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

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

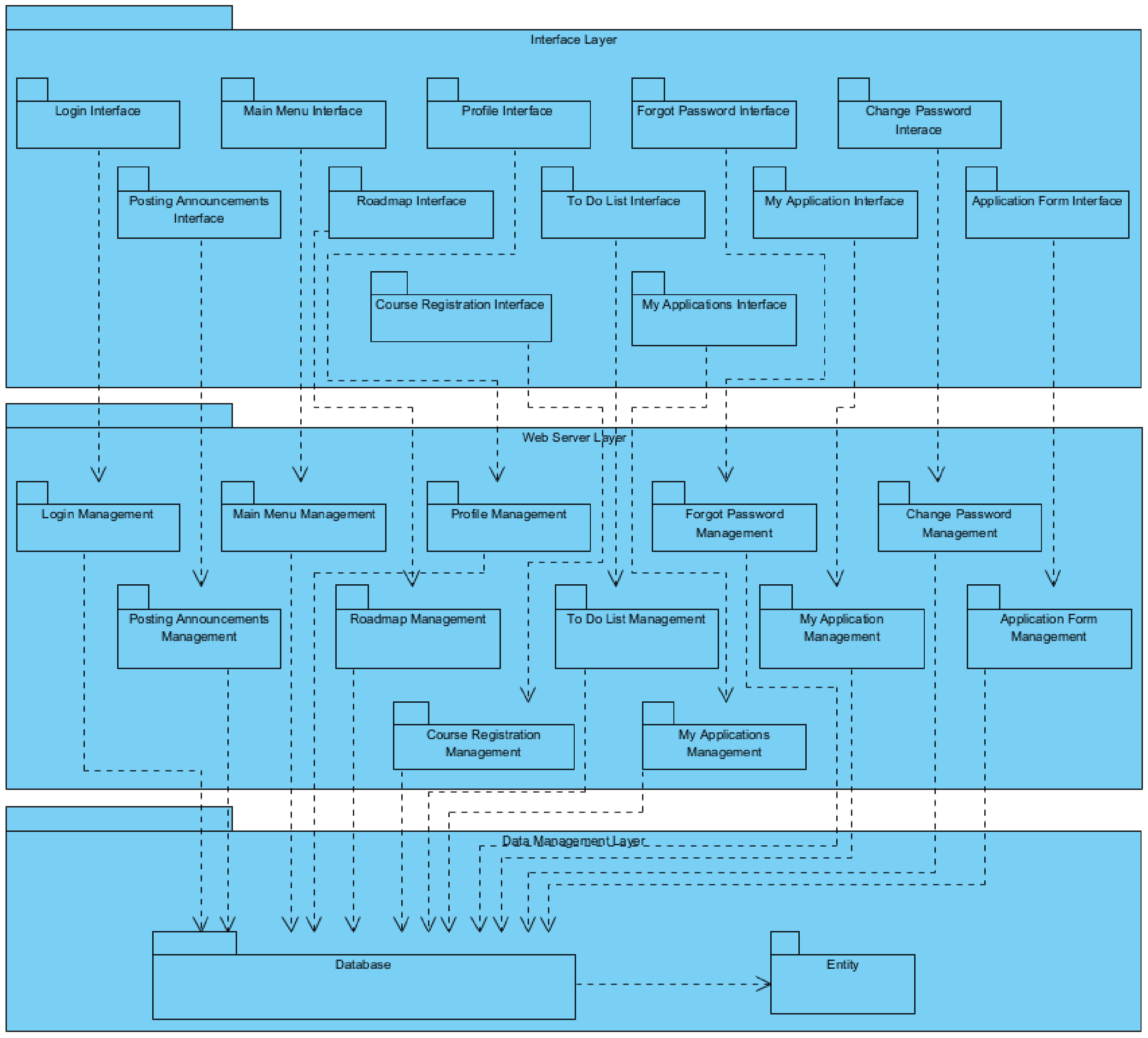

Interface Layer
Login Interface
Main Menu Interface
Profile Interface
Forgot Password Interface
Change Password Interace
Posting Announcements Interface
Roadmap Interface
To Do List Interface
My Application Interface
Application Form Interface
Course Registration Interface
My Applications Interface
Web Server Layer
Login Management
Main Menu Management
Profile Management
Forgot Password Management
Change Password Management
Posting Announcements Management
Roadmap Management
To Do List Management
My Application Management
Application Form Management
Course Registration Management
My Applications Management
Data Management Layer
Database
Entity

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

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.

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. All objects from the users to the application forms will be represented as tables in the database. 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.

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.

2.5 Boundary Conditions

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.

