages(): Admin can view all of the communications using this function.

Function showUserInfo(): Admin can view all of the user types and their corresponding registered number of users.

AdminNavbar

Operations:

Function goToAdminDashboard(): Forwards admin to AdminDashboard page.

Function goToStudents(): Forwards admin to AdminStudents page.

Function goToApplication(): Forwards admin to AdminApplications page.

Function goToStaff(): Forwards admin to AdminStaff page.

Function goToPreapproval(): Forwards admin to AdminPreapproval page.

AdminStaff:

Operations:

Function createStaff(User user, Title title): Enables admin to create a staff object with the specified information from parameters.

Function viewStaff(Staff staff): Enables admin to view staff information of the specified staff.

Function updateStaff(Staff staff): Enables admin to edit Staff information.

Function deleteStaff(Staff staff): Enables admin to delete specified staff.

Function goToAdminStaffList(): Forwards admin to AdminStaffList page.

AdminStaffList:

Operations:

Function viewStaffList(): Enables admin to view all of the staff information of all of the existing staff in the app.

AdminStudents:

Operations:

Function createStudent(Department department, Faculty faculty, Degree degree, double GPA, Nation nation, Date birth, Gender gender, Image profile picture): Enables admin to create a Student object with the specified information from parameter list.

Function viewStudent(Student selectedStudent): Enables admin to view the specified Student.

Function updateStudent(Student selectedStudent): Enables admin to edit student object.

Function deleteStudent(Student selectedStudent): Enables admin to delete the specified student from database.

Function goToAdminStudentList(): Forwards admin to AdminStudentList page.

AdminStudentList:

Operations:

Function viewStudentList(): Enables admin to view all of the students in the database.

AdminPreapproval:

Operations:

Function createPreapproval(Department department, Faculty faculty, Degree degree, double GPA, Nation nation, Date birth, Gender gender, Image profile picture): Enables admin to create Preapproval with the specified information from parameter list.

Function viewPreapproval(PreapprovalForm preapprovalForm): Enables admin to view the specified Preapproval.

Function updatePreapproval(PreapprovalForm preapprovalForm): Enables admin to update specified Preapproval.

Function deletePreapproval(PreapprovalForm preapprovalForm): Enables admin to delete the specified Preapproval from database.

Function goToAdminPreapprovalList(): Forwards admin to AdminPreapprovalList page.

AdminPreapprovalList:

Operations:

Function viewPreapprovalList(): Enables admin to view all of the Preapproval Forms in the database.

AdminApplication:

Operations:

Function createApplication(List appliedUniversities, SemesterOfferings appliedSemester, Signature signature, Date signedAt, Coordinator coordinator, Status status, List courseReviewForms): Enables admin to create an application with the specified information.

Function viewApplication(ApplicationForm application): Enables admin to view details of the specified application.

Function updateApplication(ApplicationForm application): Enables admin to edit specified application.

Function deleteApplication(ApplicationForm application): Enables admin to delete the specified application from database.

Function goToAdminApplicationList(): Forwards admin to AdminApplicationList.

AdminApplicationList:

Operations:

Function viewApplicationList(): Enables admin to view all of the applications in the database.

3.4.2 Management Layer Explanation

In the management layer Erastmust has 2 main components: services and controllers. Service objects can access data by using repositories and main logic is implemented in these objects. Controller objects on the other hand, are used to create endpoints by using service objects. Requests like GET, POST, PATCH, PUT are created. Here controller objects are boundary objects and service objects are control objects. This design prevents actors from communicating directly with the entity objects. It also keeps the implementation of logic and data transfer object to entity/ entity to data transfer object conversions in service objects, which cannot be accessed by the actor.

In general we have the following methods for most of the services and controllers:

- getAll() : List<ObjectDTO> : returns all the objects DTO's in a list. This is useful to access all objects of a class
- getById(int id): ObjectDTO: returns DTO of the object with the given id, since id's for every class is separate, this method either returns 1 DTO or null.

Other than these common methods, classes also have special methods for getting and editing objects.

3.4.3 Database Layer Explanation

The implementation of the database was made using Hibernate, which was made available by Java. Thanks to this open-source framework, ORM (Object Relational Mapping) operations, the project's database was built with the functions offered by JPA (Java Persistence API). With HQL (Hibernate Query Language), the object-oriented version of SQL, Hibernate provides convenience on the developer side and offers high performance when searching data. For these reasons, it was decided to use JPA and Hibernate to implement the project. Moreover, classes that extend the JpaRepository are conventionally named Repository. For easy understanding of Data Integrity and classification of similar types of objects, the repositories are divided into linked packages.

