Bilkent University
Department of Computer Engineering

CS319 Term Project

Section 3
Group 3D
Erasmus++

Project Design Report

Group Members

-Yunus Eren Türkeri (22001842)
-Ender Utlu (22001983)
-Ege Ayan (22002478)
-Alp Ertan (22003912)
-Mehmet Feyyaz Küçük (22003550)

1.0 Introduction **4**
1.1 Purpose of the System 4
1.2 Design Goals 4
1.3 Criteria 4
1.3.1 End User Criteria 4
1.3.2 Maintenance Criteria 5
1.3.3 Performance Criteria 5
1.3.4 Security Criteria 5
1.4 Definitions 6

2.0 System Architecture **7**
2.1 System Decomposition 7
2.2 Hardware/Software Mapping 8
2.3 Persistent Data Management 9
2.4 Access Control and Security 9
2.5 Boundary Conditions 10
2.5.1 Initialization 10
2.5.2 Termination 10
2.5.3 Failure 11

3.0 Low-Level Design **11**
3.1. Object Design Trade-offs 11
3.2 Final Object Design 12
3.3 Layers 13
3.3.1 User Interface Layer 13
3.3.2 Web Server Layer 14
3.3.3 Data Management Layer 15
3.4 Packages 17
3.4.1 Implementation Packages 17
3.4.1.1 Authorization 17
3.4.1.2 Candidate Package 17
3.4.1.3 Coordinator Package 17
3.4.1.4 Security 17
Package that contains elements that are related to security components. 17
3.4.1.5 Repositories 17
3.4.1.6 Models (Entities) 17
3.4.1.7 Profile Management 17
3.4.2 External Library Packages 17
3.4.2.1 springframework.postgresql 17
3.4.2.2 springframework.data.jpa 17
3.4.2.3 springframework.jdbc 17
3.4.2.4 springframework.security 17

3.4.2.5 io.jsonwebtoken 18
3.4.2.7 springframework.restdocs 18
3.4.2.8 React.react 18
3.4.2.9 React.react-dom 18

4. References **18**

1.0 Introduction

1.1 Purpose of the System

Erasmus++ is a web application for Bilkent students who want to apply for Eramus and for coordinators to follow Eramus application, nomination and placement procedures, and assist students with their questions when necessary. The current Erasmus Application System is confusing for students and requires the coordinators to conduct most procedures manually (i.e., increased amount of paperwork). This system aims to implement a user-friendly UI that will make it easier for students and coordinators. Students will be able to apply for Erasmus and keep track of their progress and coordinators will be able to view the progress of their appointed students. This system also aims to reduce paper usage and mail exchange between students and coordinators compared to the former system.

1.2 Design Goals

Design is crucial for the implementation of the system since the features and the expectations of the design that should be focused on will be more concrete and thus easier to create. By determining the design goals, the design backbone of the system can be formed to a larger extent. Some of the non-functional requirements will be explained in the latter section of the report (see 1.3).

1.3 Criteria

1.3.1 End User Criteria

Usability: For any candidate or coordinator that are to use Erasmus++, they will be able to interact with the functional components of the system as the dedicated navigation bars and buttons in minimalistic UI will serve on their purposes. Since the system favors query support, in case the users lose their way or get confused about the usage of any component, they will have a chance to communicate with the system admins or directly check the FAQ section, which is accessible in all UI scenes.

1.3.2 Maintenance Criteria

Extendibility: Since this software application is designed to be continuously used in the following years after its launch, extendibility of it is a primary concern. While Erasmus++ is designed to handle online Erasmus applications, the software can be extended and it can become capable of handling online Exchange applications for the university.

Modifiability: Each feature of Erasmus++ was regarded as a singular module that works independently from the other modules which are representing other features of the software. This modular approach enables the software to be modifiable with the minimal amount of bugs occurring. With the feedback coming from the end users, the source code spread across multiple modules can be edited or reimplemented.

Reusability: The dynamic of getting information from the user, handling the information and saving that to a database system is applicable for any other program that serves the similar purpose rather than Erasmus domain. The components that are declared for the user interface can be used for any other user interface implementation since they can perform user input/handle operations efficiently and provide ease for the user.

Portability: Erasmus++ is a web application, meaning that it will run on any hardware that can run a web browser. This means that the software will be able to run in any operating system. Since almost all hardware devices with an internet connection can render web pages, the application will be very portable.

1.3.3 Performance Criteria

Response Time: Fetching data from the database and processing it should be optimal so that users' are not delayed upon taking action. The website should not have long waiting times and transition between different pages should be smooth. In general, slow response times can make the users' frustrated, so it is important for the web application to be running fast and smoothly.

1.3.4 Security Criteria

Erasmus++ is using JSON Web Token (JWT for short) for the authorization of its users and the security of the information exchange between the parties. JWT allows the software to verify that the content it is receiving has not been tampered with and it is indeed coming from a trusted party. JWT is an open, industry standard RFC 7519 method.

1.4 Definitions

Spring Boot - open source Java-based framework used to create a micro Service
JSON Web Token (JWT) - industry standard way for secure transmission of information between parties
RFC 7519 - Remote function call, process of calling a function module that is residing on a remote machine
React - a JavaScript library for building user interfaces, this will be used for front-end

2.0 System Architecture

2.1 System Decomposition

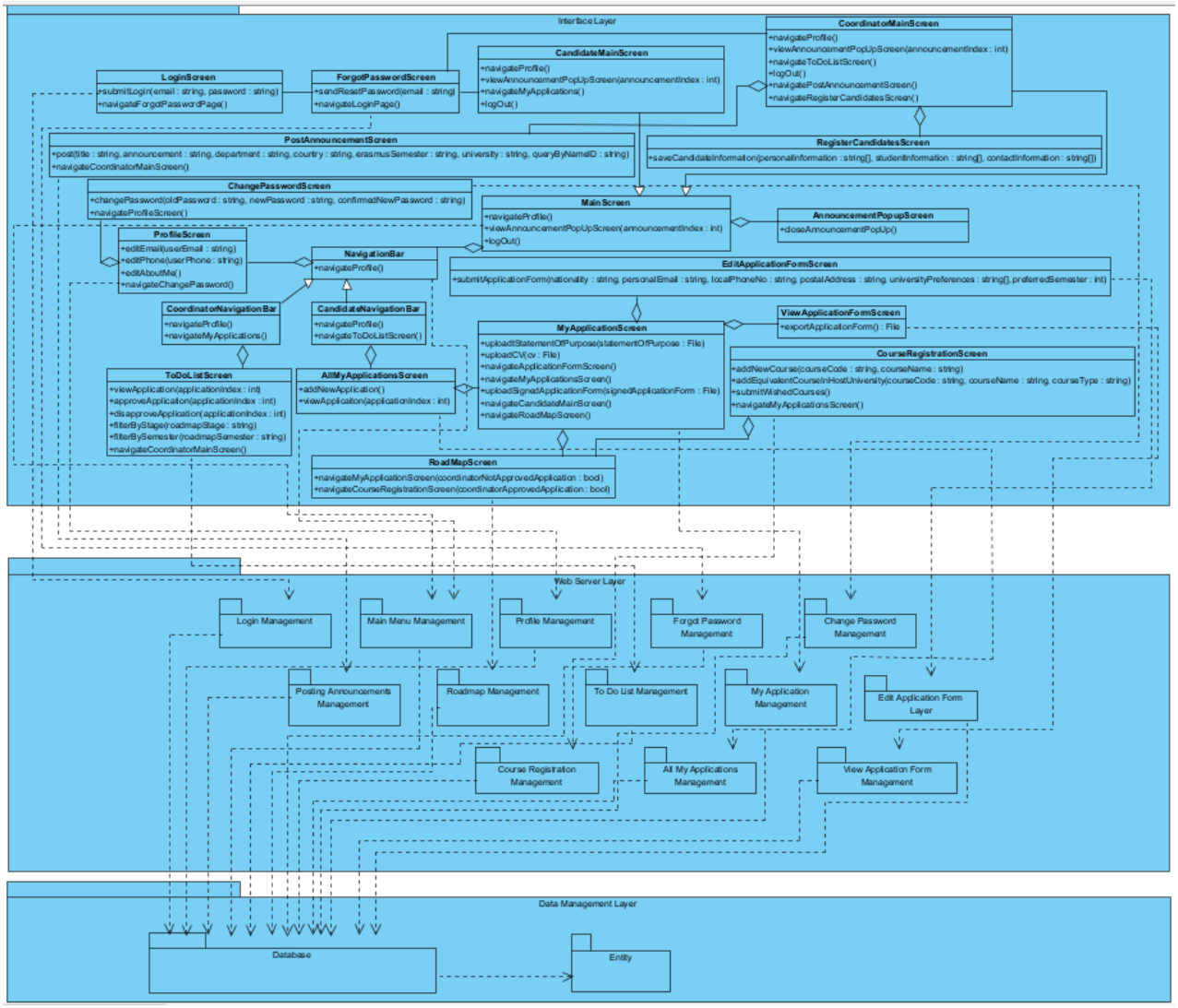


Fig. 1: Schematics for Subsystem Decomposition

In order to make the implementation easier, we will decompose our system into subsystems. The system will be decomposed based on different interface components (Login interface, profile interface, etc.). Every subsystem will be designed in terms of interface requirements; what actions can users take on a specific interface, and which information will be required by that interface.

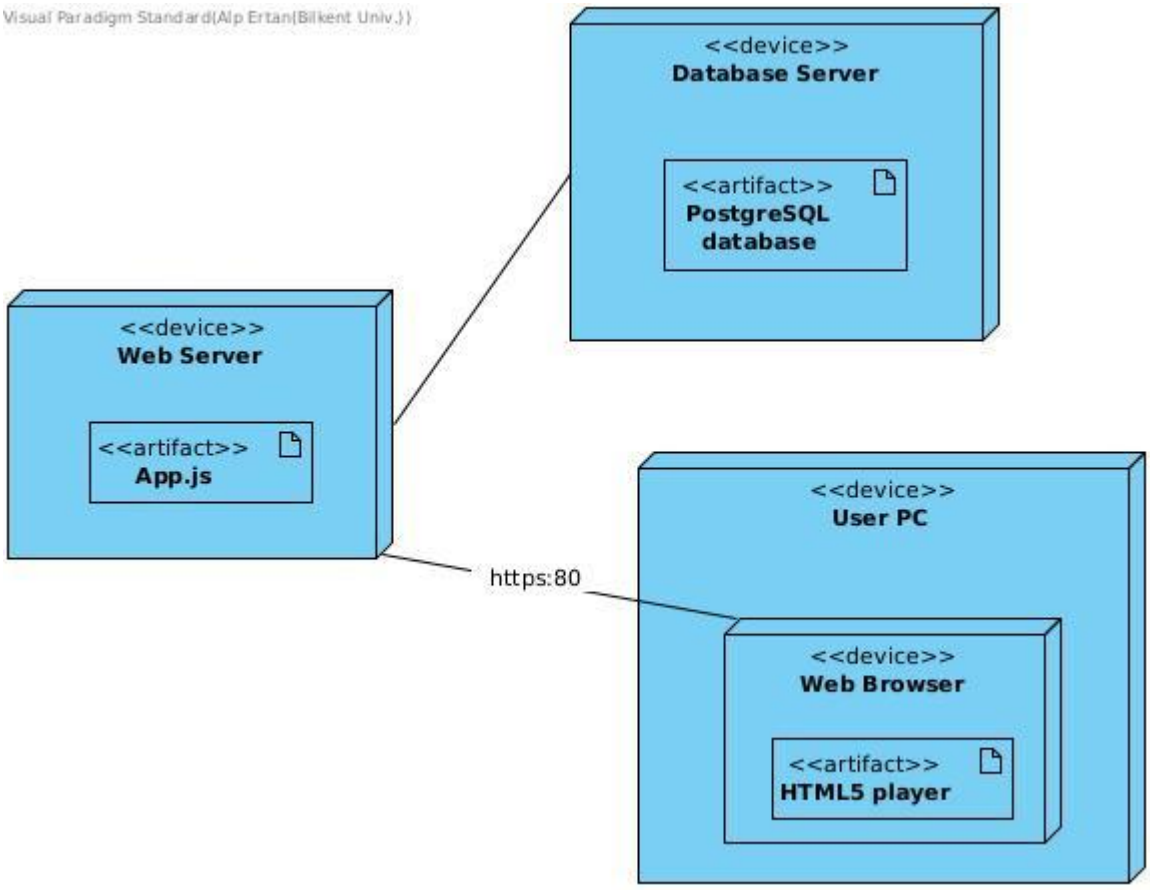
Interface layer, represents the frontend (different pages of the application). Interface layer includes the controllers on that page such as buttons, inputs (text inputs, radio buttons, checkboxes, combo boxes, and so on) and event listeners associated with them. In short, interface

layer subsystems consist of the elements the users can interact with. Therefore, our non-functional requirement, user interface, is a part of this subsystem. Interface layer subsystems will be implemented via React framework.