Erasmus++ users will have their accounts created for them and will be able to log in to the application with their password. The app can be exited via the log out button or by closing the window on which the app is being run. After closing the window, the user session will terminate.

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.

3.0 Low-Level Design

3.1. Object Design Trade-offs

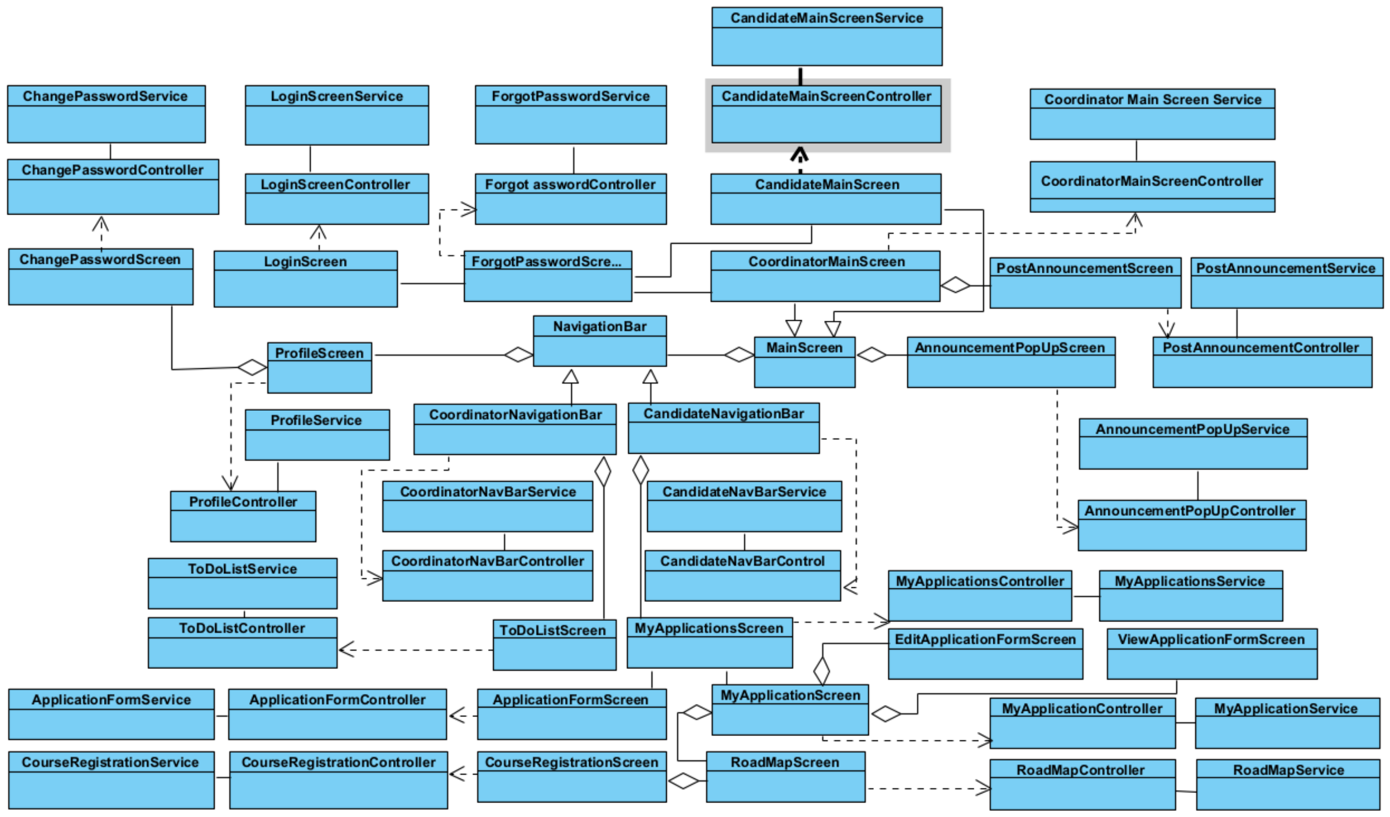
Maintainability vs Performance: By dividing the Erasmus++ into several layers, it has become much easier to maintain and implement the software. However, having multiple layers may cause minor loss in the performance which can be tolerated via the major maintainability advantages that come with layering.

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.

