Bilkent University

Department of Computer Engineering

CS319 Term Project

ErasXchange

Group 3B

Design Report

Group Members:

Ceren Akyar - 22003158

Deniz Mert Dilaverler - 22003530

Hasan Yarkın Kurt - 22003072

Ali Emir Güzey - 22003186

Dağhan Ünal - 22002182

Instructor: Eray Tüzün

Teaching Assistant(s): Emre Sülün, Muhammed Umair Ahmed, Mert Kara, İdil Hanhan

Date: November 29, 2022

Contents

1. Introduction

1.1 Purpose of the System

ErasXchange is a web-based application for managing Bilkent University students' Erasmus and Exchange application process. Its primary goal is to provide a common platform for coordinators, board members, International Students Office, and students in which they can all fulfill their duties easily. The app supports all the necessary functionalities during the Erasmus and Exchange process in an easily trackable manner which in turn, eases the workload on all parties. As a consequence, ErasXchange reduces unnecessary mail and paper usage significantly which is an ongoing problem in the current process. ErasXchange can be used for making student placements, filling and sending Pre-approval and Course Transfer forms, and rejecting and approving these forms.

1.2 Design Goals

1.2.1 Functionality

ErasXchange will replace the current Erasmus/Exchange process which is manual. In order to successfully do this app should support the necessary functionalities. App will be used by a variety of different users. These users are students, coordinators, international student office, and board members. As a result, ErasXchange should support different functionalities for different users.

1.2.2 Maintainability and Extendibility

ErasXchange will be used by Bilkent University for many years to come which makes maintainability a crucial factor. Hence during development, the codebase will be well-documented to ensure easier debugging and testing. It should also be considered that there can be changes to Erasmus/Exchange procedure which may require changes in software. Therefore, ErasXchange will be developed in accordance with Object Oriented design patterns to make sure that new features can be added easily with minimal debugging and testing required.

2. High-level software architecture

2.1 Subsystem Decomposition

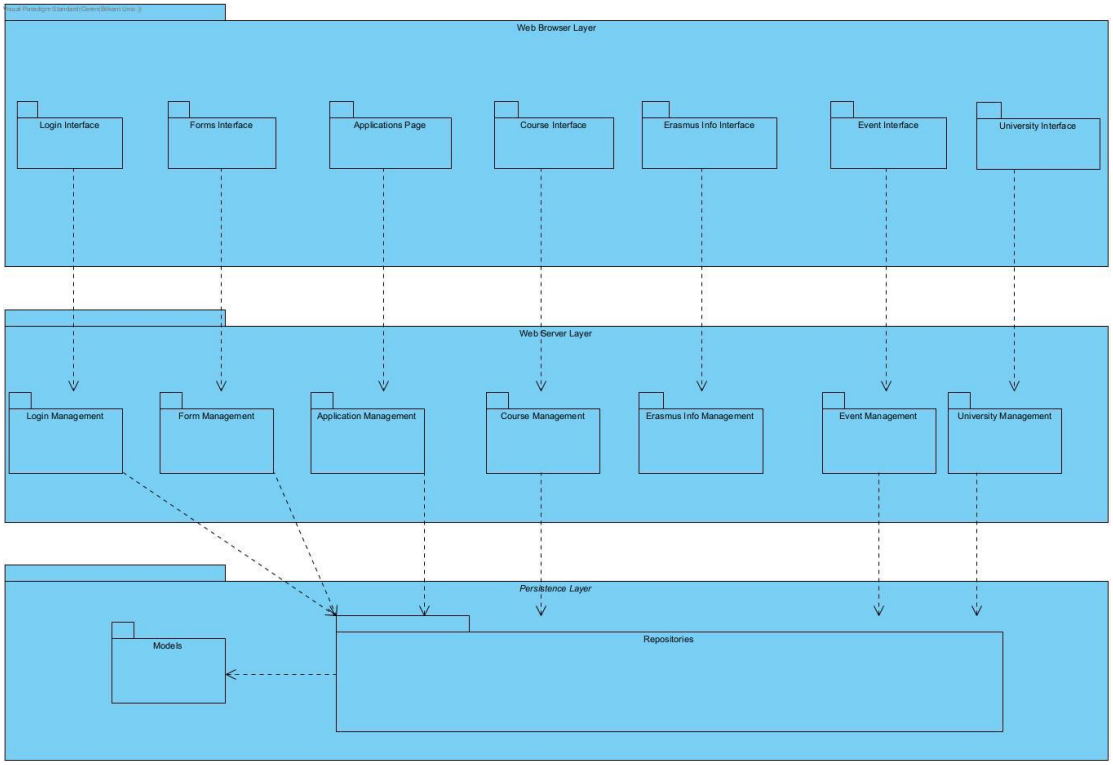


Fig. 1: 3-Layer Architecture

To develop our website, we chose 3-layer architectural style. Reasoning behind choosing 3-layer architecture is to develop each layer simultaneously without any dependencies, since we have limited time. Also, when needed, a change can be done without affecting other layers in minimum.

In our design, client layer is called Web Browser Layer, application layer Web Server Layer, and data layer is called Persistence Layer. In Figure 1, Web Browser Layer contains the user interfaces for the Erasmus Management System. The user interfaces play the role of boundary objects. The Web Server Layer contains the controllers that route the requests to their designated services to be handles. The Web Server Layer also contains the service classes that will handle user requests and interact with the Persistence Layer. Each interface has its own management package that handles its requests. Finally, the Persistence Layer stores the data that is needed for the system to operate.

2.2 Hardware/Software Mapping

ErasXchange is a web application that can run on any device with hardware strong enough to handle mediocre web applications. For the backend, Java, Spring Boot, PostgresSQL that is running inside a Docker container are used. ErasXchange can run on desktop computers, laptops, tablets,and phones. In order for ErasXchange to work properly, hardware that provides navigation is necessary. This can be a mouse and a monitor for a desktop computer, a working touchpad for a laptop, and a working touchscreen for phones. ErasXchange frontend is designed with React 18 which uses HTML 5 and CSS 3. Thus, any browser that supports React 18 should not have any issues running it [1]. Such browsers include but are not limited to: Edge, Firefox, Chrome, Safari, Opera, Brave, and Opera. In any browser that does not support React 18 applications, ErasXchange is not guaranteed to work properly. ErasXchange will be optimized for mobile browsers as well. Due to APIs and external libraries that are used, older systems might have a slower experience. Running one tab like this in such modern browsers requires at least 100 MB of RAM. However, including the modern system services which are necessary to use such browsers properly, users are recommended to have at least 4 GB of RAM [2]. A DigitalOcean (DO) virtual machine will be used to deploy and execute the application. Following the recommendations in the documentation on the DO website, a general-purpose virtual machine with 16 GB of RAM and an Intel Xeon Skylake CPU with 4 dedicated cores running at 2.7 GHz speed will be used to meet the needs of the application without having performance drops or system failures [3]. The selected plan includes 25 GB of SSD storage, which is estimated to be sufficient for the operation of the application. The storage will be extended with network-attached storage volumes in case of necessity.

2.3 Persistent Data Management

In ErasXchange, certain types of data like the user, course, application, and form data need to be stored in a database. Since the data are related, a relational database is needed. Persistent data will be stored in a PostgreSQL database hosted on a Docker container. PostgreSQL was chosen since it is an object-relational database, unlike its more known competitor MySQL. The object-relational aspect of Postgres will work

really well with our data set which is designed around entity objects. Docker is used for maintaining regularity across developers database-wise. To interact with the database the Java Persistence API (JPA) and Hibernate will be used since they allow database interactions with Java object entities and require very small SQL code to be written. With these technologies, all the CRUD operations will be handled.

2.4 Access Control and Security

ErasXchange controls each role's access and authorization for each page and function of the application. It uses the session information of the users to achieve control, with the help of Spring Session libraries. The interface of the website changes for each user role after logging in. Also, all user-specific pages are inaccessible to unauthorized users. Lastly, it utilizes the SHA-256 algorithm to encrypt user passwords and stores only the hashed values of the passwords to increase security and prevent sensitive data leaks.

	Student	**Coordinator**	**ISO**	**Board Member**
Login	**X**	**X**	**X**	**X**
Reset Password	**X**	**X**	**X**	**X**
Exchange Information	**X**	**X**	**X**	**X**
View Courses	**X**	**X**	**X**	**X**
Upload			**X**	

Application Data				
Publish Placements		**X**		
My Application	**X**			
Cancel Application	**X**	**X**		
Applications		**X**	**X**	**X**
Propose New Course	**X**			
Add New Course		**X**		
Add New University		**X**	**X**	

Table 1: Access Control Matrix

2.5 Boundary Condition

2.5.1 Initialization

Admin will activate the Docker container which will be hosting the database. The Spring server will also be initialized by the admins. Prior to getting to the steady state, the persisted data tables will be validated. Without the database validated, the system will go to a failure state.

2.5.2 Termination

While terminating the server will finish the already received requests but will reject any other requests coming in. The users won't be able to log in during termination. The termination sequence can be initiated manually by the admins.