Due to the simplicity of the report, not all methods are explained in detail. You can understand the purpose of some keys below.

3.4.3.1 UserRepositories Package

Database interfaces of all user types in the system are kept in this package.

3.4.3.1.1 UserRepository Interface

This interface was created to create the necessary queries for the User object in which the basic information of any user in the Erasmust system is inherited.

- **User findById(Integer id):** This function returns a user with the given id attribute as an input,if there is no match with the given id in the database it returns null value.

- **List<User> findAll():** This function returns a list of all users available in the database, if there is no user in the system it returns null value.

3.4.3.1.2 AdministrativeRepositories Package

Database interfaces of all administrative user types in the system are kept in this package.

3.4.3.1.2.1 AdministrativeRepository Interface

This interface was created to create the necessary queries for the Administrative object, which is extended from the User object, where the basic information of any administrative user in the Erasmust system is kept.

- **List<Administrative> findAll():** This function returns a list of all administrative users available in the database, if there is no administrative user in the system it returns null value.

- **Administrative findById(Integer id):** This function returns an administrative user with the given id attribute as an input, if there is no match with the given id in the database it returns null value.

- **List<Administrative> findAllByTitle(String name):** This function returns a list of all the administrative users with the given name attribute as an input, if there is no available administrative user in the database it returns null value.

3.4.3.1.2.2 AdministrativeAssistantRepository Interface

This interface was created to create the necessary queries for the AdministrativeAssistant object, which is extended from the Administrative object, where the administrative assistant user information in the Erasmust system is kept.

- **AdministrativeAssistant save(AdministrativeAssistant assistant):** This function saves newly registered administrative assistant to the database and returns saved object.

- **AdministrativeAssistant findById(Integer id):** This function returns an administrative assistant with the given id attribute as an input, if there is no match with the given id in the database it returns null value.

- **Integer deleteById(Integer id):** This function removes the administrative assistant from the database with the given id attribute and it returns deleted object's unique id, if there is no match with the given id it returns -1.

- **List<AdministrativeAssistant> findAll():** This function returns a list of all administrative assistants available in the database, if there is no administrative assistant in the system it returns null value.

3.4.3.1.2.3 BoardMemberRepository Interface

This interface was created to create the necessary queries for the AdministrativeAssistant object, which is extended from the Administrative object, where the board member's information in the Erasmust system is kept.

- **BoardMember save(BoardMember boardMember):** This function saves newly registered board member to the database and returns saved object.

- **BoardMember findById(Integer id):** This function returns a board member with the given id attribute as an input, if there is no match with the given id in the database it returns null value.

- **Integer deleteById(Integer id):** This function removes the board member from the database with the given id attribute and it returns deleted object's unique id, if there is no match with the given id it returns -1.

- **List<BoardMember> findAll():** This function returns a list of all board members available in the database, if there is no board member in the system it returns null value.

3.4.3.1.2.4 ChairRepository Interface

This interface was created to create the necessary queries for the Chair object, which is extended from the Administrative object, where the chair's information in the Erasmust system is kept.

- **Chair save(Chair chair):** This function saves newly registered a chair to the database and returns saved object.

- **Chair findById(Integer id):** This function returns chair with the given id attribute as an input, if there is no match with the given id in the database it returns null value.

- **Integer deleteById(Integer id):** This function removes the chair from the database with the given id attribute and it returns deleted object's unique id, if there is no match with the given id it returns -1.

- **List<Chair> findAll():** This function returns a list of all chairs available in the database, if there is no chair in the system it returns null value.

- **Chair findByFaculty_Id(Integer id):** This function returns chair with the given id attribute as an input by looking for chair's faculty, if there is no match with the given faculty id in the database it returns null value.

3.4.3.1.2.5 CourseCoordinatorRepository Interface

This interface was created to create the necessary queries for the CourseCoordinator object, which is extended from the Administrative object, where the course coordinator's information in the Erasmust system is kept.

- **CourseCoordinator save(CourseCoordinator coordinator):** This function saves newly registered course coordinator to the database and returns saved object.