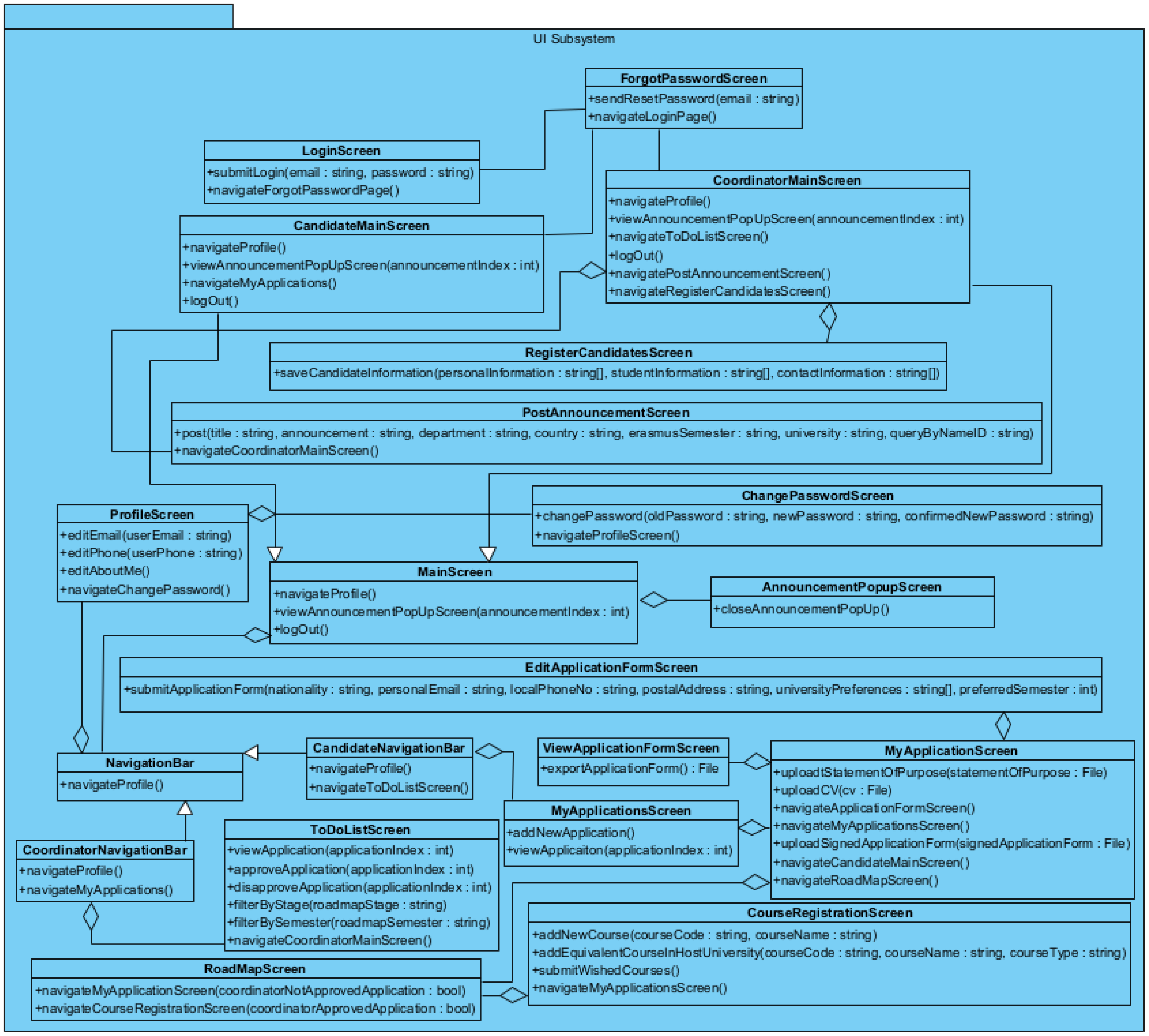
Functionality vs Usability: Erasmus++ have many actors that will be using the software. Providing different functionalities for coordinators and candidates decreases the usability for the users since there is massive function-load.