2.5.3 Failure

Application does not crash during a failure. Instead, the application logs these failures to inform system admins. A failure can be caused by the presentation network problems in the server, the inability to reach the database container in a couple of tries, and an external service not responding.

3. Low-Level Design

3.1 Object Design Trade-offs

Maintainability versus Performance: ErasXchange is designed with components that divide the whole application as small as possible. Such an implementation can create a drawback for performance but it is very easy to maintain such a code.

Security versus Usability: ErasXchange will not store an electronic signature to provide security for the official files. Users will be enforced to download the file and add their own signature to it. Such an implementation increases the security of the application, but as a drawback, it decreases usability.

Reliability versus Usability: To provide a more reliable system, ErasXchange does not rely on the external system BCC in order to fetch instructor email while requesting to add a new course to the preapproval form. Rather the user has to enter the email manually to the form, which decreases usability.

3.2 Final Object Design

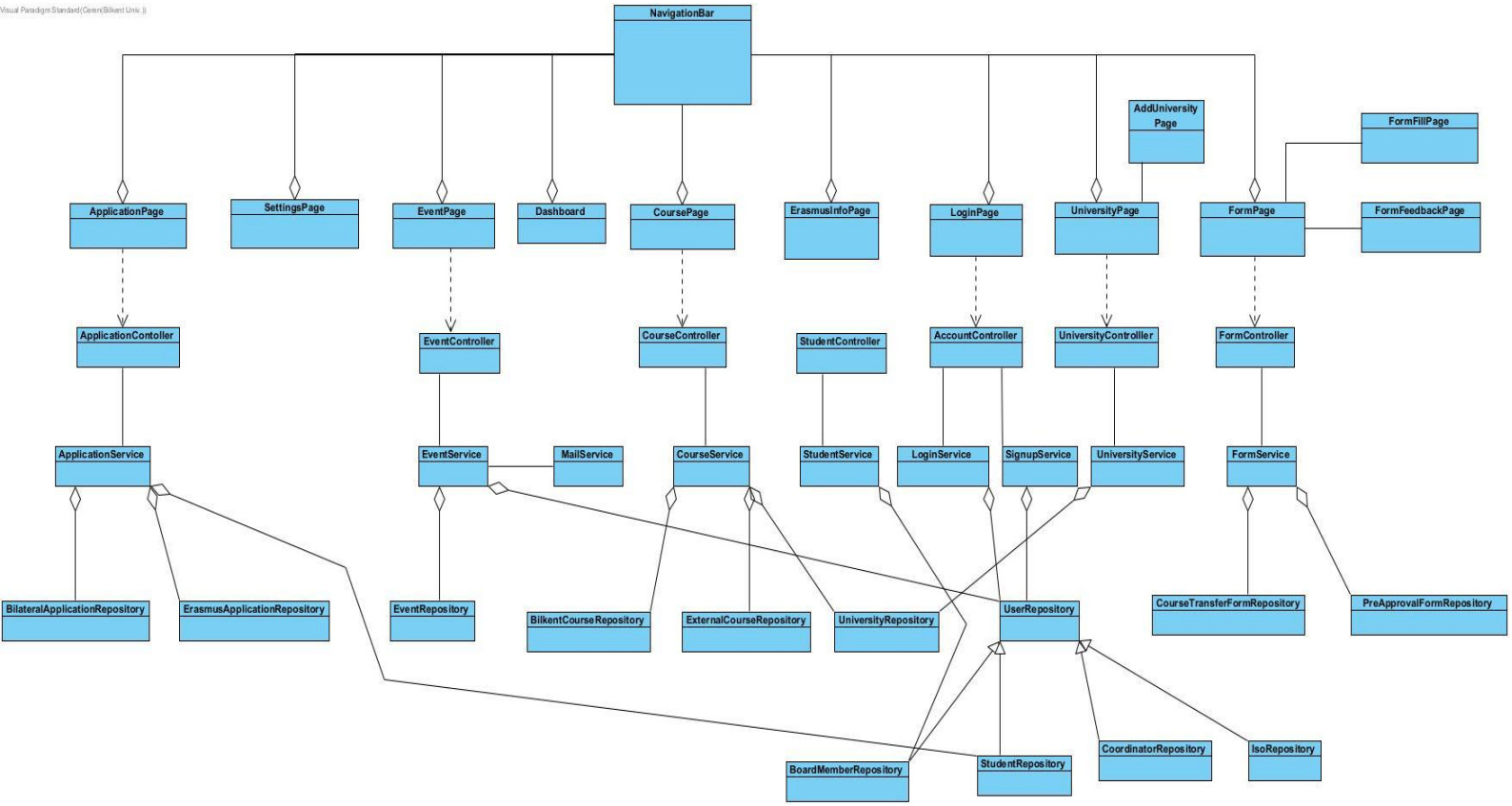


Fig. 2: Final Object Design Diagram

Fig. 2 is the final object design diagram. Diagram can also be reached from the following link for readability: https://github.com/deniz-dilaverler/BilkentErasmus/blob/main/reports/design-report/Class%20Diagram1.jpg

3.3 Layers

3.3.1 Web Browser Layer

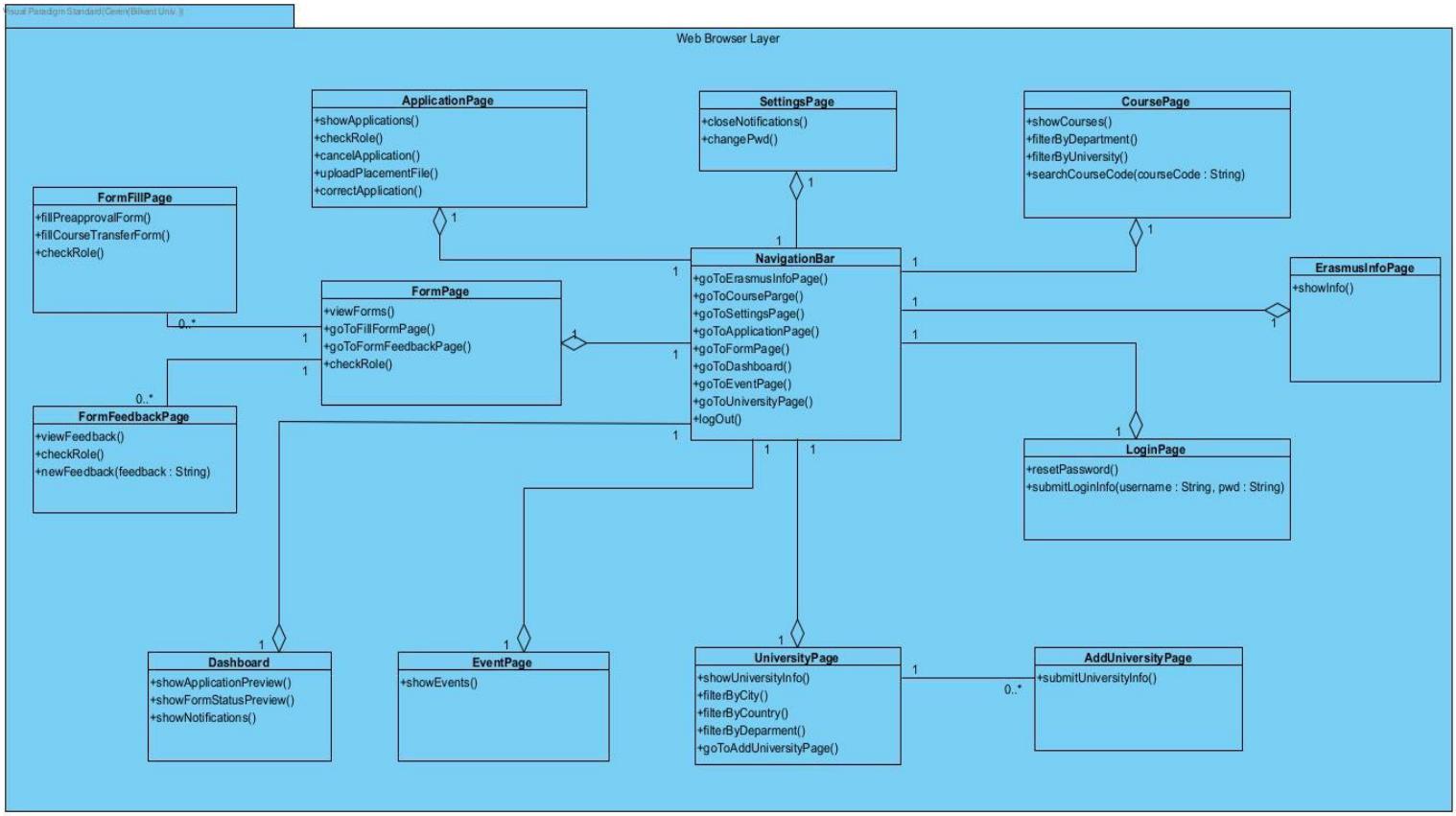


Fig. 3: Web Browser Layer Class Diagram

Figure 3 is the Web Browser Layer of the system. In other words, it is the user interface. It consists of boundary objects that are in communication with Web Server Layer.