- **CourseCoordinator findById(Integer id):** This function returns a course coordinator with the given id attribute as an input, if there is no match with the given id in the database it returns null value.

- **Integer deleteById(Integer id):** This function removes the course coordinator from the database with the given id attribute and it returns deleted object's unique id, if there is no match with the given id it returns -1.

- **List<CourseCoordinator> findAll():** This function returns a list of all course coordinators available in the database, if there is no course coordinator in the system it returns null value.

3.4.3.1.2.6 DeanRepository Interface

This interface was created to create the necessary queries for the Dean object, which is extended from the Administrative object, where the dean's information in the Erasmust system is kept.

- **Dean save(Dean dean):** This function saves newly registered a dean to the database and returns saved object.

- **Dean findById(Integer id):** This function returns a dean with the given id attribute as an input, if there is no match with the given id in the database it returns null value.

- **Integer deleteById(Integer id):** This function removes the dean from the database with the given id attribute and it returns deleted object's unique id, if there is no match with the given id it returns -1.

3.4.3.1.2.7 ExchangeStaffRepository Interface

This interface was created to create the necessary queries for the ExchangeStaff object, which is extended from the Administrative object, where the exchange staff's information in the Erasmust system is kept.

- **ExchangeStaff save(ExchangeStaff staff):** This function saves newly registered exchange staff to the database and returns saved object.

- **ExchangeStaff findById(Integer id):** This function returns an exchange staff with the given id attribute as an input, if there is no match with the given id in the database it returns null value.

- **Integer deleteById(Integer id):** This function removes the exchange staff from the database with the given id attribute and it returns deleted object's unique id, if there is no match with the given id it returns -1.

- **List<ExchangeStaff> findAll():** This function returns a list of all exchange staffs available in the database, if there is no exchange staff in the system it returns null value.

3.4.3.1.2.8 ExchangeCoordinatorRepository Interface

This interface was created to create the necessary queries for the ExchangeCoordinator object, which is extended from the Administrative object, where the exchange coordinator's information in the Erasmust system is kept.

- **ExchangeCoordinator save(ExchangeCoordinator coordinator):** This function saves newly registered exchange coordinator to the database and returns saved object.

- **ExchangeCoordinator findById(Integer id):** This function returns an exchange coordinator with the given id attribute as an input, if there is no match with the given id in the database it returns null value.

- **Integer deleteById(Integer id):** This function removes the exchange coordinator from the database with the given id attribute and it returns deleted object's unique id, if there is no match with the given id it returns -1.

- **List<ExchangeCoordinator> findAll():** This function returns a list of all exchange coordinators available in the database, if there is no exchange coordinator in the system it returns null value.

3.4.3.1.3 StudentRepositiories Package

Database interfaces of all student user types in the system are kept in this package.

3.4.3.1.3.1 StudentRepository Interface

This interface was created to create the necessary queries for the Student object, which is extended from the User object, where the basic information of any student in the Erasmust system is kept.

- **List<Student> findAll():** This function returns a list of all students available in the database, if there is no student in the system it returns null value.

- **Student findById(Integer id):** This function returns a student with the given id attribute as an input, if there is no match with the given id in the database it returns null value.

3.4.3.1.3.2 OutGoingStudentExchangeRepository Interface

This interface was created to create the necessary queries for the OutGoingStudentExchange object, which is extended from the Student object, where the information of the students who have applied to the exchange program in the Erasmust system is kept.

- **List<OutGoingStudentExchange> findAll():** This function returns a list of all students accepted to the exchange program in the database, returns a null value if there are no students in the system.

- **OutgoingStudentExchangefindById(Integer id):** This function returns a student accepted to the exchange program with the given id attribute as an input, if there is no match with the given id in the database it returns null value.

- **Integer deleteById(Integer id):** This function removes the student accepted to the exchange program from the database with the given id attribute and it returns deleted object's unique id, if there is no match with the given id it returns -1.

- **OutGoingStudentExchange save(OutGoingStudentExchange student):** This function saves newly registered students accepted to the exchange program to the database and returns saved object.

- **List<OutGoingStudentExchange> findAllOrderByExchangePointAsc():** This function returns a descending list of all students accepted to the exchange program in the database by exchange points of students, returns a null value if there are no students in the system.

3.4.3.1.3.3 OutGoingStudentErasmusRepository Interface

This interface was created to create the necessary queries for the OutGoingStudentErasmus object, which is extended from the Student object, where the information of the students who have applied to the Erasmus program in the Erasmust system is kept.