Web server layer, represents the backend (data handling and processing layer). Web server layer consists of the manager classes that are responsible for the management of the data that is fetched from the frontend layer and provide a connection between the frontend and the database. Web server layer subsystems will be implemented via Spring Boot framework.

2.2 Hardware/Software Mapping

2.2.1 Deployment Diagram



Eramus++ will be implemented using React and Spring Boot framework. It will be a web application, so it requires a computer powerful enough to run a web browser. The application will be available on mobile phones as well as the UI of the application will be responsive. If the users are accessing through a computer, they will need a mouse and a keyboard to interact with the UI. In the development side, according to the internet, an Amazon Web Services EC2 instance can support up to 1000 concurrent users. Properties for the minimum instance are 61 GB RAM, 8

vCPUs and 2 TB SSD [1]. We think that our server's specifications should be equivalent to these ones since we will also have up to 1000 concurrent students logged in to our system.

2.3 Persistent Data Management

PostgreSQL is a free and open-source relational database management system. We chose to use PostgreSQL because Spring Boot (back-end framework we use) supports PostgreSQL Each object group in our system will have its own table in the database. For example, user object will have its username, password, email, phone etc. in its own table, and announcement objects will have its own table where the title, content etc. will be written there. This way, SQL queries will be called onto this tables to retrieve these data. These tables will be edited by the users while using the website. While the users are interacting with the page, proper requests (GET, POST, DELETE OR PUT) will be sent to the database so the contents and structure of the tables will be modified. We will be using promise based HTTP client called Axios[2]. We chose this library because it works easily with React.

2.4 Access Control and Security

Erasmus++ is designed to be simultaneously used by many users with different levels of privileges. This is why we needed to make sure that users can only access parts of the site that are relevant for them. This requires authentication of users on login and a continuous way of sending and receiving requests securely. This is why JSON Web Tokens are used in our application. JWT defines a way to securely transmit information between parties as a JSON object. Each time a user is logged in to our application, the user is given a JW token and this token is again required by the servers every time the user wants to communicate with the application. If the user sends the key with the request they have, we know that the request is valid, and we also know who the user is. This whole process makes our system secure. Additionally, PostgreSQL provides several encryption methods to use when storing data. We tried to make sure that information that is sent by the user is securely processed and stored.

System Access Matrix:

Features	Candidate	Coordinator	Administrators	International Students Office	Host University Representative	Faculty Board
Register		●	●	●	●	●
Log In	●	●	●	●	●	●
Apply to Erasmus	●					

Approve Documents		●				●
Submit Documents	●				●	
Post Announcement		●	●	●	●	
Remove Announcement		●	●	●	●	
ToDo List		●				
Roadmap	●	●				

2.5 Boundary Conditions

2.5.1 Initialization

Erasmus++ will not require any downloads or extra software to access it. It will be a web-based application which requires internet connection only. This makes it available on every computer that has a browser to connect to the internet. When the web-based application is launched via an internet browser, App.js component of the React application will be executed since it contains all other components. If the user has logged in before and did not clear the cookies, the main page specific for that user will be loaded by React Router. Otherwise, the log-in page will be loaded by React Router in order for them to log-in. Erasmus++ users will have their accounts created for them already and saved in the database. Users will be able to log in to the application with their Bilkent mail and password.

2.5.2 Termination

The app can be exited via the log out button or by closing the browser window on which the app is being run. After closing the window, the user session will terminate. The safe termination makes sure that all the changes related with backend calls and database are applied. However, if the user is making local changes, such as filling an application form but not submitting it, the user won't be able to continue to make local changes from where he/she has left and has to start from the beginning since these local updates will not be saved into the database

until they are submitted via backend call. In other words, any action that does not make any backend calls will not be saved after termination where actions that do backend calls will be applied to the database when the application is terminated safely.

2.5.3 Failure

If there is a connection issue the user cannot connect to the application since it all runs on the web. In case the connection is lost while the application is running, the user will be asked to refresh the page to try reconnecting to the server. If the user cannot reconnect, the unsaved changes will be lost since they aren't written to the server via backend calls. If there is a server related failure, the caller function that makes calls to the backend will notice that the call is failed and thus notify the user that the submission/application process could not be completed. In other words, the web-server layer will be communicating with the interface layer when there is a problem regarding getting responses from the database. By this communication, the user will acknowledge that the process has not completed as expected, thus he/she has to try again until the server is operational again.

3.0 Low-Level Design

3.1. Object Design Trade-offs

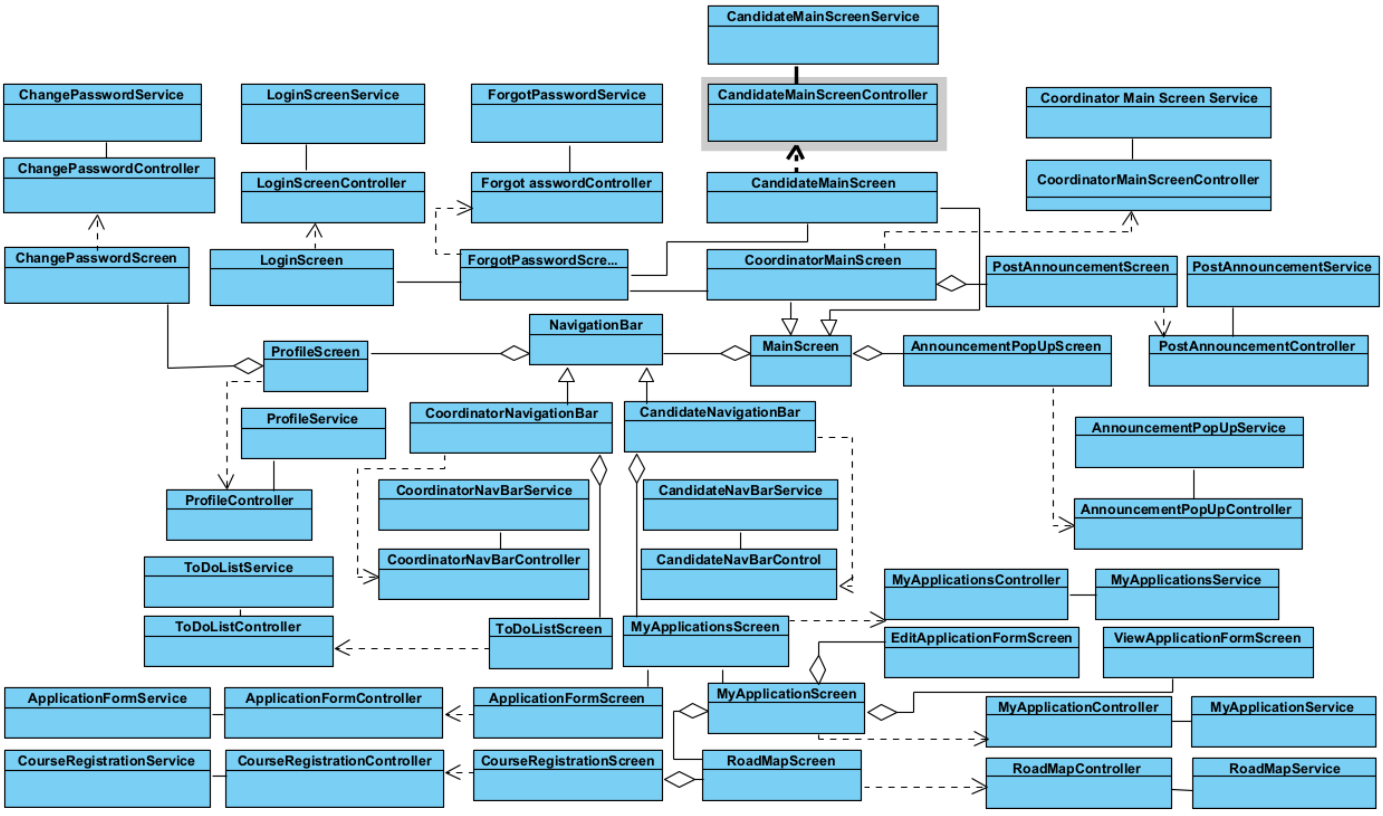
Maintainability vs Performance: By dividing the Erasmus++ into several layers, it has become much easier to maintain and implement the software. However, having multiple layers may cause minor loss in the performance which can be tolerated via the major maintainability advantages that come with layering.

Memory vs Performance: Since the object-oriented approach suggests creating objects for almost each component, it is expected to have vast memory use of the program. By using the AWS(Amazon Web Services) cloud platform, memory usage can be maintained by the auto-scaling utility and a balance between the memory and the performance is provided.

Security vs Usability: Erasmus++ uses cookies and saves them into the user's local storage in order to distinguish user types and prevent users from to log-in each time they open the web application. Even though this utility tremendously increases the usability of the application, security leaks might occur due to the risk of storing cookies at the local storage which is less secure than the database itself.