3.2 Final Object Design



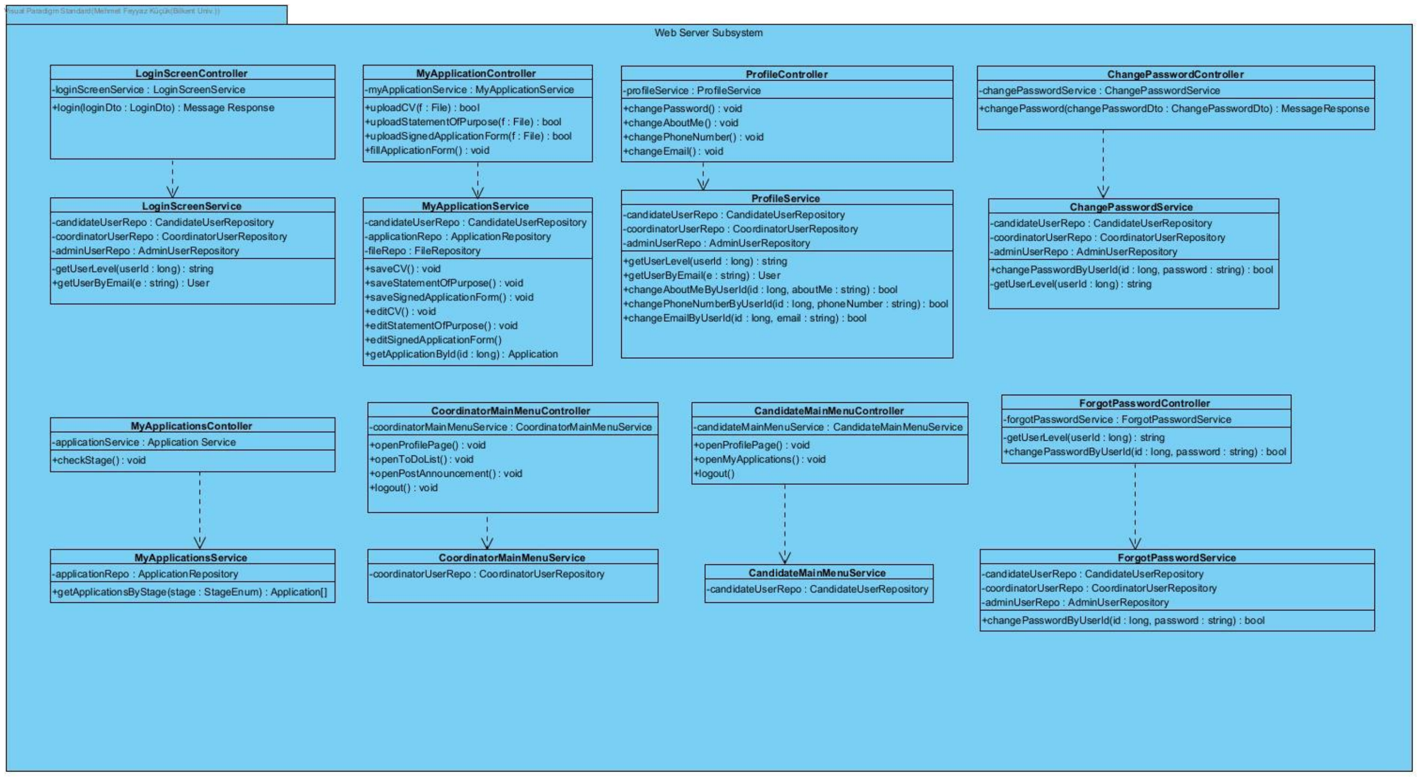
3.3 Layers

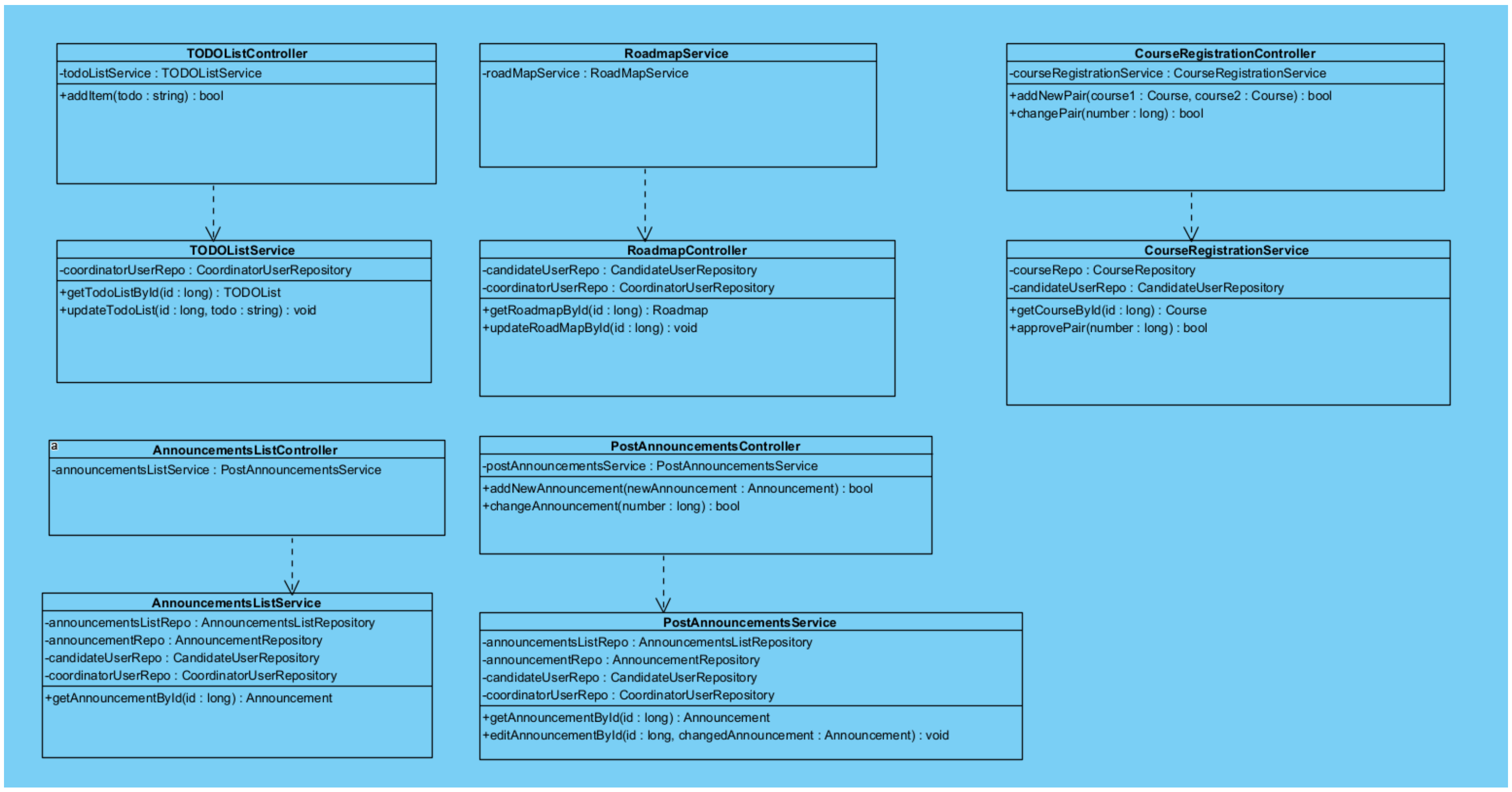
3.3.1 User Interface Layer


UI Subsystem
ForgotPasswordScreen
+sendResetPassword(email : string)
+navigateLoginPage()
LoginScreen
+submitLogin(email : string, password : string)
+navigateForgotPasswordPage()
CoordinatorMainScreen
+navigateProfile()
+viewAnnouncementPopUpScreen(announcementIndex : int)
+navigateToDoListScreen()
+logOut()
+navigatePostAnnouncementScreen()
+navigateRegisterCandidatesScreen()
CandidateMainScreen
+navigateProfile()
+viewAnnouncementPopUpScreen(announcementIndex : int)
+navigateMyApplications()
+logOut()
RegisterCandidatesScreen
+saveCandidateInformation(personalInformation : string[], studentInformation : string[], contactInformation : string[])
PostAnnouncementScreen
+post(title : string, announcement : string, department : string, country : string, erasmusSemester : string, university : string, queryByNameID : string)
+navigateCoordinatorMainScreen()
ChangePasswordScreen
+changePassword(oldPassword : string, newPassword : string, confirmedNewPassword : string)
+navigateProfileScreen()
ProfileScreen
+editEmail(userEmail : string)
+editPhone(userPhone : string)
+editAboutMe()
+navigateChangePassword()
MainScreen
+navigateProfile()
+viewAnnouncementPopUpScreen(announcementIndex : int)
+logOut()
AnnouncementPopupScreen
+closeAnnouncementPopUp()
EditApplicationFormScreen