3.3.2 Web Server Layer

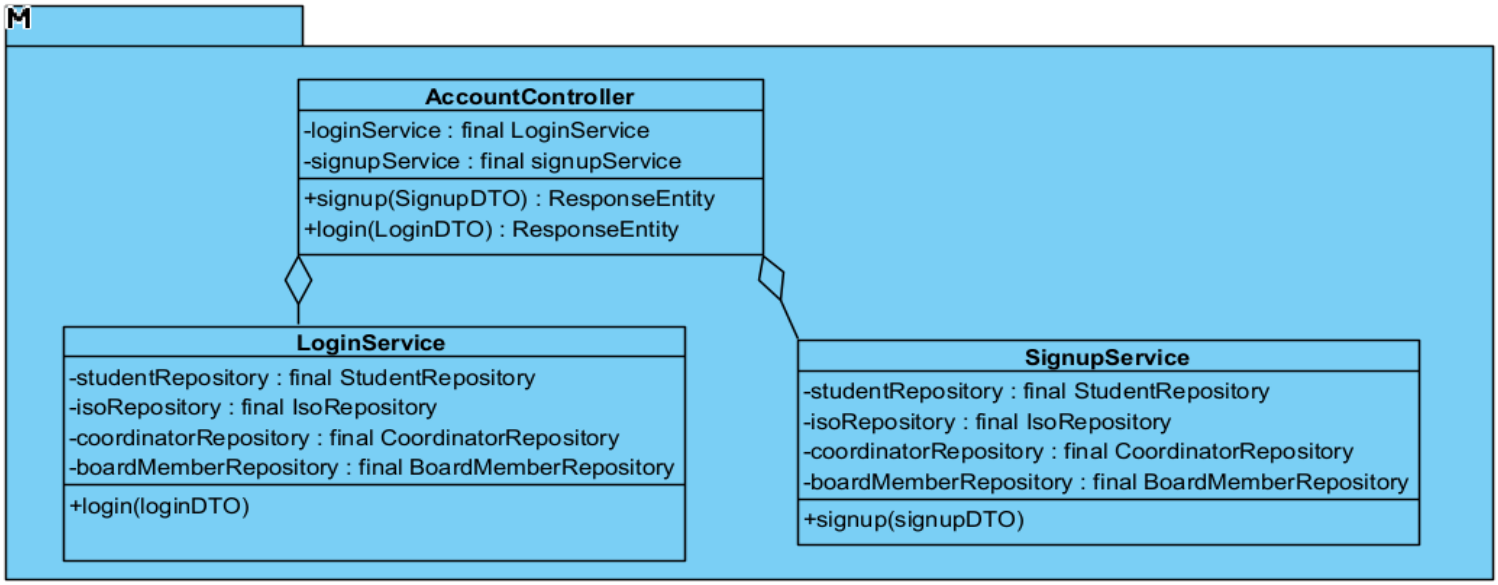


Fig. 4: Account Service Diagram

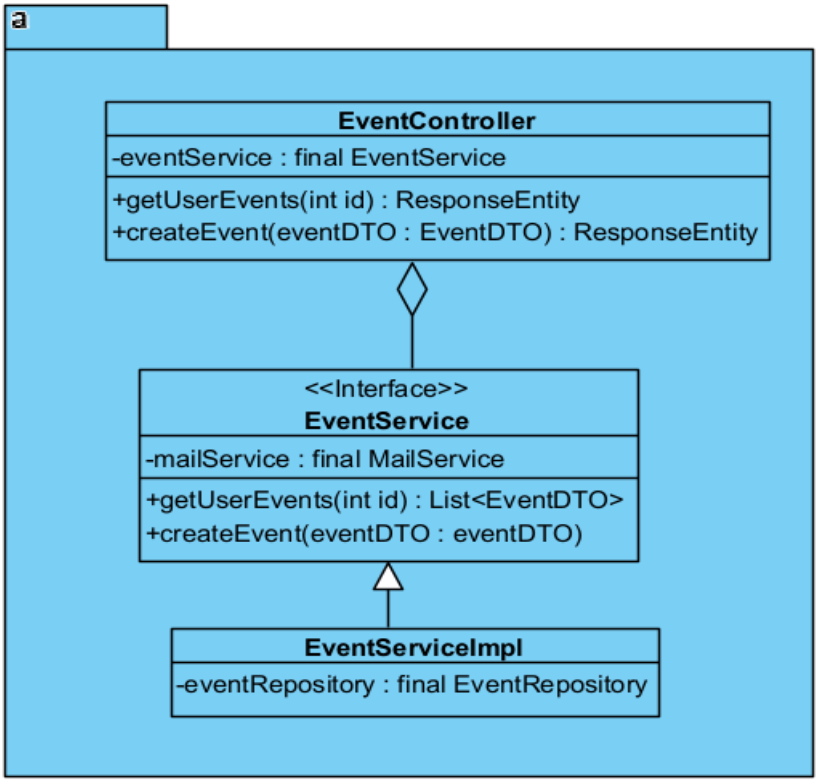


Fig. 5: Event Service Diagram

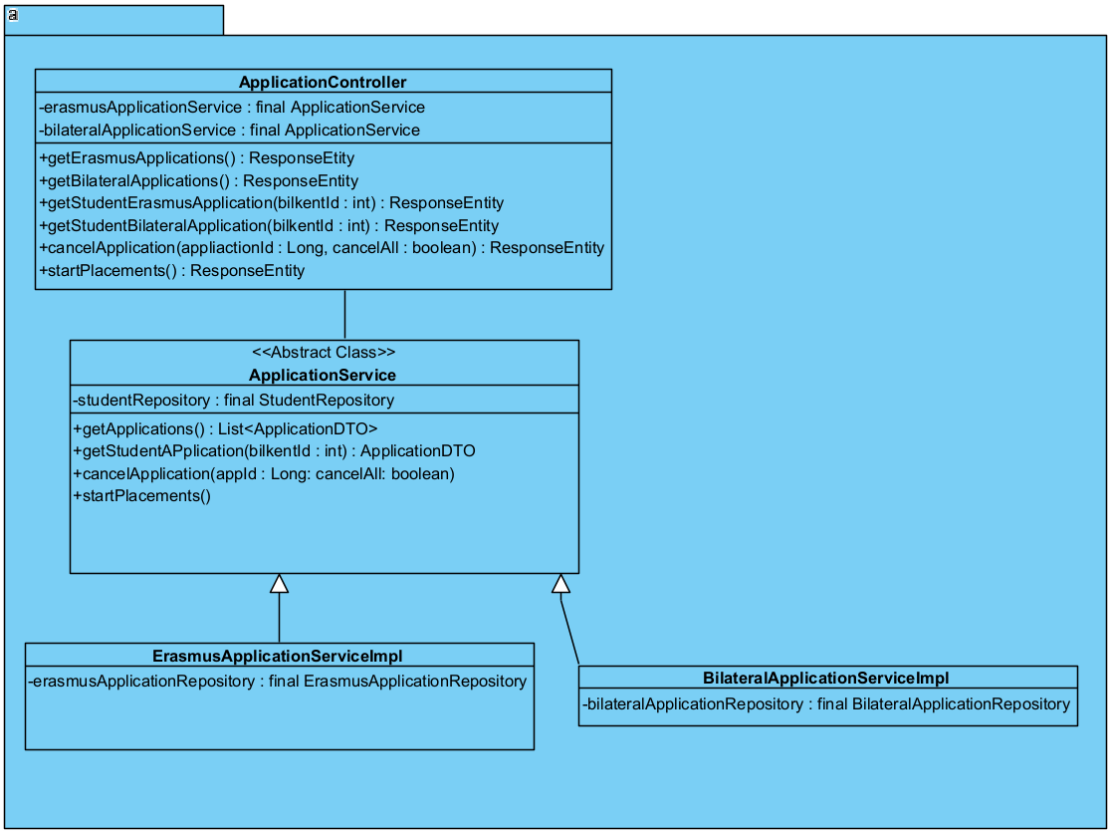


Fig. 6: Application Service Diagram

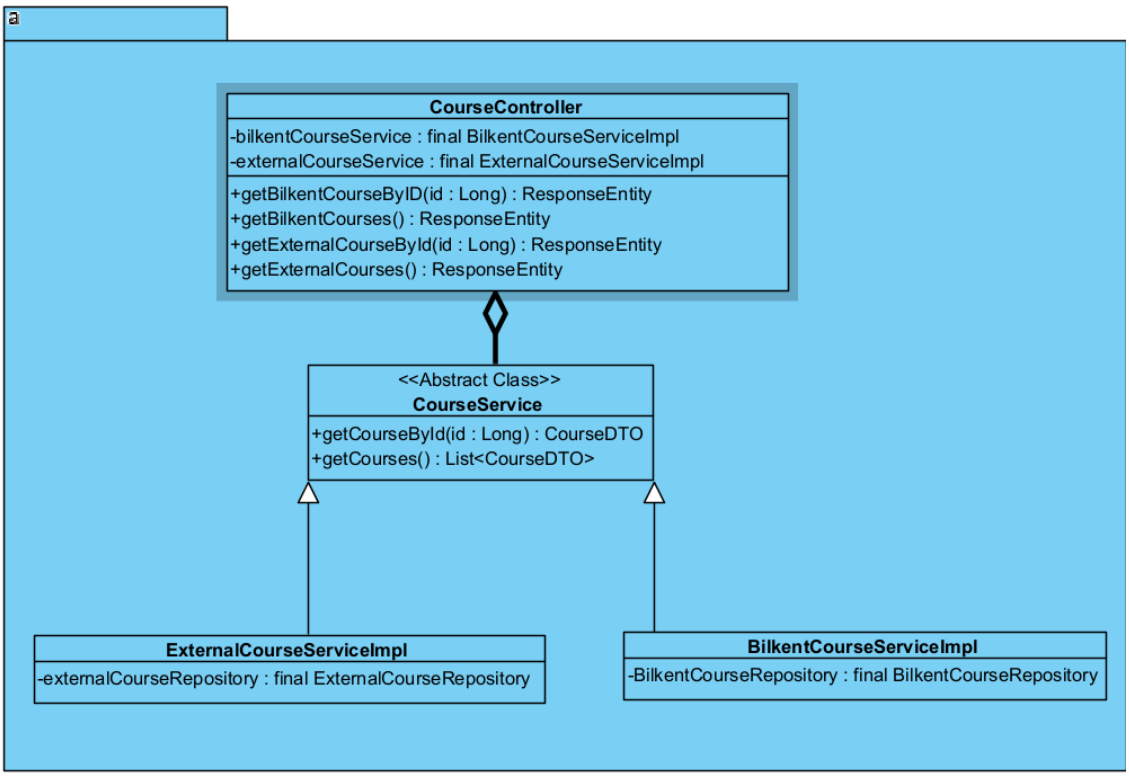


Fig. 7: Course Service Diagram

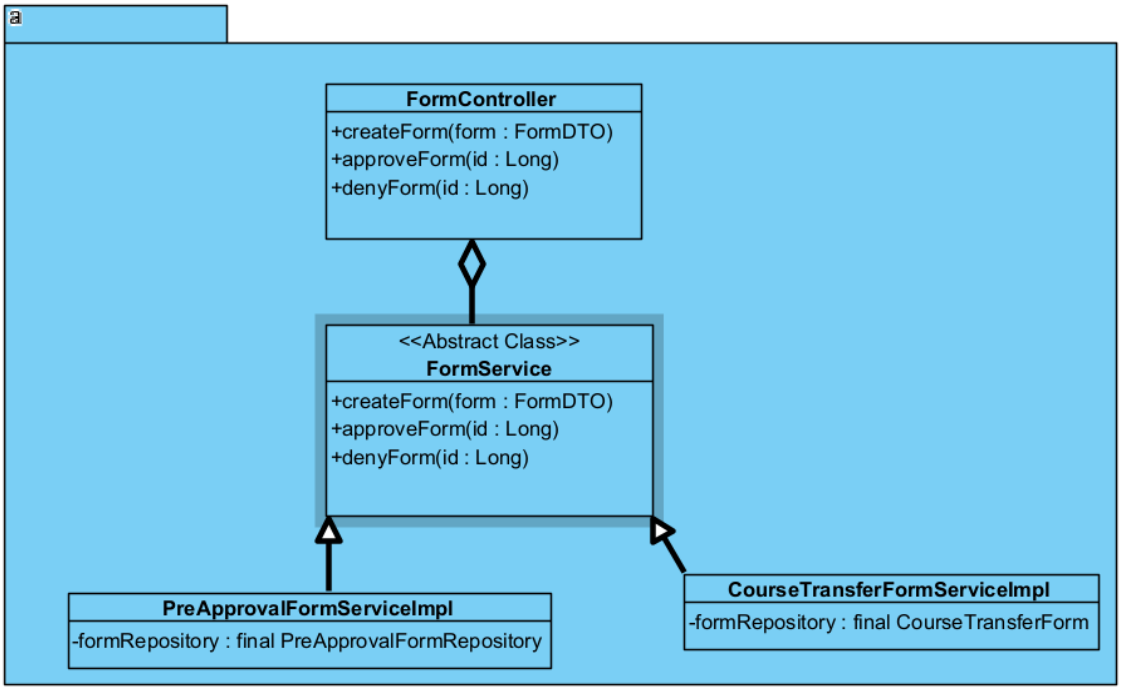