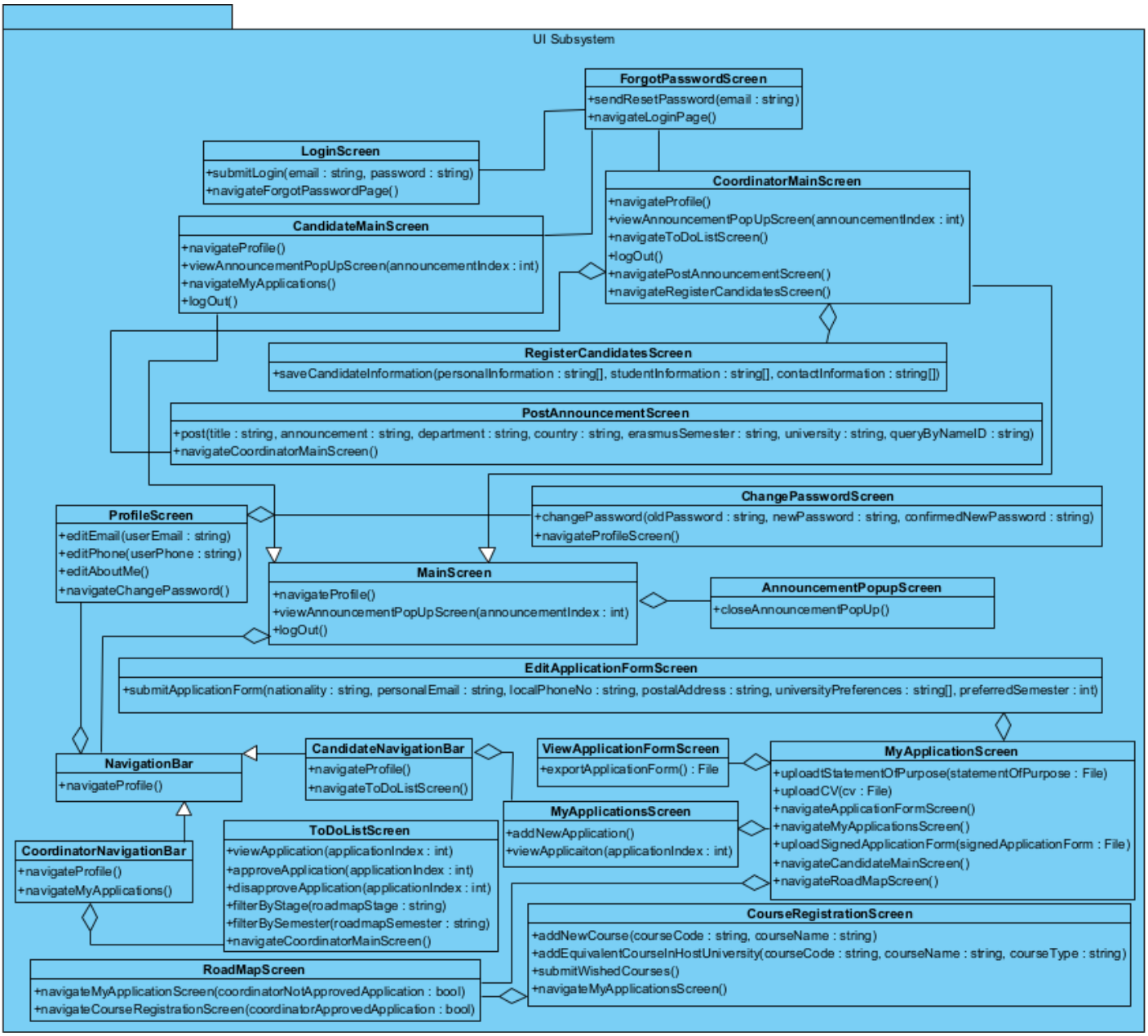
Functionality vs Usability: Erasmus++ have many actors that will be using the software. Providing different functionalities for coordinators and candidates decreases the usability for the users since there is massive function-load.

3.2 Final Object Design



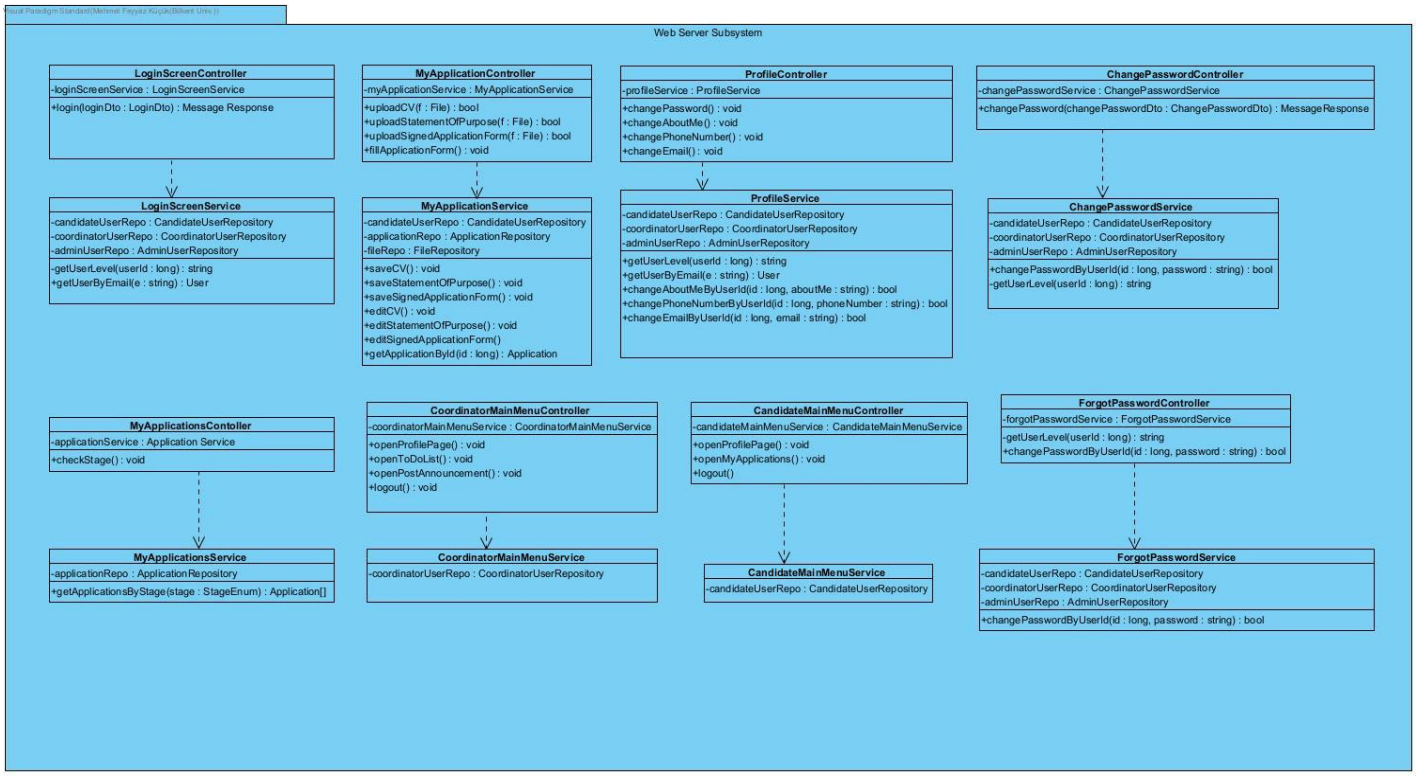
3.3 Layers

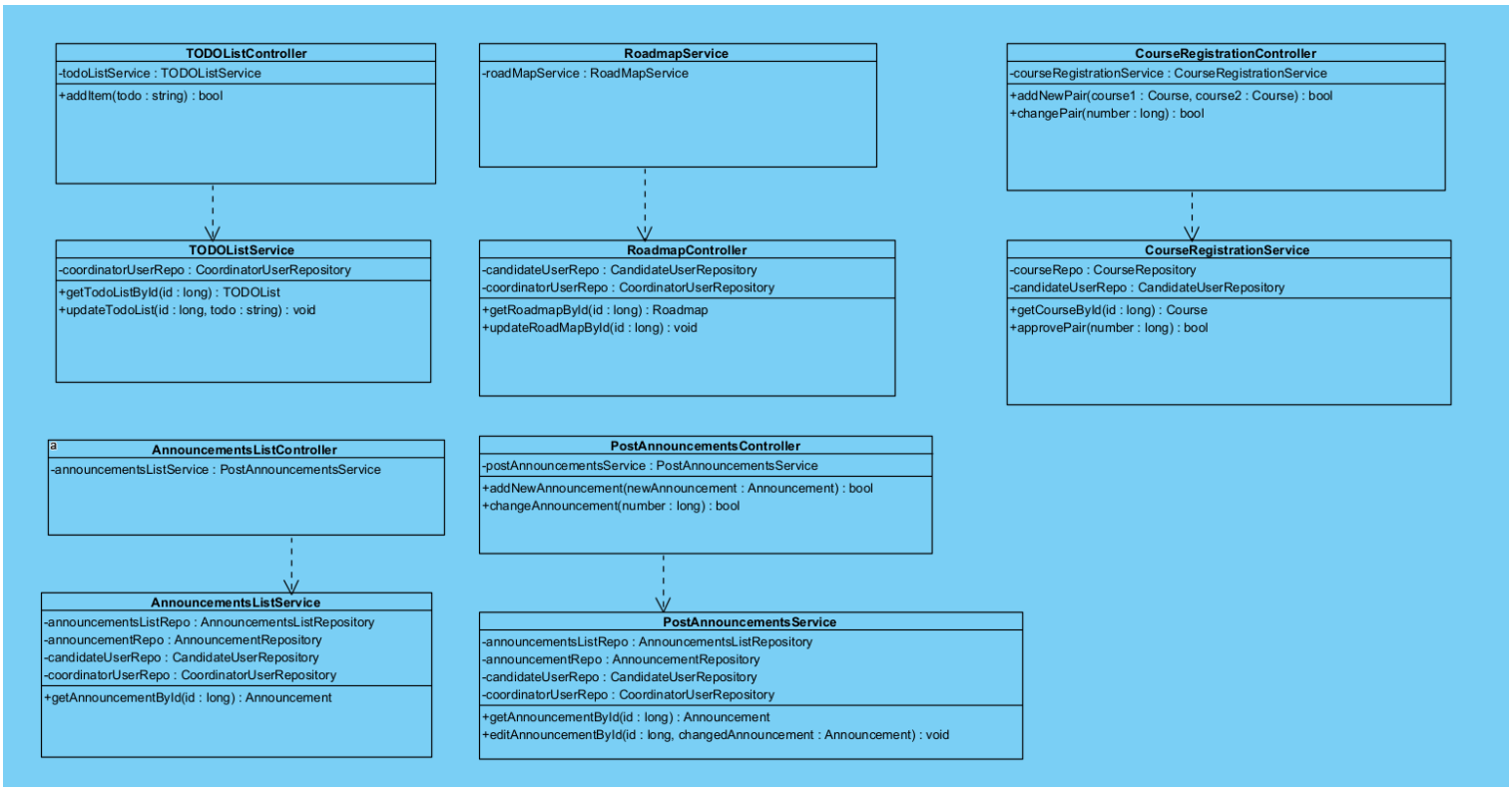
3.3.1 User Interface Layer



This layer functions as a boundary (view in MVC) between the system and users. It contains elements that the users can interact with (such as clicking buttons, navigating between pages, filling out input fields). We separated the objects considering the different screens that the users will see. For example, the users can either login or go to the forgot password screen from the login screen. This way, handling all the behaviors will be easier.