+submitApplicationForm(nationality : string, personalEmail : string, localPhoneNo : string, postalAddress : string, universityPreferences : string[], preferredSemester : int)
NavigationBar
+navigateProfile()
CandidateNavigationBar
+navigateProfile()
+navigateToDoListScreen()
ViewApplicationFormScreen
+exportApplicationForm() : File
MyApplicationScreen
+uploadStatementOfPurpose(statementOfPurpose : File)
+uploadCV(cv : File)
+navigateApplicationFormScreen()
+navigateMyApplicationsScreen()
+uploadSignedApplicationForm(signedApplicationForm : File)
+navigateCandidateMainScreen()
+navigateRoadMapScreen()
MyApplicationsScreen
+addNewApplication()
+viewApplicaiton(applicationIndex : int)
CoordinatorNavigationBar
+navigateProfile()
+navigateMyApplications()
ToDoListScreen
+viewApplication(applicationIndex : int)
+approveApplication(applicationIndex : int)
+disapproveApplication(applicationIndex : int)
+filterByStage(roadmapStage : string)
+filterBySemester(roadmapSemester : string)
+navigateCoordinatorMainScreen()
CourseRegistrationScreen
+addNewCourse(courseCode : string, courseName : string)
+addEquivalentCourseInHostUniversity(courseCode : string, courseName : string, courseType : string)
+submitWishedCourses()
+navigateMyApplicationsScreen()
RoadMapScreen
+navigateMyApplicationScreen(coordinatorNotApprovedApplication : bool)
+navigateCourseRegistrationScreen(coordinatorApprovedApplication : bool)

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.