Fig. 8: Form Service Diagram

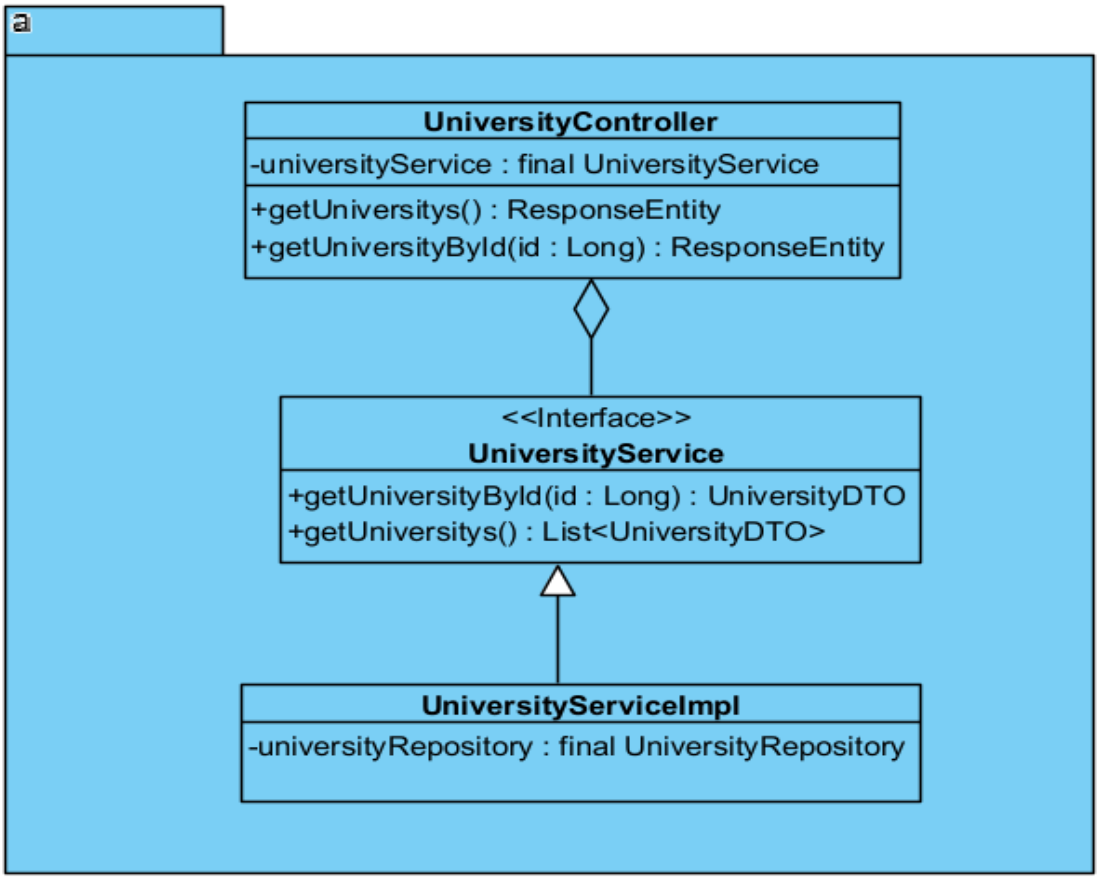


Fig. 9: University Service Diagram

The figures above represent the Server Layer of our application. Here, the user requests are processed, routed and then handled. This Layer communicates with the data management layer via the Repository classes and communicates with the UI layer using the API's