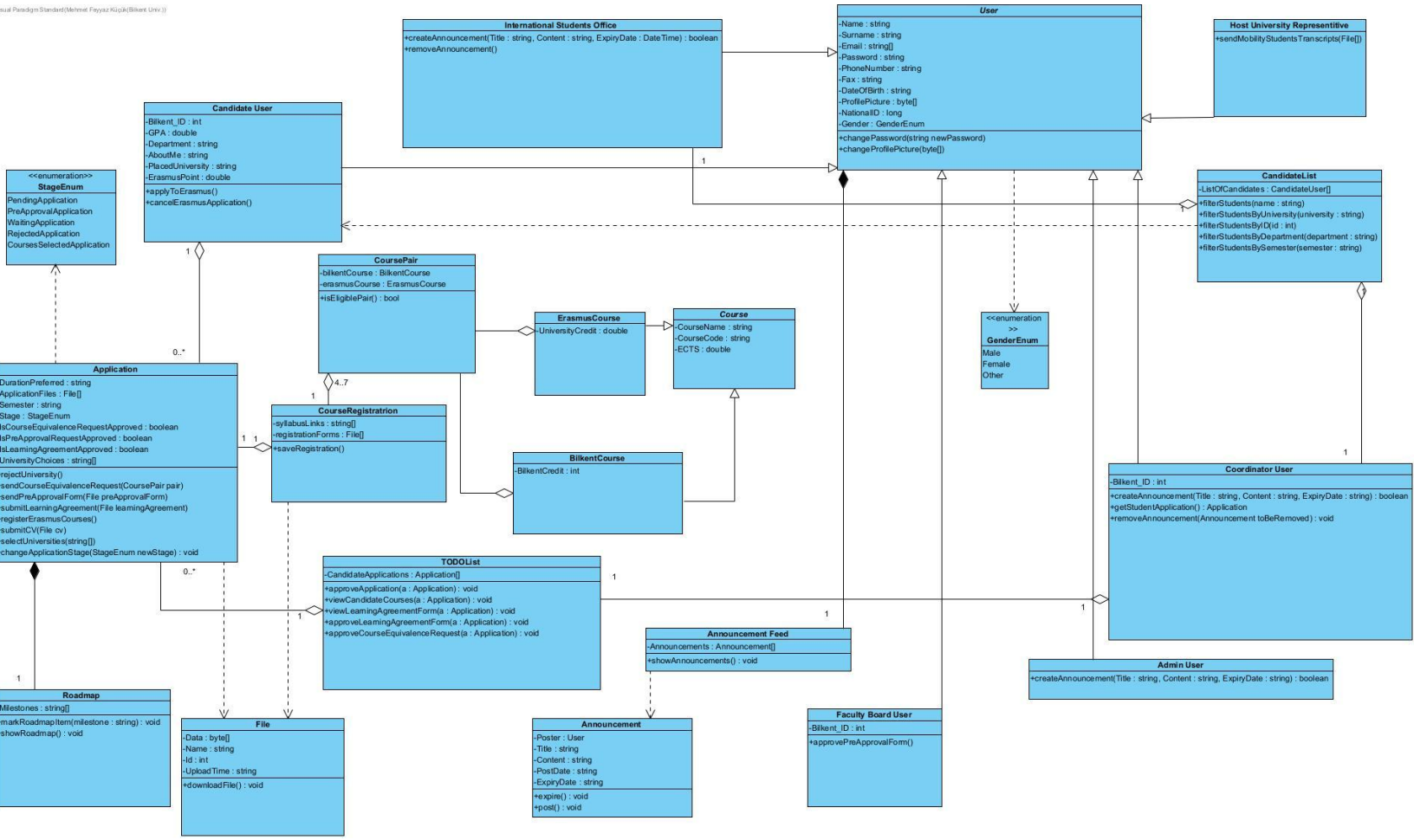
3.3.2 Web Server Layer

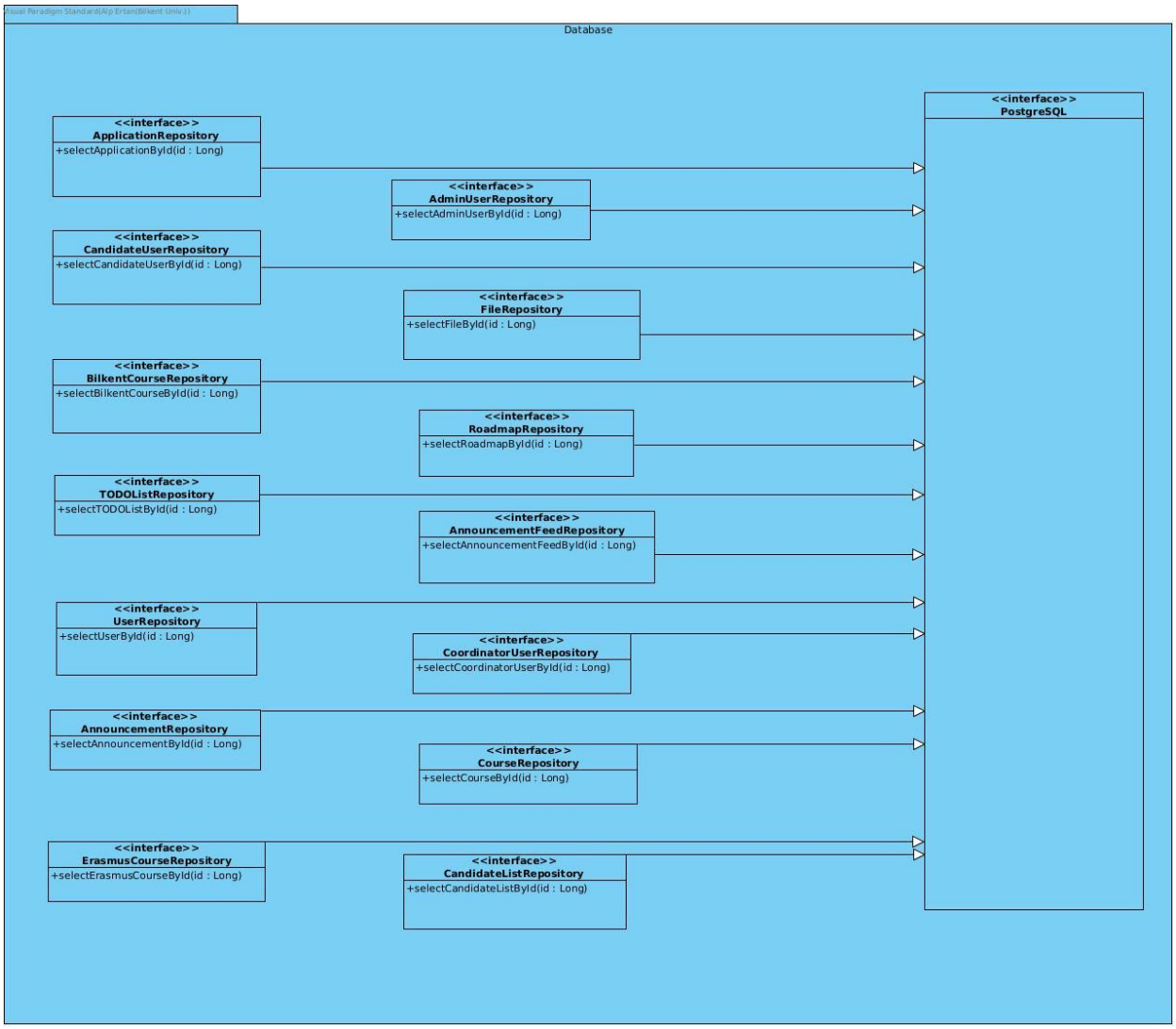




This layer functions as the control (controller in MVC) between the boundary and entity (and database) objects. When the users take action on the boundary object, Web Server Layer is notified and responds to it. Also, if any data is required (login, changing password, etc.), Web Server Layer sends a request to the Data Management Layer and fetches the data needed.

3.3.3 Data Management Layer





This layer contains two different subsystems that communicate with Web Server Server. Those are the Entity and the Database subsystems. The Entity subsystem contains the model classes that contain data and do certain operations with them. The Database subsystem contains classes that act as packages for the model classes that will be stored in the database. The Data Management .Layer will respond to the Web Server Layer by communicating with the appropriate interfaces (if user information is needed, one of the UserRepository classes will be used).

3.4 Packages

3.4.1 Implementation Packages

3.4.1.1 Authorization

Package containing password and login classes.

3.4.1.2 Candidate Package

Package that contains roadmap, course registration and my applications.

3.4.1.3 Coordinator Package

Package containing to do list management of coordinators, posting announcement management , candidate registration management.

3.4.1.4 Security

Package that contains elements that are related to security components.

3.4.1.5 Repositories

Package containing database management & processing objects.

3.4.1.6 Models (Entities)

Package containing all model classes.

3.4.1.7 Profile Management

Package that contains profile settings and management of profile information.

3.4.2 External Library Packages

3.4.2.1 springframework.postgresql

This package is for the connection between our database (postgresql) and our backend (Spring Boot)

3.4.2.2 springframework.data.jpa

This package enables our database (spring boot) to become object oriented.

3.4.2.3 springframework.jdbc

This is for making database queries using java language.

3.4.2.4 springframework.security

This package will be used for password hashing, sessions, etc.

3.4.2.5 io.jsonwebtoken

This is for integrating JWT to our system. This will be used for user authentication, cookies, sessions, etc.

3.4.2.6 springframework.thymeleaf

This package allows us to use html templates to use while developing the app, this will improve our productivity

3.4.2.7 springframework.restdocs

This package is for documenting the actions made in our REST APIs.

3.4.2.8 React.react

This package allows us to use three main parts of React to construct the UI of our application:

- react.Components: allows us to create custom DOM elements.
- react.Router: navigation between pages
- react.Services: validation

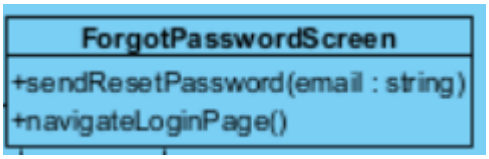
3.4.2.9 React.react-dom

This package allows us to go out of the React model if needed.

3.5 Class Interfaces

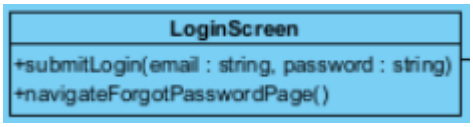
3.5.1 User Interface Layer Class Interfaces

3.5.1.1 Forget Password Screen



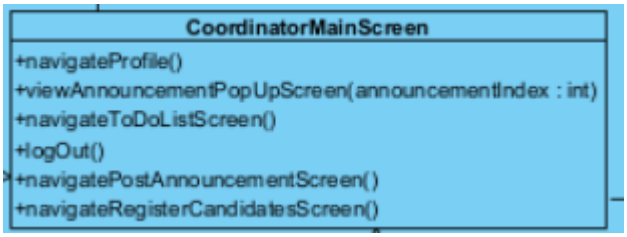
sendResetPassword(email: string): this method sends reset password request with email address.
navigateLoginPage(): this method returns back to the login page.

3.5.1.2 Login Screen



submitLogin(email: string, password: string): this method sends login request with entered email and password.
navigateForgotPasswordPage(): this method navigates to the forgot password page.

3.5.1.3 Coordinator Main Screen

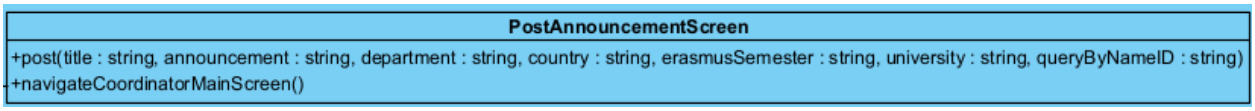


navigateProfile(): this method navigates to the profile page.
viewAnnouncemenetsPopUpScreen(announcementIndex: int): this method displays the announcement that is selected at a given index.
navigateToDoListScreen(): this method navigates to the to-do list page.
logOut(): this method performs a log-out operation for the current user.
navigatePostAnnouncementScreen(): this method navigates to the post announcements page.
navigateRegisterCandidatesScreen(): this method navigates to the register candidate screen.