- **List<OutgoingStudentErasmus> findAll():** This function returns a list of all students accepted to the erasmus program in the database, returns a null value if there are no students in the system.

- **OutGoingStudentErasmus findById(Integer id):** This function returns a student accepted to the exchange program with the given id attribute as an input, if there is no match with the given id in the database it returns null value.

- **Integer deleteById(Integer id):** This function removes the student accepted to the erasmus program from the database with the given id attribute and it returns deleted object's unique id, if there is no match with the given id it returns -1.

- **OutGoingStudentErasmus save(OutGoingStudentErasmus student):** This function saves newly registered students accepted to the erasmus program to the database and returns saved object.

- **List<OutGoingStudentErasmus> findAllOrderByErasmusPointAsc():** This function returns a descending list of all students accepted to the erasmus program in the database by erasmus points of students, returns a null value if there are no students in the system.

3.4.3.1.3.4 InComingStudentExchangeRepository Interface

This interface was created to create the necessary queries for the InComingStudentExchange object, which is extended from the Student object, where the information of the students who came to Bilkent with the exchange program in the Erasmust system is kept.

- **List<InGoingStudentExchange> findAll():** This function returns a list of all students who are coming to the Bilkent with exchange program in the database, returns a null value if there are no students in the system.

- **Integer deleteById(Integer id):** This function removes the student who is coming to the Bilkent with exchange program from the database with the

given id attribute and it returns deleted object's unique id, if there is no match with the given id it returns -1.

- **InComingStudentExchange findById(Integer id):** This function returns a student who is coming to the Bilkent with exchange program with the given id attribute as an input, if there is no match with the given id in the database it returns null value.

- **InComingStudentExchange save(InComingStudentErasmus student):** This function saves newly registered students who are coming to the Bilkent with exchange program to the database and returns saved object.

3.4.3.1.3.5 InComingStudentErasmusRepository Interface

This interface was created to create the necessary queries for the InComingStudentErasmus object, which is extended from the Student object, where the information of the students who came to Bilkent with the Erasmus program in the Erasmust system is kept.

- **List<OutGoingStudentErasmus> findAll():**
- **Integer deleteById(Integer id):**
- **InComingStudentErasmus findById(Integer id):**
- **InComingStudentErasmus save(InComingStudentErasmus student):**

3.4.3.2 PaperWorkRepositories

Database interfaces of all paperwork documents in the system are kept in this package.

3.4.3.2.1 ApplicationRepository Interface

This interface was created to create the necessary queries for the Application object, where the information of the applicants and their application forms in the Erasmust system is kept.

- **Application findById(Integer id):**
- **Application save(Application application):**
- **Integer deleteById(Integer id):**
- **List<Application> findAll():**
- **List<Application> findAllByStatus(Status status):**
- **Application findByStudent_Id(Integer id):**

- **List<Application> findAllByAppliedSemester(SemesterOfferings semester):**
- **List<Application> findAllByCourseRequest_Id(Integer id):**
- **List<Application> findAllByExchangeCoordinatorId(Integer id):**
- **List<Application> findAllByExchangeCoordinator_IdAndStatus(Integer id, Status status):**
- **List<Application> findAllByAppliedSemesterAndStatus(SemesterOfferings semester, Status status):**
- **List<Application> findAllByStudentDepartment_Id(Integer id):**
- **List<Application> findAllByStatusAndStudentDepartment_Name(Status status, Integer id):**

3.4.3.2.2 PreApprovalFormRepository Interface

This interface was created to create the necessary queries for the PreApprovalForm object, where the information of those applicants who are officially nominated and approved by the exchange coordinator and their application forms in the Erasmust system is kept.

- **PreApprovalForm findBy(Integer id):**
- **List<Pre ApprovalForm> findAll():**
- **PreApprovalForm save(PreApprovalForm form):**
- **Integer deleteByld(Integer id):**
- **PreApprovalForm findByStudent_Id(Integer id):**
- **List<PreApproval Form> findAllByStatus(Status status):**
- **List<PreApprovalForm> findAllByAppliedSemester (SemesterOfferings semester):**
- **PreApprovalForm findByStudent_IdAndCourseRequest_Status(Integer id, Status status):**
- **List<PreApprovalForm> findAllByExchangeCoordinator_Id(Integer id):**
- **List<Pre ApprovalForm> findAllByExchangeCoordinator_IdAndStatus (Integer id, Status status):**
- **List<PreApprovalForm> findAllByApplied Semester And Status (Semester Offerings semester, Status status):**
- **List<PreApproval Form> findAllByStatusAndStudent Department_Name(Status status, String name):**

3.4.3.2.3 CourseReviewFormRepository Interface

This interface was created to create the necessary queries for the CourseReviewForm object, where the information of the desired host university courses created for the course selection section in the pre-approval form in the Erasmust system is kept.