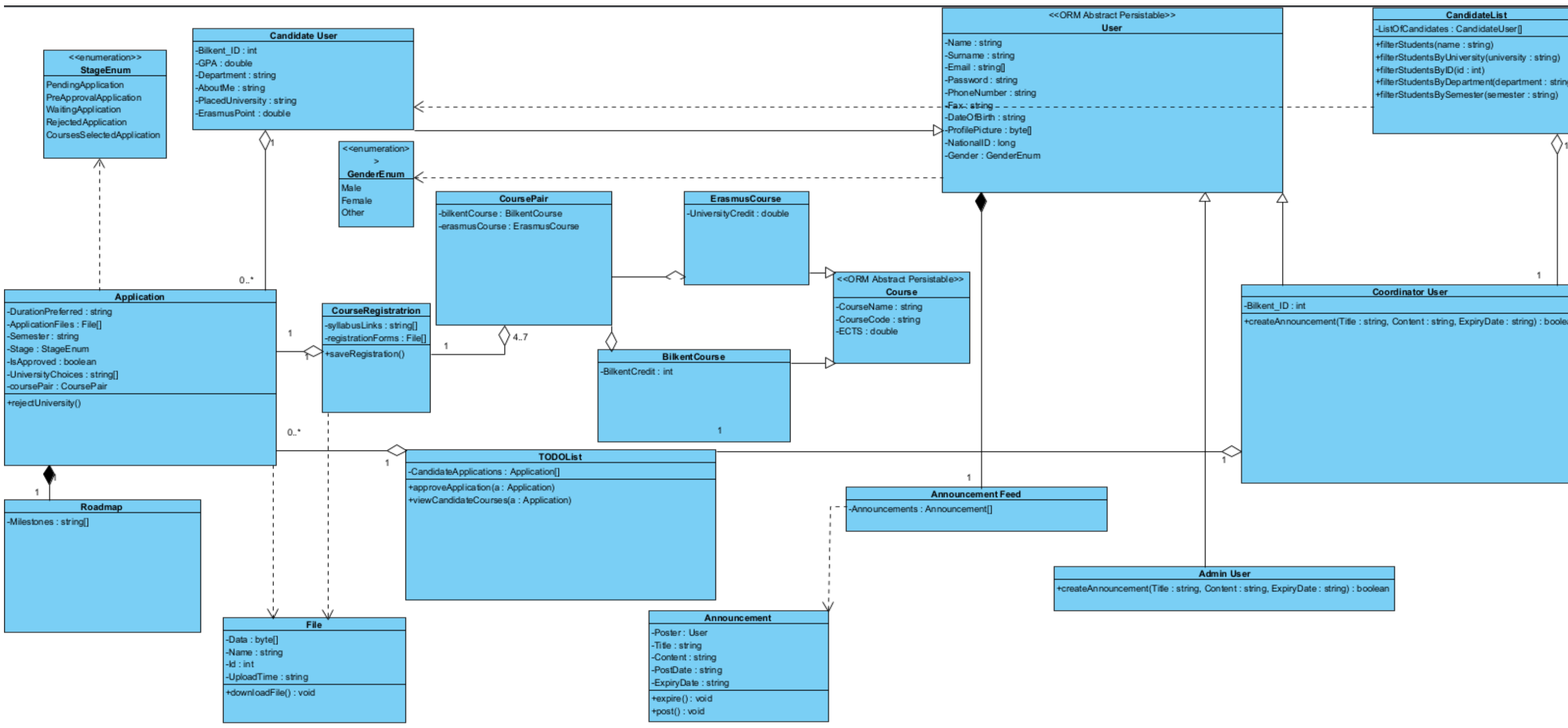
3.3.2 Web Server Layer

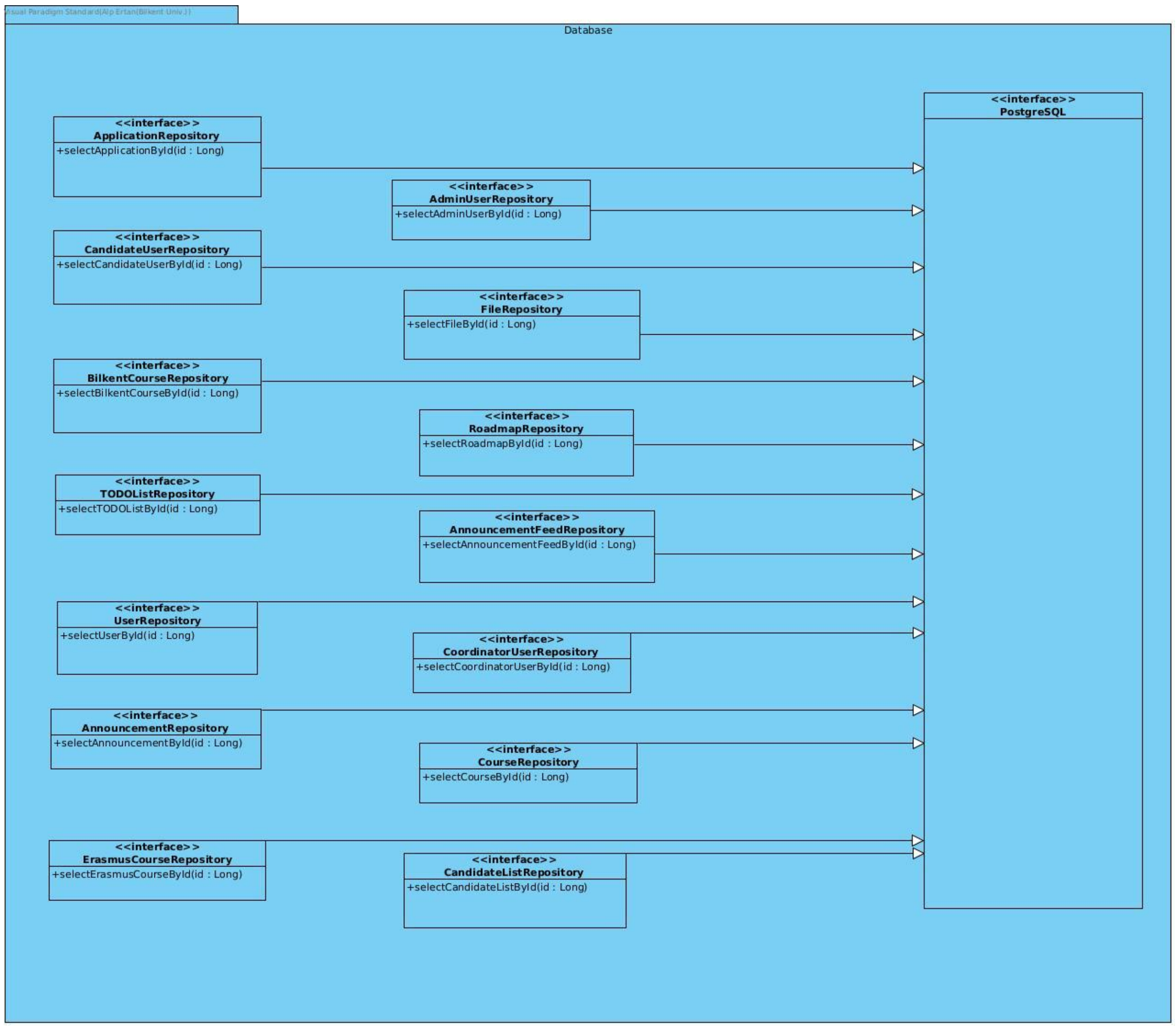

Visual Paradigm Standard(Mehmet Feyyaz Küçük(Bilkent Univ.))
Web Server Subsystem
LoginScreenController
-loginScreenService : LoginScreenService
+login(loginDto : LoginDto) : Message Response
LoginScreenService
-candidateUserRepo : CandidateUserRepository
-coordinatorUserRepo : CoordinatorUserRepository
-adminUserRepo : AdminUserRepository
-getUserLevel(userId : long) : string
+getUserByEmail(e : string) : User
MyApplicationController
-myApplicationService : MyApplicationService
+uploadCV(f : File) : bool
+uploadStatementOfPurpose(f : File) : bool
+uploadSignedApplicationForm(f : File) : bool
+fillApplicationForm() : void
MyApplicationService
-candidateUserRepo : CandidateUserRepository
-applicationRepo : ApplicationRepository
-fileRepo : FileRepository
+saveCV() : void
+saveStatementOfPurpose() : void
+saveSignedApplicationForm() : void
+editCV() : void
+editStatementOfPurpose() : void
+editSignedApplicationForm()
+getApplicationById(id : long) : Application
ProfileController
-profileService : ProfileService
+changePassword() : void
+changeAboutMe() : void
+changePhoneNumber() : void
+changeEmail() : void
ProfileService
-candidateUserRepo : CandidateUserRepository
-coordinatorUserRepo : CoordinatorUserRepository
-adminUserRepo : AdminUserRepository
+getUserLevel(userId : long) : string
+getUserByEmail(e : string) : User
+changeAboutMeByUserId(id : long, aboutMe : string) : bool
+changePhoneNumberByUserId(id : long, phoneNumber : string) : bool
+changeEmailByUserId(id : long, email : string) : bool
ChangePasswordController
-changePasswordService : ChangePasswordService
+changePassword(changePasswordDto : ChangePasswordDto) : MessageResponse
ChangePasswordService
-candidateUserRepo : CandidateUserRepository
-coordinatorUserRepo : CoordinatorUserRepository
-adminUserRepo : AdminUserRepository