3.5.1.4 Register Candidates Screen

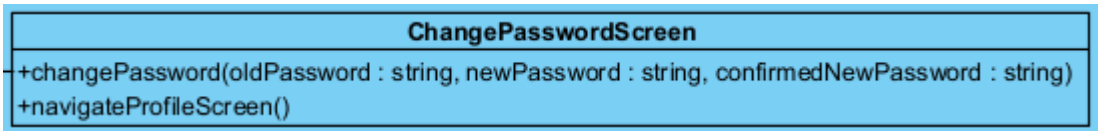
saveCandidateInformation(personalInformation : string[], studentInformation : string[], contactInformation : string[]): Saves candidate information.

3.5.1.5 Post Announcement Screen



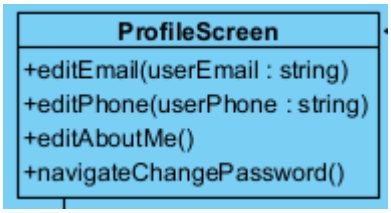
post(title : string, announcement : string, department : string, country : string, erasmusSemester : string, university : string, queryByNameID : string): Posts a new announcement.
navigateCoordinatorMainScreen(): this method navigates back to the coordinator main screen.

3.5.1.6 Change Password Screen



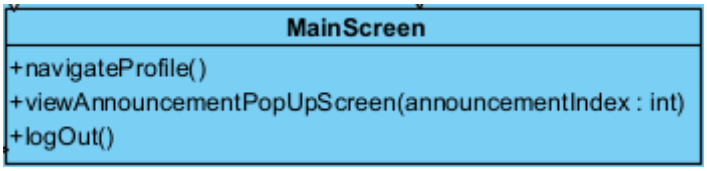
changePassword(oldPassword : string, newPassword : string, confirmedNewPassword : string): This method replaces the old password variable's information with another string.
navigateProfileScreen(): This method navigates the user back to the profile screen.

3.5.1.7 Profile Screen



editEmail(userEmail : string): This method changes the email address of the user by taking the email address as a string parameter.
editPhone(userPhone : string): This method edits the user phone number by taking it as a string parameter.
editAboutMe(): This method edits the about me information of the user.
navigateChangePassword(): This method navigates to the change password page.

3.5.1.8 Main Screen

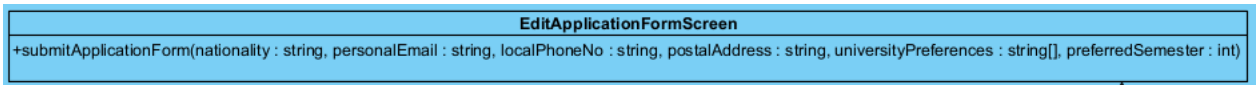


navigateProfile(): This method navigates user to their profile page.
viewAnnouncementPopUpScreen(announcementIndex : int): This method puts an announcement popping up to the screen.
logOut(): This method logs the user out of the system.

3.5.1.9 Announcement Pop Up Screen Candidates Screen

closeAnnouncementPopUp(): This method is called when the user wants to close a pop-up.

3.5.1.10 Edit Application Form Screen



submitApplicationForm(nationality:string, personalEmail:string, localPhoneNo:string, postalAddress:string, universityPreferences:string[], preferredSemester:int): This method will submit the application form with the given credentials initialized.

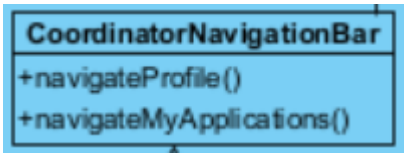
3.5.1.11 Navigation Bar

navigateProfile(): This method navigates to the profile page.

3.5.1.12 Candidate Navigation Bar

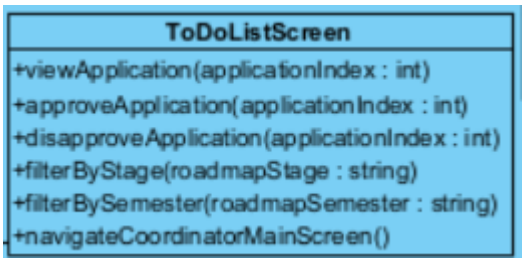
navigateProfile(): This method navigates to the profile screen.
navigateToDoListScreen(): This method navigates to the ToDo List screen.

3.5.1.13 Coordinator Navigation Bar



navigateProfile(): This method navigates to the profile page.
navigateMyApplications(): This method navigates to the my applications page.

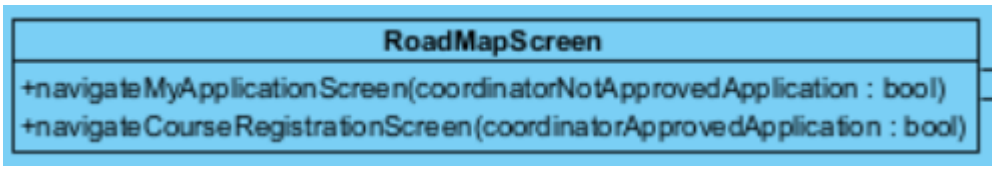
3.5.1.14 To- Do List Screen



viewApplication(applicationIndex: int): This method views application at the given index.
approveApplication(applicationIndex: int): This method approves application at the given index.
disapproveApplication(applicationIndex: int): This method disapproves application at the given index.
filterByStage(roadmapStage: string): This method filters the stages at the roadmap by given stage.
filterBySemester(roadmapSemester: string): This method filters the semesters at the roadmap by given semester.

navigateCoordinatorMainScreen(): This method navigates to the coordinator main screen

3.5.1.15 RoadMapScreen



navigateMyApplication(coordinatorNotApprovedApplication: bool): This method will take the user to the myApplication screen.
navigateCourseRegistrationScreen(coordinatorApprovedApplication: bool): This method will take the user to the navigateCourseRegistration screen.

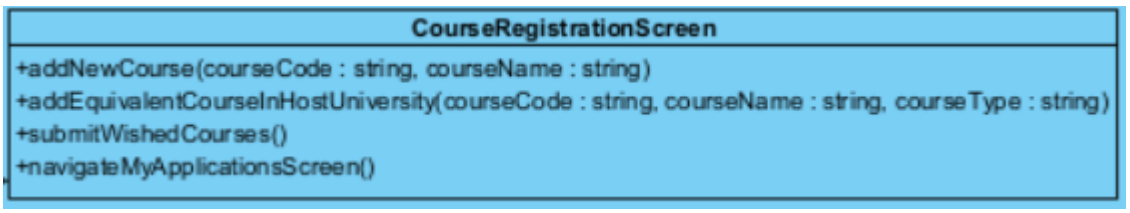
3.5.1.16 View Application Form Screen

exportApplicationForm(): This method exports application form as a pdf for the user's computer.

3.5.1.17 My Applications Screen

a**ddNewApplication():** This method will be called when the application info is entered and will add a new application to the user.
viewApplication(applicationIndex: int): This method will be called when the user clicks at the application and will provide application information.

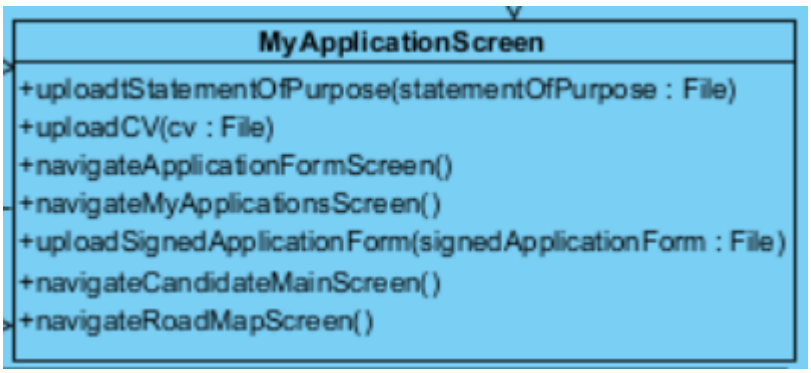
3.5.1.18 Course Registration Screen



addNewCourse(courseCode : string, courseName : string): This method adds a new course to the registration screen.

addEquivalentCourseInHostUniversity(courseCode: string, courseName: string, courseType: string): This method adds a course that is in the host university into the registration screen.
submitWishedCourses(): This method submits wanted courses to the system.
navigateMyApplicationsScreen(): This method navigates back to the applications screen.

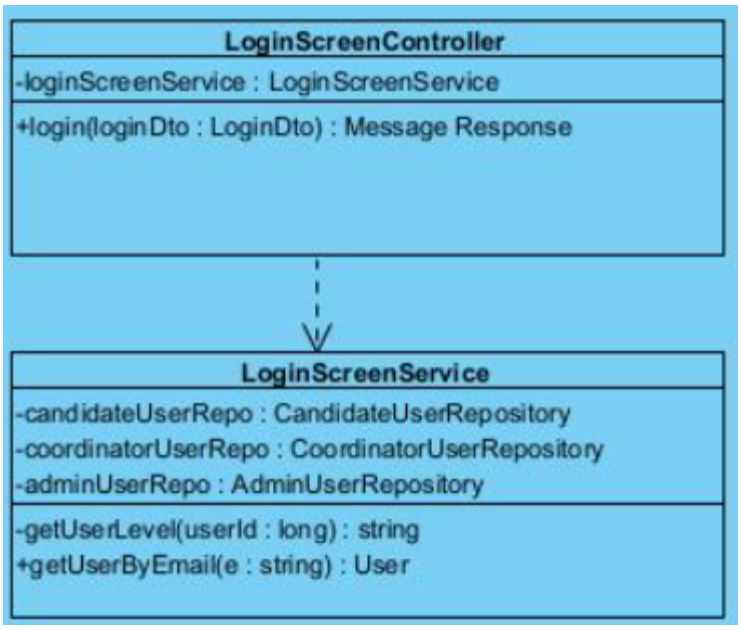
3.5.1.19 My Applications Screen



uploadStatementOfPurpose(statementOfPurpose: File): This method will store the statement of purpose in the application.
uploadCV(cv: File): This method is for uploading CV to the system as a pdf file.
navigateApplication(): This method is for navigating to the application screen.
navigateMyApplicationsScreen(): This method is for navigating back to the my applications screen.
uploadSignedApplicationForm(signedApplicationForm: File): This method is for uploading signed application form as a file to the system.
navigateCandidateMainScreen(): This method navigates back to the main screen.
navigateRoadMapScreen(): This method navigates back to the roadmap screen.