- **CourseReviewForm findById(Integer id):**
- **CourseReviewForm save(CourseReviewForm form):**
- **Integer deleteById(Integer id):**
- **List<CourseReviewForm> findAllByStudentId(Integer id):**
- **List<CourseReviewForm> findAllByExchangeCoordinator_Id(Integer id):**
- **List<CourseReviewForm> findAllByStudent_IdAndStatus(Integer id, Status status):**
- **List<CourseReviewForm> findAllByStatus(status Status):**

3.4.3.2.4 CourseRequestFormRepository Interface

This interface was created to create the necessary queries for the CourseRequestForm object, where the information about the course choices of students who come to Bilkent with the Erasmus or exchange program in the Erasmust system is kept

- **CourseRequestForm findById(Integer id):**
- **List<CourseBilkent> findAllByCourseCoordinator_Name(String name):**
- **CourseRequestForm save(CourseRequestForm form):**
- **Integer deleteById(Integer id):**
- **List<CourseRequestForm> findAllByCourseCoordinator_Id(Integer id):**
- **List<CourseRequestForm> findAllByCourseCoordinator_IdAndStatus(Integer id, Status status):**
- **List<CourseRequestForm> findAllByStudentId(Integer id):**
- **List<CourseRequestForm> findAllByStatusAndStudentDepartment_Name(Status status, String name):**

3.4.3.2.5 CourseTransferFormRepository Interface

This interface was created to create the necessary queries for the CourseTransferForm object, where the information about the the transcripts of the students who return to Bilkent from the host University after completing the Erasmus or exchange program in the Erasmust system is kept

- **CourseTransferForm findById(Integer id):**

- **List<CourseTransferForm> findAll():**
- **Integer deleteById(Integer id):**
- **List<CourseTransferForm> findAllByDepartment_Name(String name):**
- **List<CourseTransferForm> findAllByStatus(Status status):**
- **List<CourseTransferForm> findAllByStatusAndSignedByBoardMember_Id(Status status, Integer id):**
- **List<CourseTransferForm> findAllByStatusAndSignedByCoordinator_Id(Status status, Integer id):**
- **CourseTransferForm findByStudent_Id(Integer id):**
- **List<CourseTransferForm> findAllByStatusAndStudentDepartment_Name(Integer id, String name):**

3.4.3.2.6 LearningAgreementRepository Interface

This interface was created to create the necessary queries for the LearningAgreement object, where the learning agreement forms of those applicants whose preapproval forms are officially approved by exchange coordinator in the Erasmust system is kept.

- **LearningAgreement findById(Integer id):**
- **LearningAgreement save(LearningAgreement form):**
- **LearningAgreeement deleteById(Integer id):**
- **List<LearningAgreement> findAll():**
- **List<LearningAgreement> findAllByStatus(Status status):**
- **List<LearningAgreement> findAllByAppliedSemester(SemesterOfferings semester):**
- **List<LearningAgreement> findAllByCoordinator_Id(Integer id):**
- **List<LearningAgreement> findAllByCoordinator_IdAndStatus(Integer id, Status status):**
- **LearningAgreement findByStudent_Id(Integer id):**
- **List<LearningAgreement> findAllByReceivingInstitutionInfoPartnerUniversity_Id(Integer id):**
- **List<LearningAgreement> findAllByReceivingInstitutionInfo_contactPersonName(String name):**
- **List<LearningAgreement> findAllByBeforeTheMobility_Status(Status status):**
- **List<LearningAgreement> findAllByExchangeCoordinator_IdAndBeforeTheMobility_Status(Integer id, Status status):**
- **List<LearningAgreement> findAllByMobilityType(MobilityType type):**

- **List<LearningAgreement> findAllByStatusAndStudentDepartment_Name(Status status, String name):**

3.4.3.3 OfferingsRepositories Package

Database interfaces of courses and partner universities in the system are kept in this package.