+changePasswordByUserId(id : long, password : string) : bool
-getUserLevel(userId : long) : string
MyApplicationsContoller
-applicationService : Application Service
+checkStage() : void
MyApplicationsService
-applicationRepo : ApplicationRepository
+getApplicationsByStage(stage : StageEnum) : Application[]
CoordinatorMainMenuController
-coordinatorMainMenuService : CoordinatorMainMenuService
+openProfilePage() : void
+openToDoList() : void
+openPostAnnouncement() : void
+logout() : void
CoordinatorMainMenuService
-coordinatorUserRepo : CoordinatorUserRepository
CandidateMainMenuController
-candidateMainMenuService : CandidateMainMenuService
+openProfilePage() : void
+openMyApplications() : void
+logout()
CandidateMainMenuService
-candidateUserRepo : CandidateUserRepository
ForgotPasswordController
-forgotPasswordService : ForgotPasswordService
-getUserLevel(userId : long) : string
+changePasswordByUserId(id : long, password : string) : bool
ForgotPasswordService
-candidateUserRepo : CandidateUserRepository
-coordinatorUserRepo : CoordinatorUserRepository
-adminUserRepo : AdminUserRepository
+changePasswordByUserId(id : long, password : string) : bool


TODOListController
-todoListService : TODOListService
+addItem(todo : string) : bool
TODOListService
-coordinatorUserRepo : CoordinatorUserRepository
+getTodoListById(id : long) : TODOList
+updateTodoList(id : long, todo : string) : void
RoadmapService
-roadMapService : RoadMapService
RoadmapController
-candidateUserRepo : CandidateUserRepository
-coordinatorUserRepo : CoordinatorUserRepository
+getRoadmapById(id : long) : Roadmap
+updateRoadMapById(id : long) : void
CourseRegistrationController
-courseRegistrationService : CourseRegistrationService
+addNewPair(course1 : Course, course2 : Course) : bool
+changePair(number : long) : bool
CourseRegistrationService
-courseRepo : CourseRepository
-candidateUserRepo : CandidateUserRepository