3.3.3 Persistence Layer

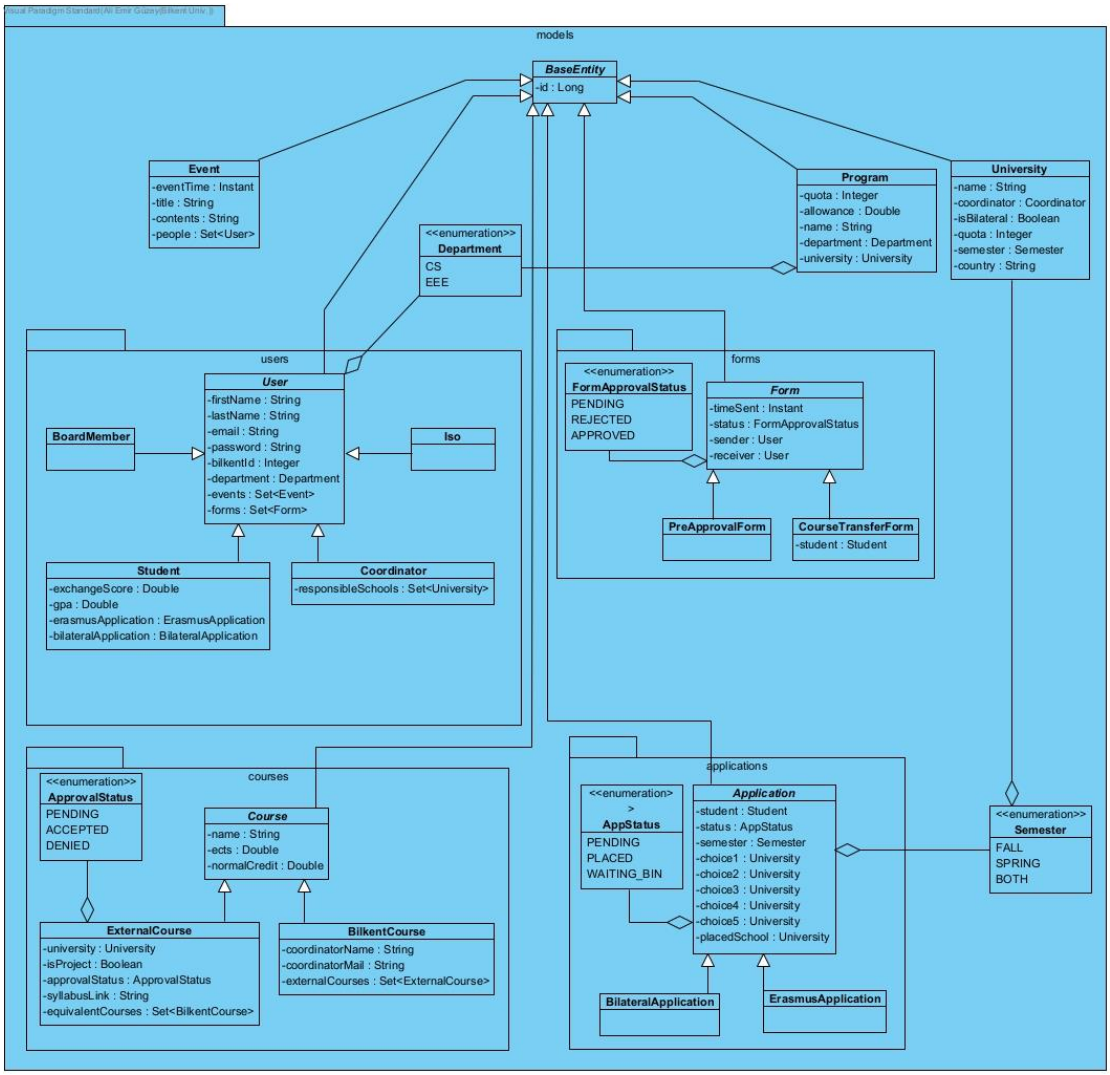


Fig. 10: Models Subsystem

All classes depicted in fig. 10 have getter and setter methods for each of their attributes, a constructor with no arguments, and a constructor with an argument for each of the attributes.

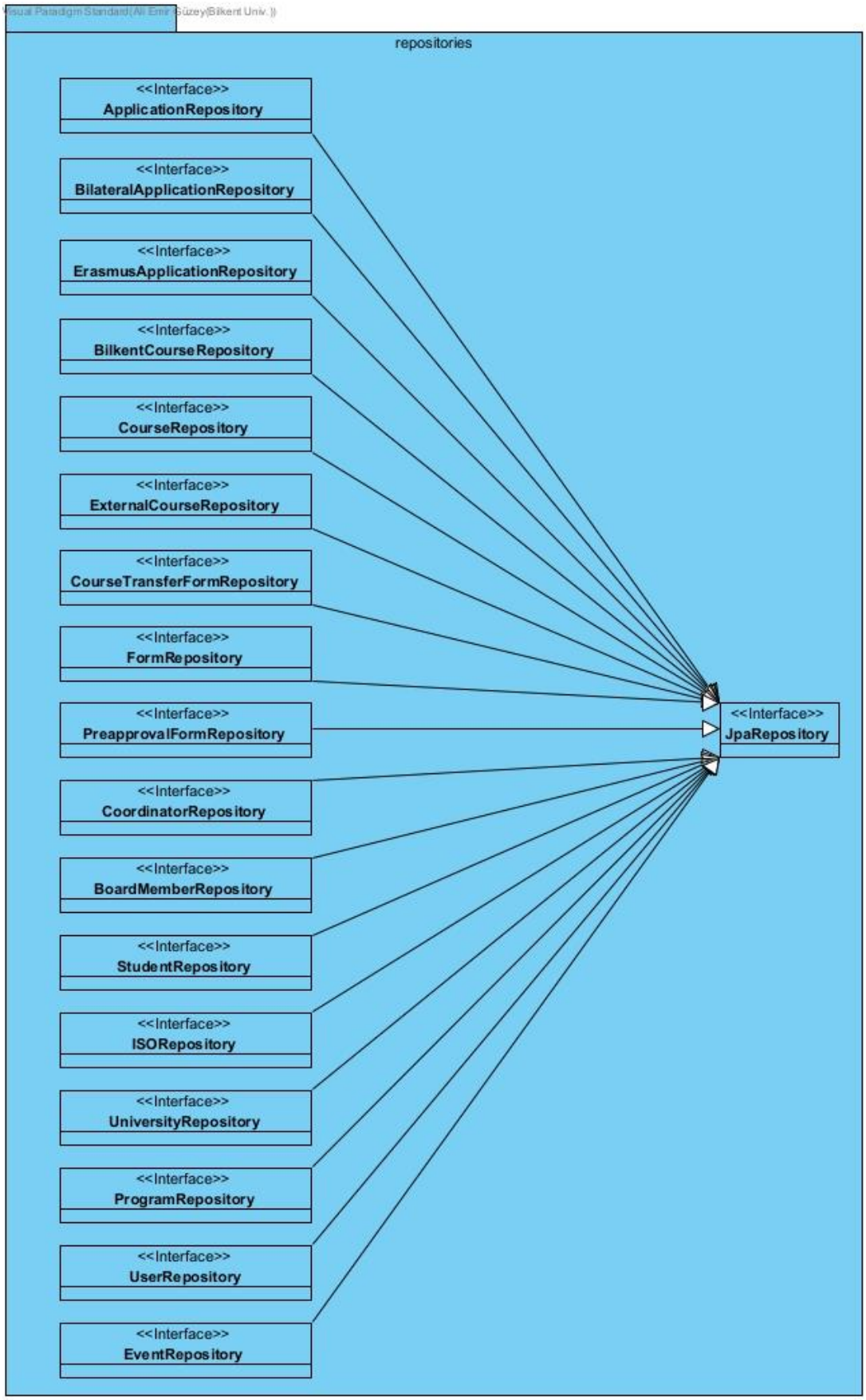


Fig. 11: Repositories Subsystem

Persistence Layer handles database management and helps the Web Server Layer with its functions related to data querying or data manipulation through its subsystems, Models, and Repositories. The model's subsystem maps information from relational database tables' rows to Java objects, and the Repository subsystem handles communication with the database with its interfaces descending from the JpaRepository interface.

3.4 Packages

3.4.1 Developer Introduced Packages

3.4.1.1 Models

The Models package holds the entity objects that are used for data persistence to the database and general model classes that will be used in the system.

3.4.1.2 Repositories

The repositories package holds the classes that are responsible for doing the CRUD operations on the database.

3.4.1.3 DTOs

The DTOs package stores the Data Transfer Objects which are purely data objects that will be used for transferring data within the system and to the client.

3.4.1.4 Services

The services package stores the classes that handle the business logic of operations in our application.

3.4.1.5 Mappers

Mappers will be the classes that convert the DTOs into entities and vice versa.

3.4.1.6 Security

The security package of the system will be responsible for authentication classes and all-around security of the system.

3.4.2 External Packages

3.4.2.1 org.springframework.boot.spring-boot-maven-plugin

This external package enables Maven build automation tool within the project. Maven controls software projects through Project Object Models (POM) [4].

3.4.2.2 org.springframework.boot.spring-boot-starter-data-jpa

Spring Data JPA allows the project to perform persistence actions such as querying and updating the database.

3.4.2.3 org.springframework.boot.spring-boot-starter-data-rest

Spring Data REST enhances the Spring Data JPA package to provide more RESTful-APIs-friendly data to ease the development process.

3.4.2.4 org.springframework.boot.spring-boot-starter-web

Spring Web helps to create web-application projects easily. It includes an HTTP client and configurations for web applications.

3.4.2.5 org.springframework.boot.spring-session-core

Spring Session is used in the project to perform actions on users' session data.

3.4.2.6 org.springframework.boot.spring-boot-devtools

Spring Boot Dev Tools provide a number of useful functions to speed up the development process.

3.4.2.7 org.springframework.boot.spring-boot-starter-test

This package enables testing for the project by importing test modules and other testing libraries such as JUnit.