3.5.2 Web Server Layer Class Interfaces

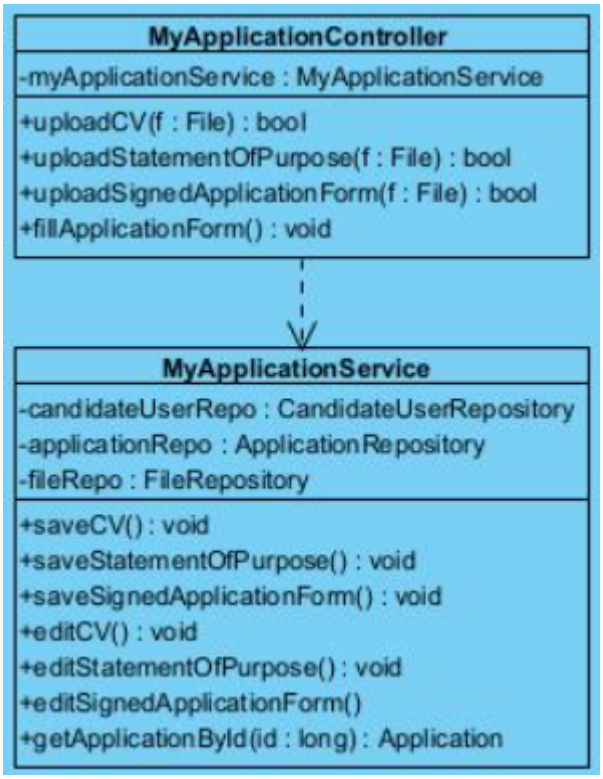
3.5.2.1 Login Screen Controller & Service



LoginScreenService:
candidateUserRepo: Repository of candidate user.
coordinatorUserRepo: Repository of coordinator user.
adminUserRepo: Repository of admin user.
getUserLevel(userId : long): Returns the user level in the actor hierarchy.
getUserByEmail(e : string): Returns the user by email.

LoginScreenController:
loginScreenService: The service attribute of the login screen controller.
login(logiDto : LoginDto): The method to message response.

3.5.2.2 My Application Controller & Service

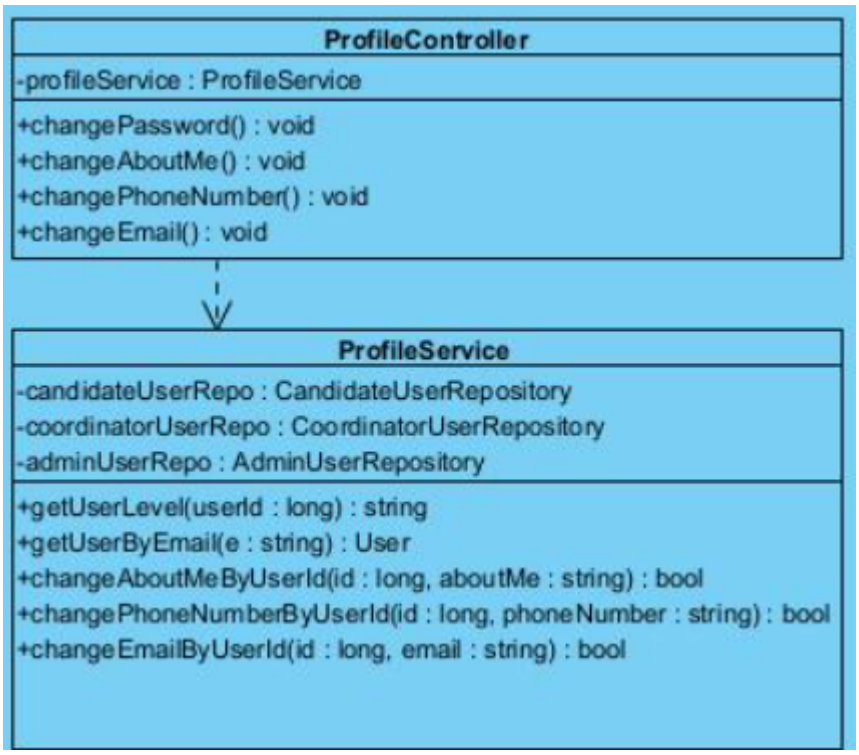


MyApplicationService:
candidateUserRepo: Repository of candidate user.
applicationRepo: Repository of application user.
fileRepo: Repository of file.
saveCV(): The method to save the CV.
saveStatementOfPurpose(): The method that saves the statement of purpose.
saveSignedApplicationForm(): The method that saves the signed application form.
editCV(): The method that is called to edit the CV.
editStatementOfPurpose(): The method that is called to edit the statement of purpose.
editSignedApplicationForm(): The method that is called to edit the signed application form.
getApplicationById(id : long): The method to get the application by id (that returns an Application when called).

MyApplicationController:
myApplicationService: The attribute for my application (for candidate) service.
uploadCV(f: File): This method checks if the Upload CV button is pressed or not; and returns a boolean value.
uploadStatementOfPurpose(f : File): This method checks if the Upload Statement of Purpose button is pressed or not; and returns a boolean value.
uploadSignedApplicationForm(f : File): This method checks if the Upload Signed Application Form button is pressed or not; and returns a boolean value.

fillApplicationForm(): The method to fill in the application.

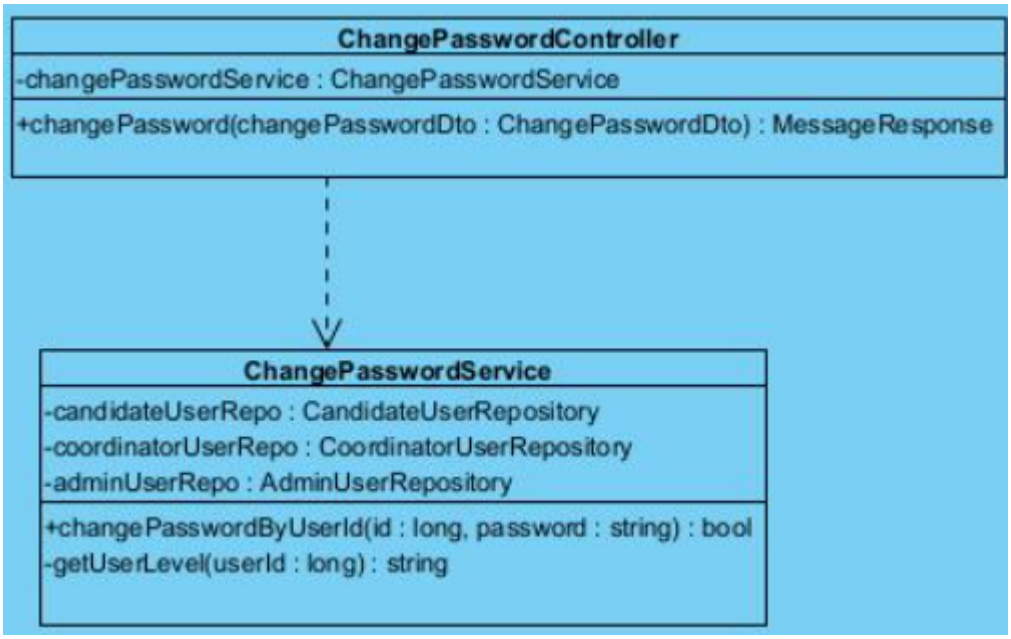
3.5.2.3 Profile Controller & Service



ProfileService:
candidateUserRepo: Repository of candidate user.
coordinatorUserRepo: Repository of coordinator user.
adminUserRepo: Repository of admin user.
getUserLevel(userId: long): Returns the user level as a string.
getUserByEmail(e: string): Returns user model by the id.
changeAboutMeByUserId(id:long, aboutMe:string): Changes the about me to the specified string of the user by the id.
changeEmailByUserId(id:long, email:string): Changes the email to the specified string of the user by the id.

ProfileController:
changePassword(): Changes the password of the user.
changeAboutMe(): Changes the about me of the user.
changePhoneNumber(): Changes the phone number of the user.
changeEmail(): void : Changes the email of the user.

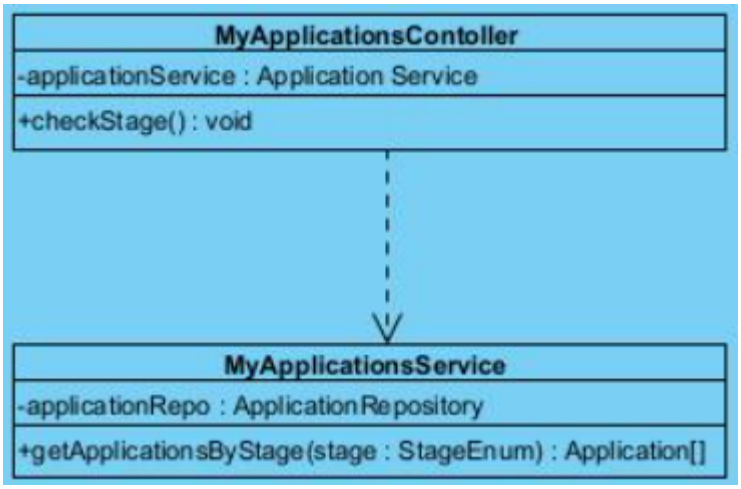
3.5.2.4 Change Password Controller & Service



ChangePasswordService:
candidateUserRepo: Repo of candidate user
coordinatorUserRepo: Repo of coordinator user
adminUserRepo: Repo of admin user
changePasswordByUserId(id : long, password : string): This method changes the password of a user.
getUserLevel(userId : long): This method returns the level of the user (coordinator, candidate etc.).

ChangePasswordController:
changePasswordService: Service for change password.
changePassword(changePasswordDto : ChangePasswordDto): Dto for change password.

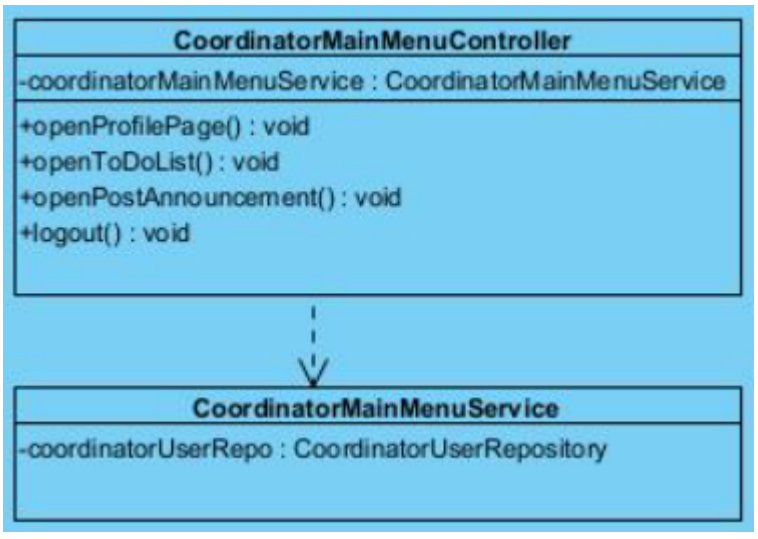
3.5.2.5 MyApplications Controller & Service



MyApplicationService:
applicationRepo: Repo for applications.
getApplicationsByStage(stage : StageEnum): Returns the stage of the current application.

MyApplicationsController:
applicationService: service for application.
checkStage(): checks the current stage.

3.5.2.6 Coordinator Main Menu Controller & Service

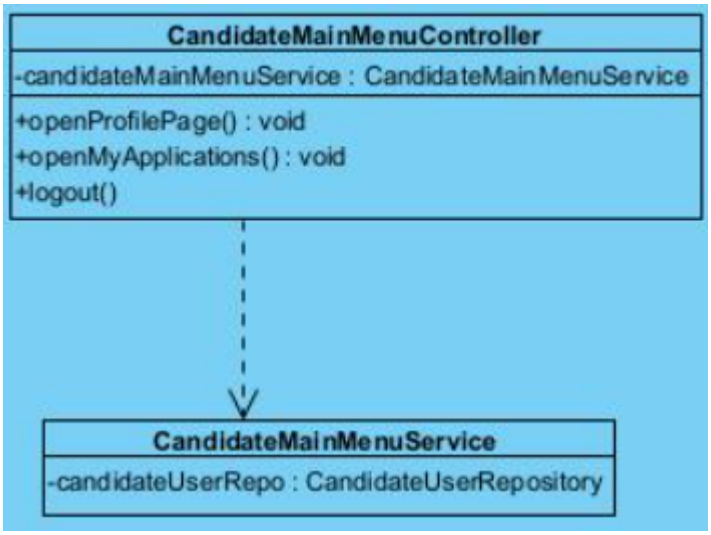


CoordinatorMainMenuService:
coordinatorUserRepo: Repository for coordinator user.