3.4.3.3.1 CourseRepository Interface

This interface was created to create the necessary queries for the Course object where the basic information of offered courses in Bilkent and some of the courses offered by the partner universities in the Erasmust system is kept.

- **Course findById(Integer id):**
- **List<Course> findAll():**
- **Integer deleteById(Integer id):**
- **List<Course> findAllByDepartment_Name(String name):**

3.4.3.3.2 CourseBilkentRepository Interface

This interface was created to create the necessary queries for the CourseBilkent object, which is extended from the Course object, where the information offered courses in Bilkent in the Erasmust system is kept.

- **CourseBilkent findById(Integer id):**
- **List<CourseBilkent> findAll():**
- **Integer deleteById(Integer id):**
- **List<CourseBilkent>findAllByDepartment_Name(String name):**
- **List<CourseBilkent> findAllByIsStudentNeedToAskCoordinator(Boolean flag):**
- **List<CourseBilkent> findAllByIsStudentNeedToAskCoordinatorAndDepartment_Name(Boolean flag, String name):**
- **CourseBilkent save(CourseBilkent course):**

3.4.3.3.3 CourseHostRepositiory Interface

This interface was created to create the necessary queries for the CourseHost object, which is extended from the Course object, where the information about some of the courses offered by the partner universities the Erasmust system is kept. These courses saved in the database are previously approved courses. Moreover, objects created for the courses that students are waiting for approval in the approval form and learning agreement are also kept here.

- **CourseHost findById(Integer id):**
- **List<CourseHost> findAll():**
- **Integer deleteById(Integer id):**
- **List<CourseHost> findAllByDepartment_Name(String name):**
- **CourseHost save(CourseHost course):**
- **List<CourseHost> findAllByIsApprovedPreviously(Boolean flag):**

3.4.3.3.4 PartnerUniversityRepository Interface

This interface was created to create the necessary queries for the PartnerUniversity object where theinformation about the universities that Bilkent has contracted with for the exchange and Erasmus program in the Erasmust system is kept.

- **PartnerUniversityRepository findById(Integer id):**
- **List<PartnerUniversityRepository> findAll():**
- **PartnerUniversityRepository save(PartnerUniversityRepository university):**
- **Integer deleteById(Integer id)**

4 Improvement Summary

- Added and modified missing interface layer pages and their corresponding commentaries.
- Added aggregation relations to missing parts.
- Changed some descriptions according to the feedback.
- Formatting of the file was fixed.

5 References

[1] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.

[2]“Erasmus+ online başvuru ve Yönetim Sistemi - Değişim Ofisi - Bilkent üniversitesi,” *Bilkent Üniversitesi | Erasmus+ Başvuru Sistemi*. [Online]. Available: https://app.erasmus.bilkent.edu.tr/tr/. [Accessed: 06-Nov-2022].

[3]“Uluslararası Değişim programları uygulama esasları,” *TR / Bilkent Universitesi –*. [Online]. Available: https://w3.bilkent.edu.tr/www/uluslararasi-degisim-programlari-uygulama-esaslari/. [Accessed: 06-Nov-2022].

[4] H. A. Güvenir, “Student Exchange FAQ,” *Information for Bilkent CS Exchange Students*. [Online]. Available: http://cs.bilkent.edu.tr/~exchange/exchange_faq.html. [Accessed: 06-Nov-2022].

[5] “Bilkent University - exchange programs,” *Bilkent University - Exchange Programs*. [Online]. Available: http://www.exchange.bilkent.edu.tr/. [Accessed: 06-Nov-2022].

[6] “Exchange Procedure,” *Erasmus/Exchange procedure*. [Online]. Available: http://cs.bilkent.edu.tr/~exchange/procedure.html. [Accessed: 06-Nov-2022].

[7] C. Alkan, *Term Project Requirements Conference*, 11-Oct-2022. Bilkent University.

[8] “Spring makes java simple.,” *Spring*. [Online]. Available: https://spring.io/. [Accessed: 06-Nov-2022].

[9] “PostgreSQL: The World's Most Advanced Open Source Relational Database”.https://www.postgresql.org/. [Accessed: 06-Nov-2022].

[10] “React – a JavaScript library for building user interfaces,” – *A JavaScript library for building user interfaces*. [Online]. Available: https://reactjs.org/. [Accessed: 06-Nov-2022].