3.4.2.8 org.postgresql.postgresql

This package configures the project to connect to a PostgreSQL relational database.

3.4.2.9 org.mapstruct.mapstruct

Mapstruct generates code for Java bean mapping to ease the development process.

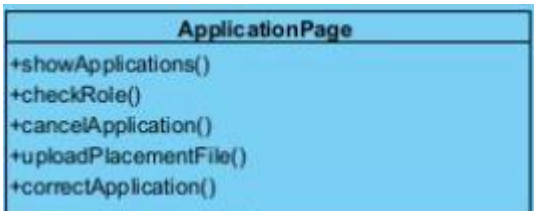
3.4.2.10 org.projectlombok.lombok

Lombok uses annotations to generate setter and getter methods for Java classes.

3.5 Class Interface

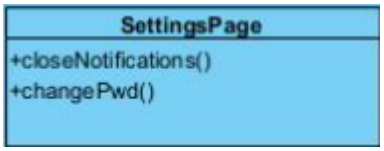
3.5.1 UI Layer Class Interfaces

3.5.1.1 ApplicationPage



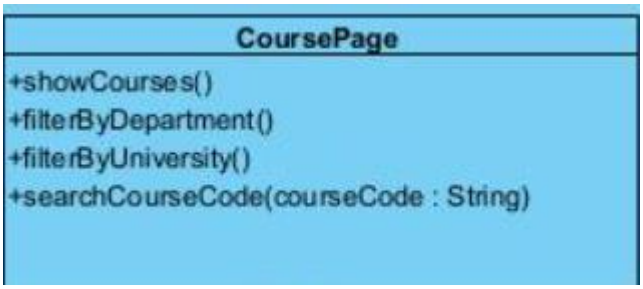
This page's function is different for different users. Students can use this page to view their own application and cancel the application if they wish. Coordinators use this page to view all of the applications made by students and do placements accordingly.

3.5.1.2 SettingsPage



This page allows users to change their settings to their own preferences. Users can switch between a dark mode and a light mode, they can customize the notifications they receive. They can also change their data such as passwords.

3.5.1.3 CoursePage

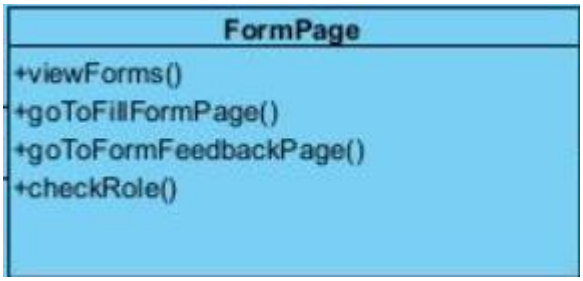


Users can see the courses provided on this page. They can filter these courses by university and by department that provides the course. They can also use the search functionality in order to search for a course by name.

3.5.1.4 FormFillPage

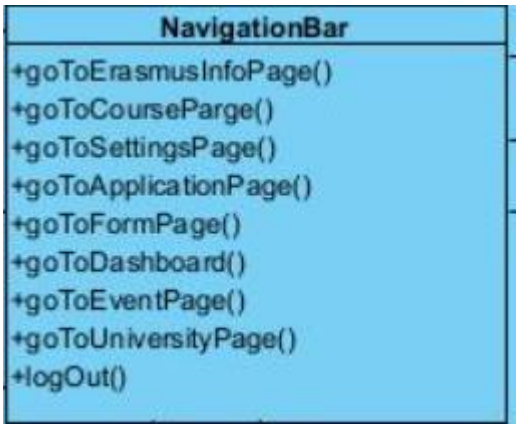
On this page students can create pre approval forms while coordinators can create Course transfer forms.

3.5.1.5 FormPage



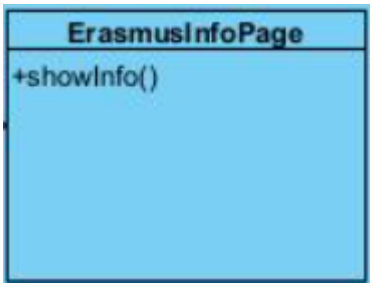
This page's function is different for different users.Students can use this page to view pre approval forms they have submitted and check their feedback if there is any. Coordinators use this page to view Course Transfer Forms they have submitted and check their feedback if there is any. Both actors can navigate to FormFillPage from here if they wish to create a new form.

3.5.1.6 NavigationBar



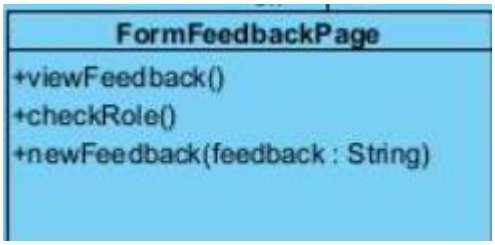
Navigation is handled with a Sidebar which will be visible on most of the pages. Users can easily navigate between by clicking the corresponding buttons on the Sidebar.

3.5.1.7 ErasmusInfoPage



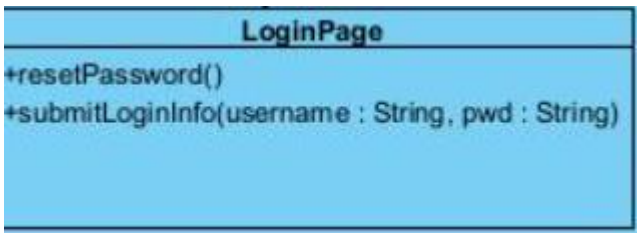
In this page, users can see information about the exchange program and the application.

3.5.1.8 FormFeedbackPage



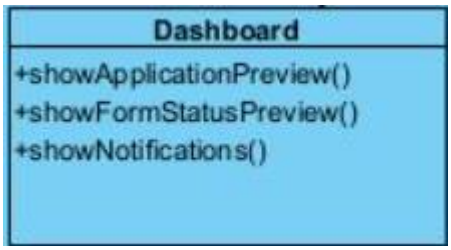
This page is used for viewing the feedback given or giving feedback to the current form.

3.5.1.9 LoginPage



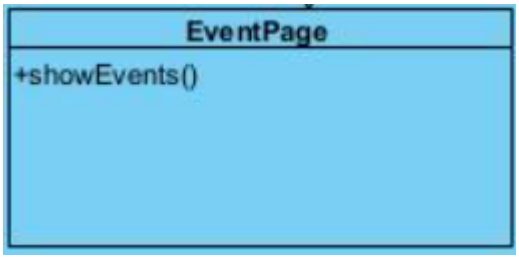
This page is where users must enter the correct data (ID and password) in order to interact with the application. Users can also reset their passwords if they forgot it.

3.5.1.10 Dashboard



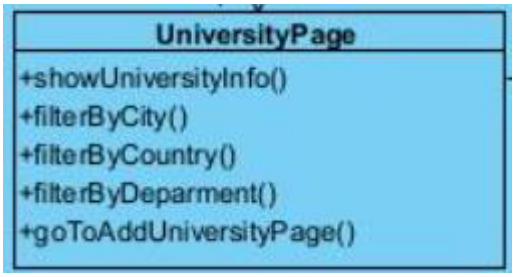
This is the main page shown after a user logs in. Here users can see the status of the forms they have submitted before. Additionally, students can see the status of their Erasmus/Exchange application.

3.5.1.11 EventPage



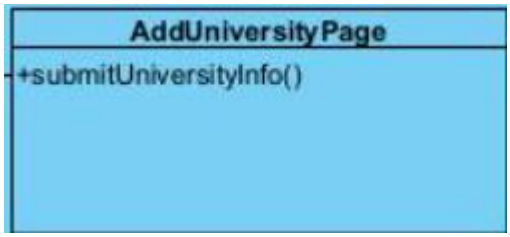
This page shows urgent events to inform the user. This page also allows users to access this urgent event's functionality with 1 click.

3.5.1.12 University Page



Users can see the universities and their information on this page. They can filter the universities by their cities, the departments, and their countries. Coordinators can also add new universities and provide information for them in this page.

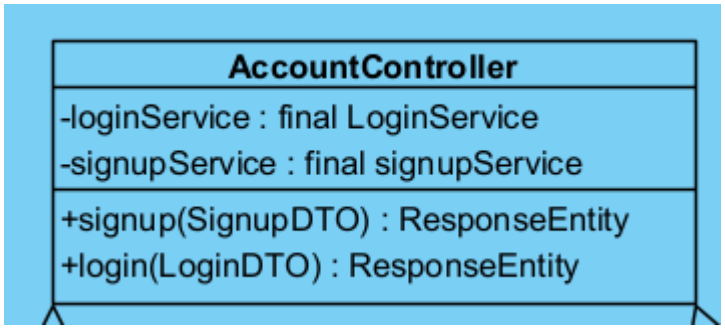
3.5.1.13 AddUniversityPage



If coordinators want to add a university, they are directed to this page where they provide the necessary information for the operation.