CoordinatorMainMenuController:
coordinatorMainMenuService: service for coordinator main menu.

openProfilePage(): This method opens profile page.
openToDoList(): This method to do list page.
openPostAnnouncement(): This method opens post announcements page.
logout(): This methods logs out from the current user.

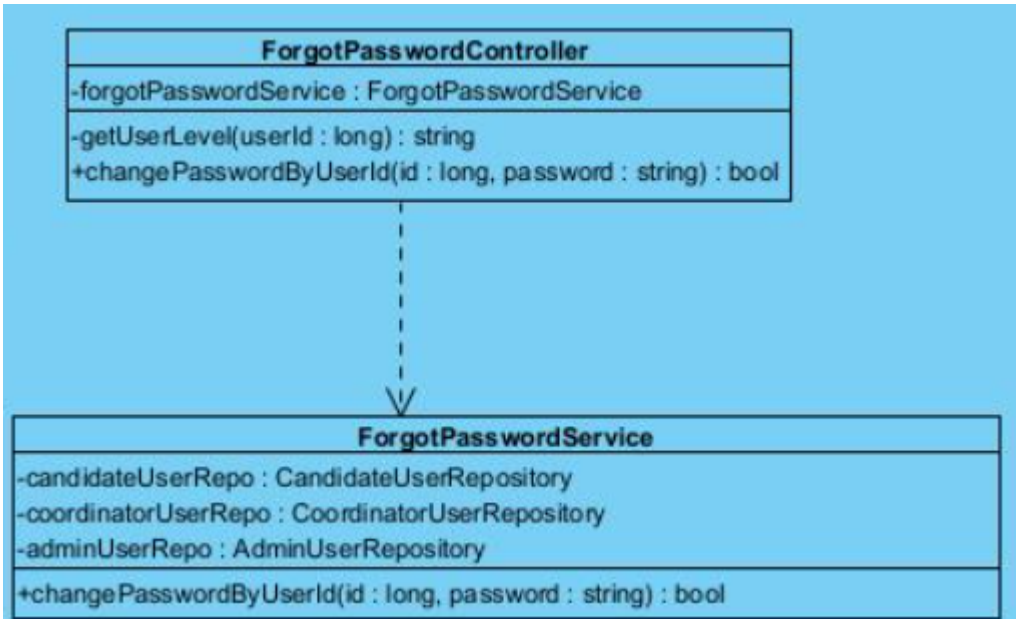
3.5.2.7 Candidate Main Menu Controller & Service



Candidate Main Menu Service:
candidateUserRepo: Repository for candidate user.

Candidate Main Menu Controller:
candidateMainMenuService: service for candidate main menu.
openProfilePage(): This method opens the profile page.
openMyApplications(): This method opens my applications page.
logout(): This method logs the user out from the system.

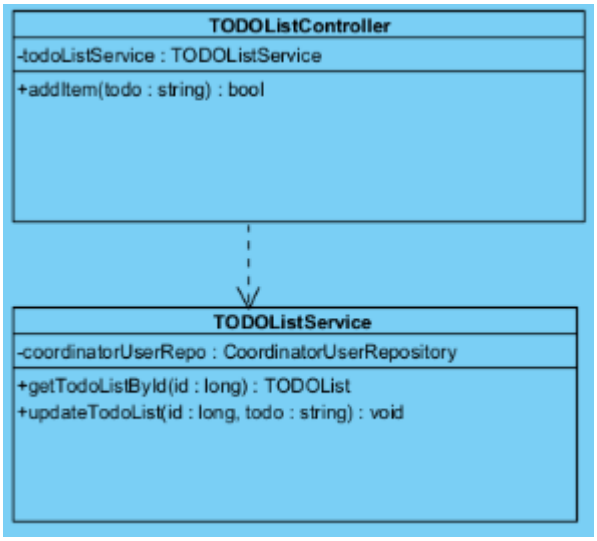
3.5.2.8 Forgot Password Controller & Service



Forgot Password Service:
candidateUserRepo: Repo for candidate user.
coordinatorUserRepo: Repo for coordinator user.
adminUserRepo: Repo for admin user.
changePasswordById(id : long, password : string): This method is for changing password of a user.

ForgotPasswordController:
forgotPasswordService: Service for forgot password.
getUserLevel(userId : long): Returns the level of the user.
changePasswordByUserId(id : long, password : string): Changes the password of an user.

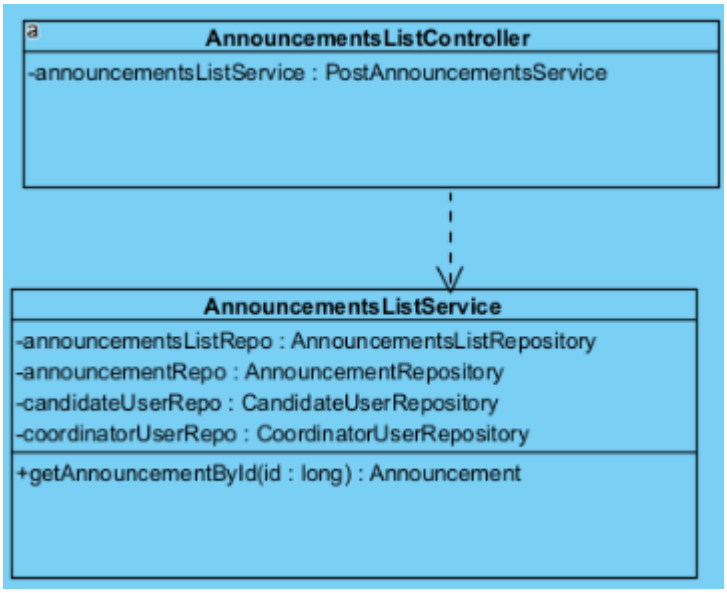
3.5.2.9 TODO List Controller & Service



TODOListService:
coordinatorUserRepo: Repository for coordinator user.
getTodoListById(id:long): Returns the todo list of the user with specified id.
updateTodoList(id:long, todo:string): Updates todo list of the user with specified string.

TODOListController:
addItem(todo:string): Adds item to the todo list using the todo list service.

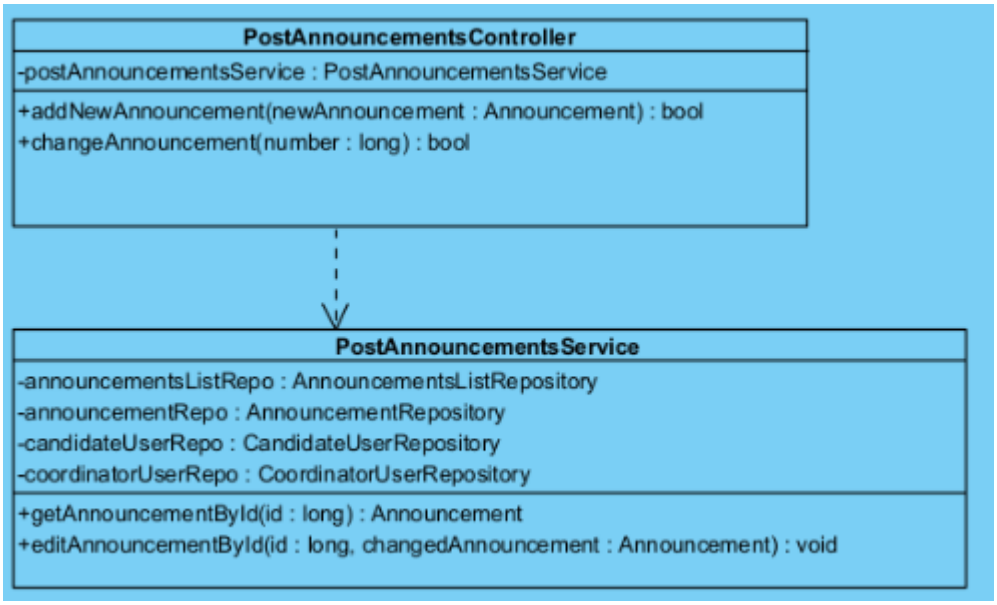
3.5.2.10 Announcements List Controller & Service



AnnouncementsListService:
announcementsListRepo: Repository for announcement list
announcementsRepo: Repository for announcement.
candidateUserRepo: Repository for candidate user.
coordinatorUserRepo: Repository for coordinator user.
getAnnouncementById(id: long): This method returns the announcement at the given ID.

AnnouncementsListController:
announcementsListService: PostAnnouncementsService

3.5.2.11 Post Announcements Controller & Service

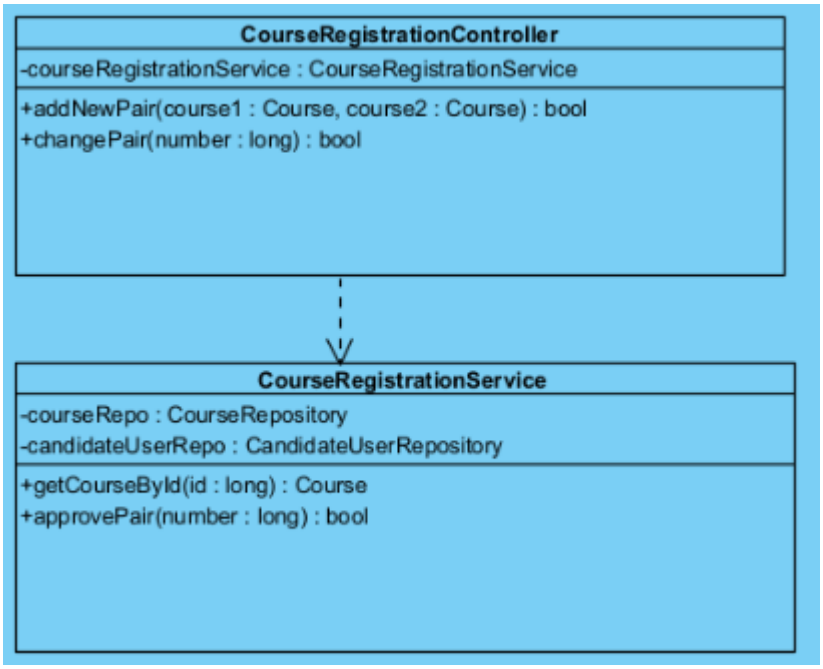


PostAnnouncementService:
announcementsListRepo: Repository for announcement lists.
announcementsRepo: Repository for a single announcement.
candidateUserRepo: Repository for candidate user.
coordinatorUserRepo: Repository for coordinator user.
getAnnouncementById(id:long): Announcement : Returns the announcement with specified id from the repository.
editAnnouncement(id: long, changedAnnouncement: Announcement): Edits the announcement with the specified announcement.

PostAnnouncementService:
addNewAnnouncement(newAnnouncement: Announcement): Uses service to add a new announcement.

changeAnnouncement(number: long): bool : Uses service to change the announcement with the specified id.

3.5.2.12 Course Registration Controller & Service



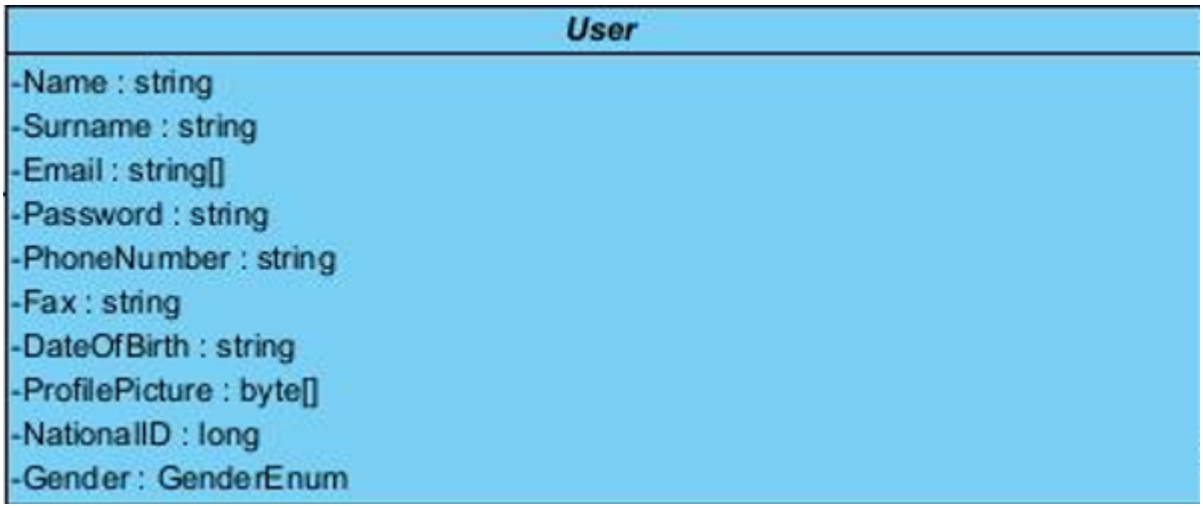
Course Registration Service:
courseRepo: Repo for courses.
candidateUserRepo: Repo for candidate users.
getCourseById(id : long): Returns course.
approvePair(number : long): Returns if the course pair is applicable or not.

Course Registration Controller:
courseRegistrationService: Service for course registration.
addNewPair(course1 : Course): Adds a new course pair to course registration page.
changePair(number : long): Changes a pair.

3.5.3 Data Management Layer Class Interfaces

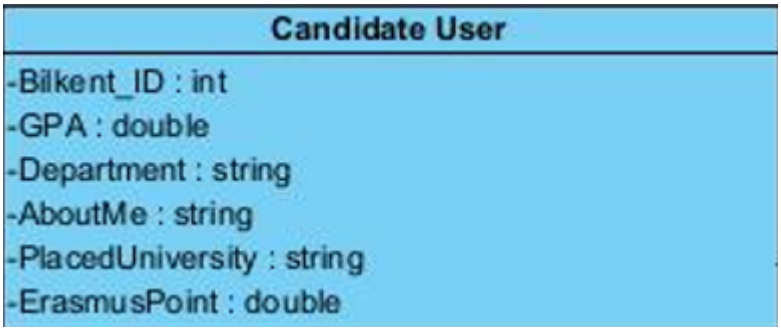
Note: Each model class listed here is omitted of their constructors and their getter and setter functions.

3.5.3.1 User



Name: Name of the user.
Surname: Surname of the user.
Email[]: Email array for possible emails of the user.
Password: Password for login of the user.
PhoneNumber: Communication number of the user.
Fax: Fax information of the user.
DateOfBirth: Birthday of the user.
ProfilePicture: Profile photo for the user.
NationalID: National ID number of the user.
Gender: Gender of the user.

3.5.3.2 CandidateUser



BilkentID: Bilkent ID of the candidate

GPA: GPA of the candidate
Department: Department of the candidate student
AboutMe: Additional about me info for candidate student
PlacedUniversity: Host university that the candidate will be going
ErasmusPoint: Erasmus point calculated by GPA and ENG101/102 grades

3.5.3.3 CoordinatorUser

Bilkent_ID: Bilkent ID of the coordinator

3.5.3.4 AdminUser

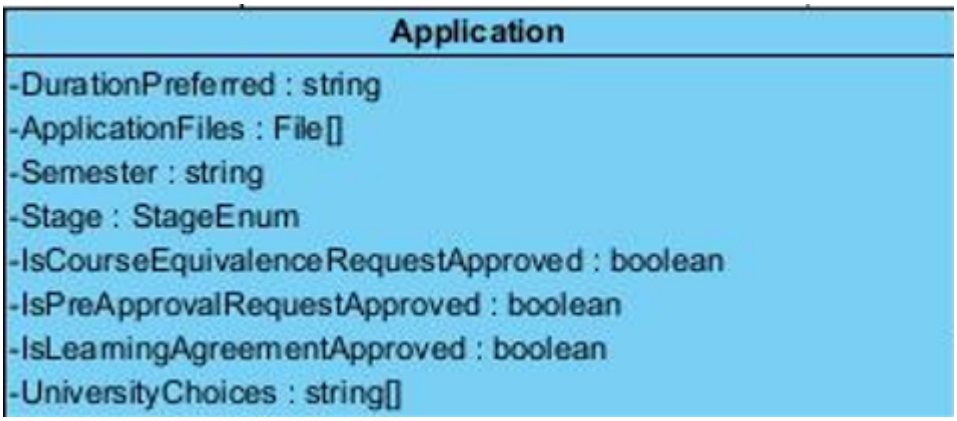
3.5.3.5 FacultyBoardUser

Bilkent_ID: Faculty Administration Board user's dedicated Bilkent ID.

3.5.3.6 HostUniversityRepresentative

3.5.3.7 InternationalStudentsOffice

3.5.3.8 Application

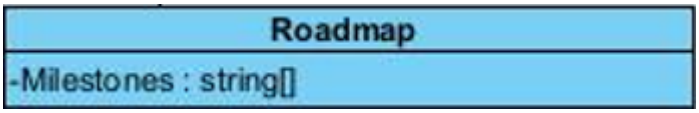


DurationPreferred: Erasmus duration that is preferred by the candidate
ApplicationFiles: array of files required for application
Semester: Semester that the Erasmus program will be performed
Stage: the stage of the application process that the student is at
IsCourseEquivalenceRequestApproved: Checks if the equivalence request is approved or not
IsPreApprovalRequestApproved: Checks if the pre-approval request is approved or not
IsLearningAgreementApproved: Checks if the learning agreement request is approved or not
UniversityChoices: University choices that are made by the candidate

3.5.3.9 File

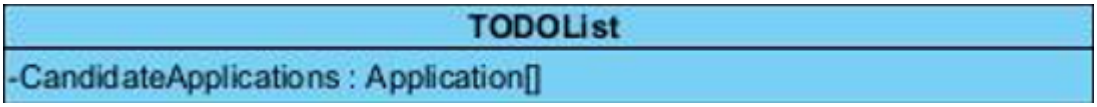
Data: The data of the file.
Name: Name of the file.
ID: File ID.
UploadTime: The information of the timestamp.

3.5.3.10 Roadmap



Milestones: The milestone elements in the roadmap.

3.5.3.11 TodoList



CandidateApplications: Array that holds the applications for the coordinator to view.

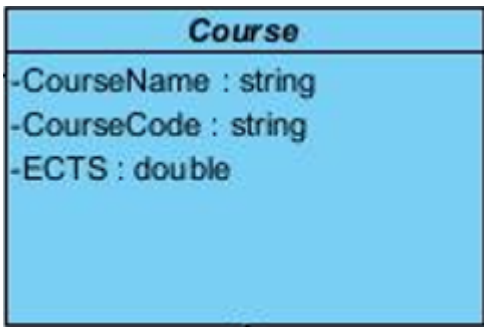
3.5.3.12 CourseRegistration

syllabusLinks: Course syllabus' links for the students to view.
registrationForms: Files that are needed for application.

3.5.3.13 CoursePair

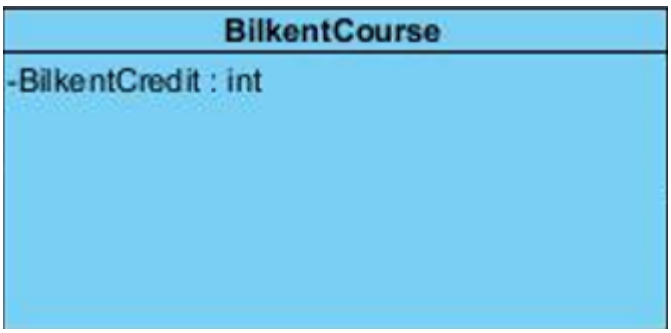
bilkentCourse: Bilkent course that will be counted for.
erasmusCourse: Host university course that will be counted as.

3.5.3.14 Course



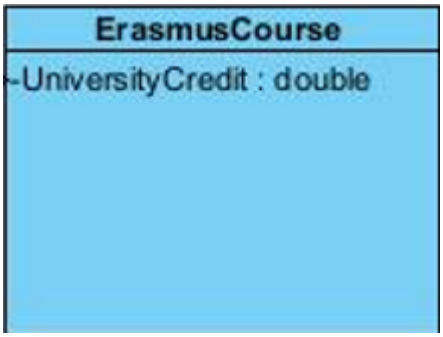
CourseName: Name of the course.
CourseCode: Course code.
ECTS: The credits for ECTS.

3.5.3.15 BilkentCourse



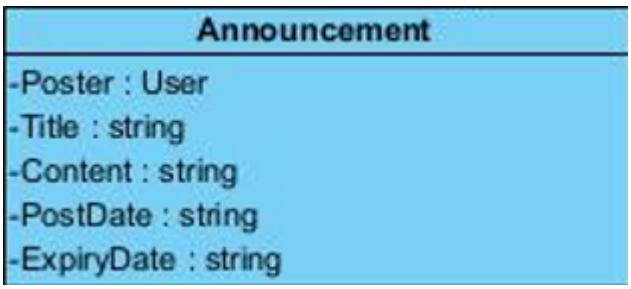
BilkentCredit: Bilkent credit for a course.

3.5.3.16 ErasmusCourse



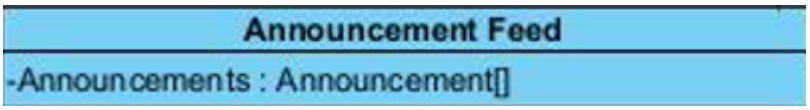
UniversityCredit: The applied university's credit for the course.

3.5.3.17 Announcement



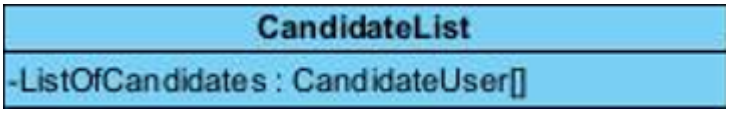
Poster: The user who posted the announcement.
Title: Title of the announcement.
Content: The text content of the announcement.
PostDate: The date the announcement will be posted.
ExpiryDate: The date until the announcement will be shown.

3.5.3.18 AnnouncementFeed



Announcements: array of announcements that are made

3.5.3.19 CandidateList



ListOfCandidates: array of candidate users that are visible to coordinators

4. References

[1]https://www.iguazio.com/docs/latest-release/cluster-mgmt/deployment/cloud/aws/aws-hw-spec/
[2]https://github.com/axios/axios