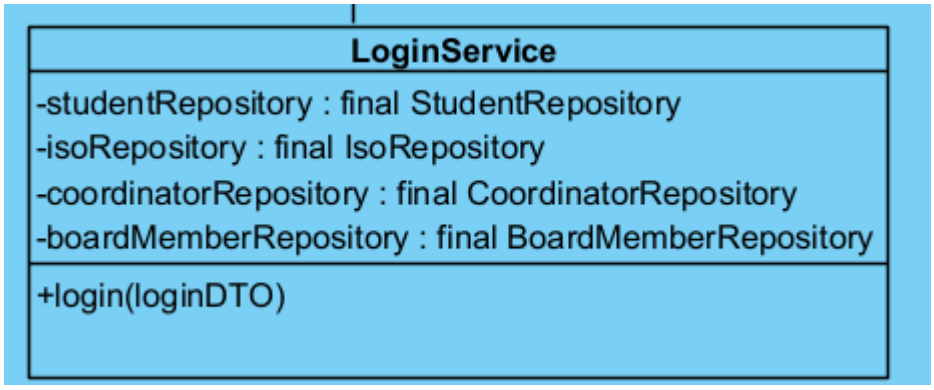
3.5.2 Service Layer Class Interfaces

3.5.2.1 AccountController



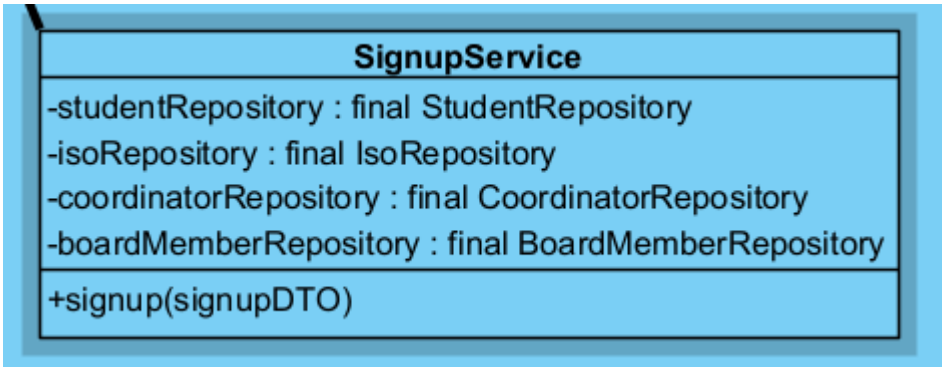
Routes the login and signup requests to their relevant services.

3.5.2.2 LoginService



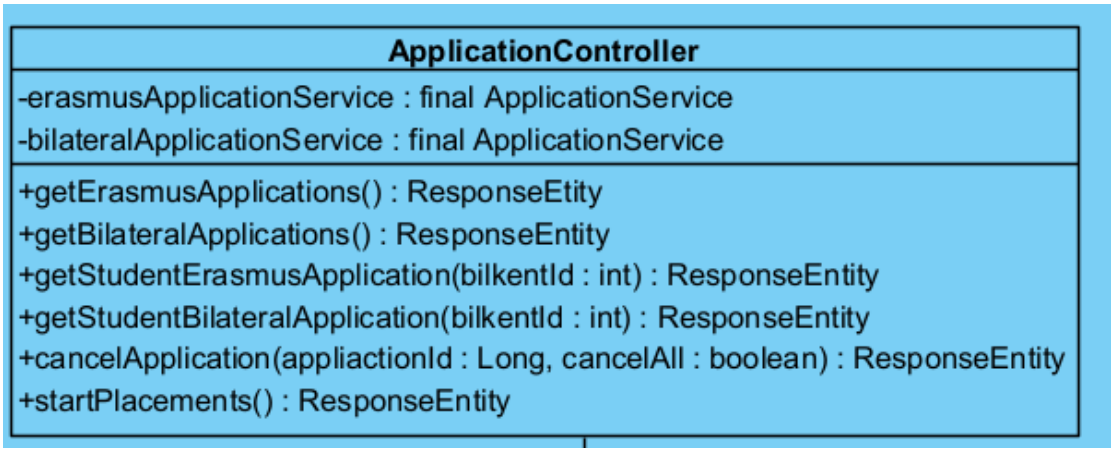
Checks the user in the database and logs the user in if authenticated.

3.5.2.3 SignupService



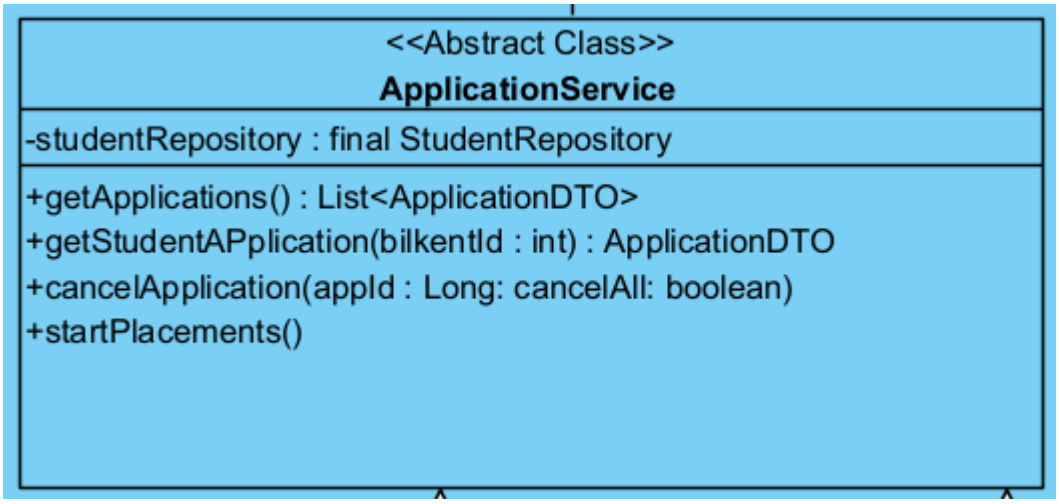
This service handles the requests for creating a new user account.

3.5.2.4 ApplicationController



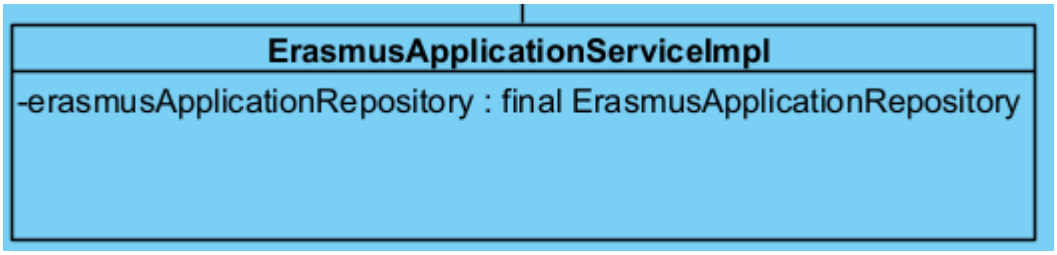
This class processes and routes the requests about Erasmus and Exchange applications.

3.5.2.5 Application Service



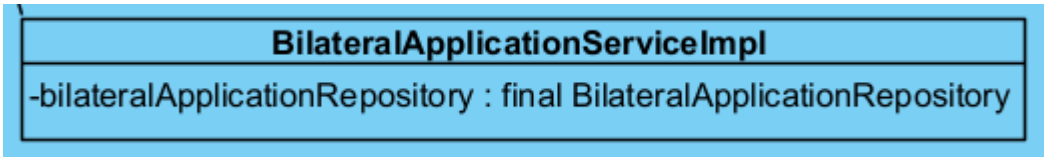
This class handles the base requests for any application.

3.5.2.6 ErasmusApplicationServiceImpl



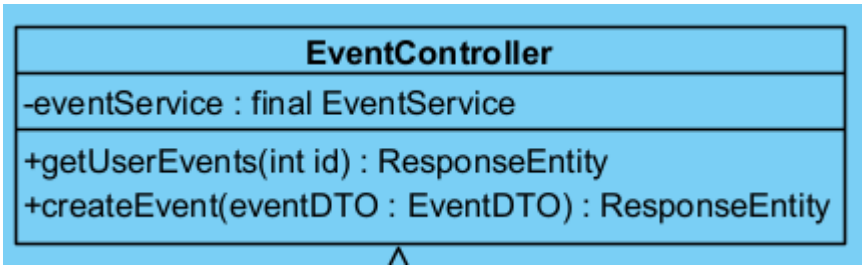
Extends ApplicationService to process requests on Erasmus applications.

3.5.2.7 BilateralApplicationServiceImpl



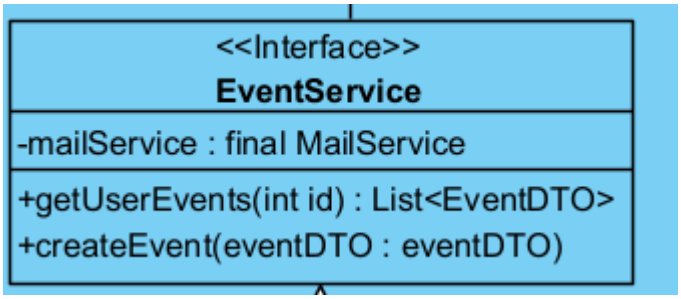
Extends ApplicationService to process requests on Exchange applications.

3.5.2.8 BilateralApplicationServiceImpl



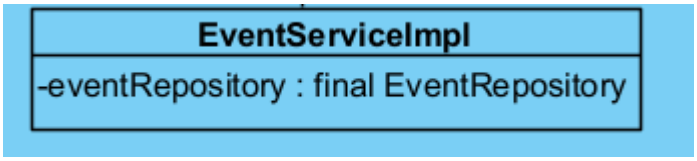
This class routes the requests for events that the users will be notified about

3.5.2.9 EventService



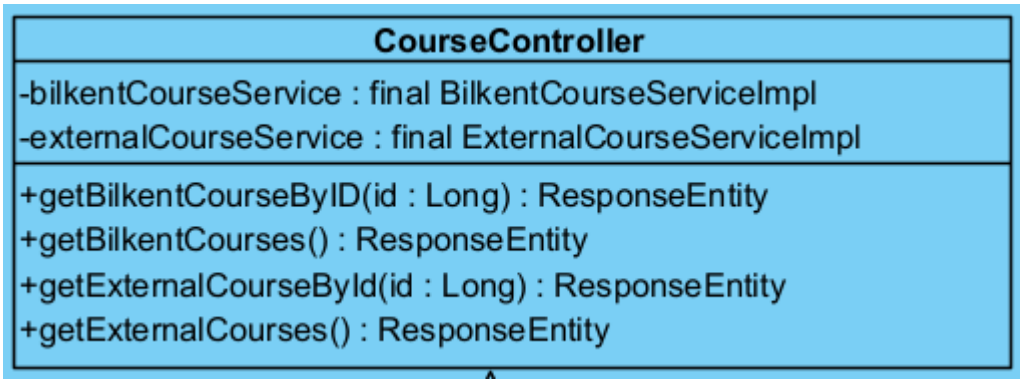
This class is a template for any event service that will be implemented to the system.

3.5.2.10 EventServiceImpl



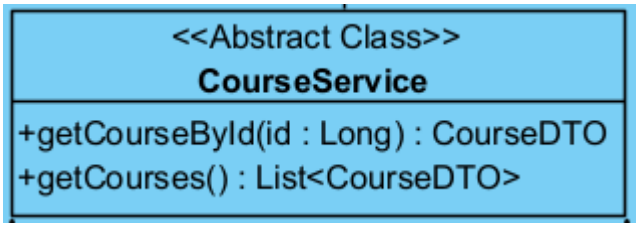
Implements the EventService interface and handles the events that the user will be notified about.

3.5.2.11 CourseController



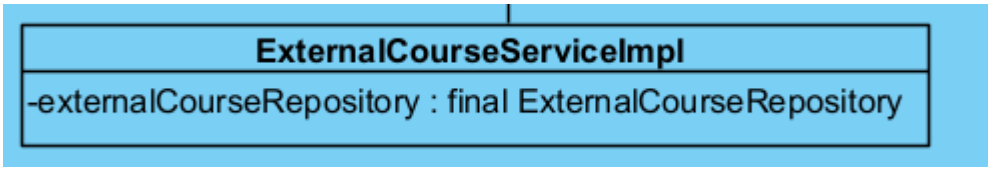
This class routes the requests about courses to their relevant service.

3.5.2.12 CourseService



This Abstract Class handles the base requests for courses.

3.5.2.13 ExternalCourseServiceImpl

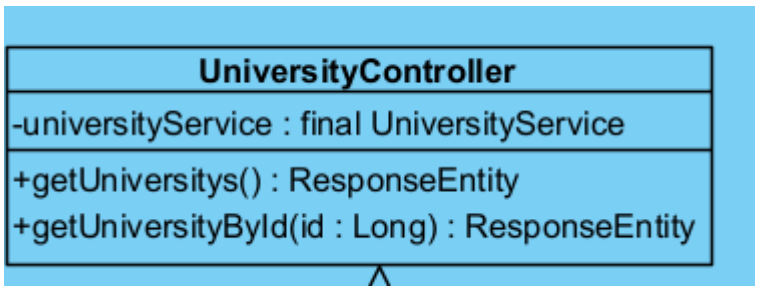


Extends the CourseService abstract classes to handle requests about external courses.

3.5.2.14 BilkentCourseServiceImpl

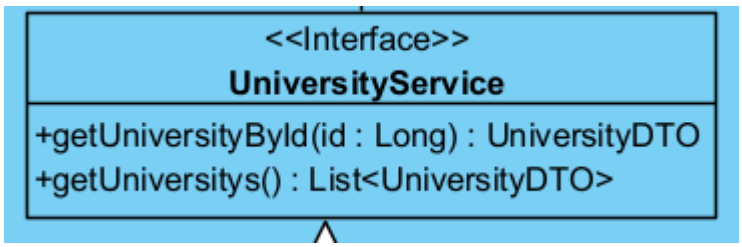
Extends the CourseService abstract classes to handle requests about Bilkent courses.

3.5.2.15 UniversityController



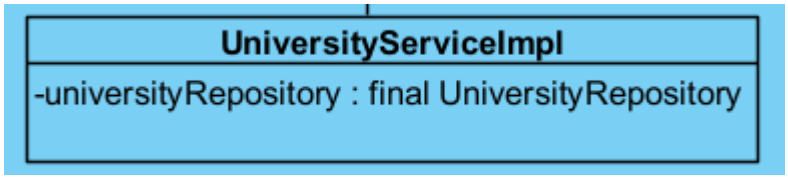
This class routes the requests about Universities to their relevant service.

3.5.2.16 CourseController



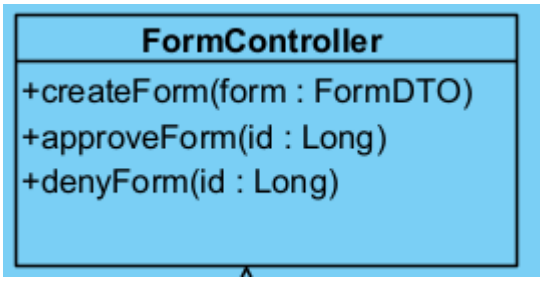
This class is a template for any UniversityService implementations.

3.5.2.17 UnivesityServiceImpl



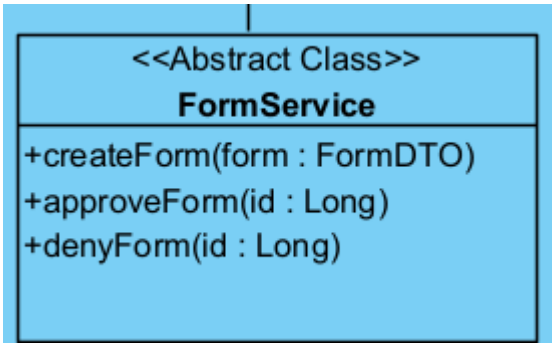
Implements the UniversityService interface and handles the requests about the universities.

3.5.2.18 FormController



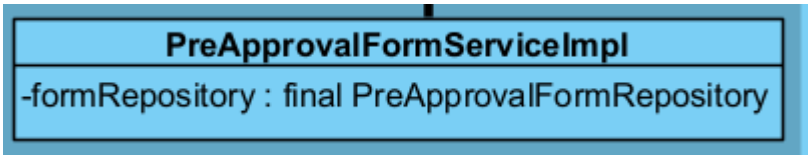
This class routes the requests about Forms to their relevant service.

3.5.2.19 FormService



This abstract class handles the base requests for forms.

3.5.2.20 PreApprovalFormServiceImpl



Extends the FormService abstract class to handle requests about preapproval forms.

3.5.2.21 FormController

Extends the FormService abstract class to handle requests about course transfer forms.

3.5.3 Data Management Layer Class Interface

[On next iteration]

4. Glossary & references

[1] G. Tiwari, “Browser compatibility for reactjs web apps,” *BrowserStack*, 14-Nov-2022. [Online]. Available: https://www.browserstack.com/guide/browser-compatibility-for-reactjs-web-apps. [Accessed: 29-Nov-2022].

[2] GWS Team, “How system requirements for browsing the internet have changed - 3,” *GWS Media*, 18-May-2022. [Online]. Available: https://www.gwsmedia.com/articles/how-internet-system-requirements-have-changed-3. [Accessed: 29-Nov-2022].

[3] “Choosing the right droplet plan,” *DigitalOcean Documentation*. [Online]. Available: https://docs.digitalocean.com/products/droplets/concepts/choosing-a-plan/. [Accessed: 29-Nov-2022].

[4] B. Porter, J. van Zyl, and O. Lamy, “Welcome to Apache Maven,” *Maven*. [Online]. Available: https://maven.apache.org/. [Accessed: 29-Nov-2022].