+getCourseById(id : long) : Course
+approvePair(number : long) : bool
a
AnnouncementsListController
-announcementsListService : PostAnnouncementsService
AnnouncementsListService
-announcementsListRepo : AnnouncementsListRepository
-announcementRepo : AnnouncementRepository
-candidateUserRepo : CandidateUserRepository
-coordinatorUserRepo : CoordinatorUserRepository
+getAnnouncementById(id : long) : Announcement
PostAnnouncementsController
-postAnnouncementsService : PostAnnouncementsService
+addNewAnnouncement(newAnnouncement : Announcement) : bool
+changeAnnouncement(number : long) : bool
PostAnnouncementsService
-announcementsListRepo : AnnouncementsListRepository
-announcementRepo : AnnouncementRepository
-candidateUserRepo : CandidateUserRepository
-coordinatorUserRepo : CoordinatorUserRepository
+getAnnouncementById(id : long) : Announcement
+editAnnouncementById(id : long, changedAnnouncement : Announcement) : void

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




Visual Paradigm Standard(Alp Ertan(Bilkent Univ.))
Database
<<interface>> PostgreSQL
<<interface>> ApplicationRepository
+selectApplicationById(id : Long)
<<interface>> AdminUserRepository
+selectAdminUserById(id : Long)
<<interface>> CandidateUserRepository
+selectCandidateUserById(id : Long)
<<interface>> FileRepository
+selectFileById(id : Long)
<<interface>> BilkentCourseRepository
+selectBilkentCourseById(id : Long)
<<interface>> RoadmapRepository
+selectRoadmapById(id : Long)
<<interface>> TODOListRepository
+selectTODOListById(id : Long)
<<interface>> AnnouncementFeedRepository
+selectAnnouncementFeedById(id : Long)
<<interface>> UserRepository
+selectUserById(id : Long)
<<interface>> CoordinatorUserRepository
+selectCoordinatorUserById(id : Long)
<<interface>> AnnouncementRepository
+selectAnnouncementById(id : Long)
<<interface>> CourseRepository
+selectCourseById(id : Long)
<<interface>> ErasmusCourseRepository
+selectErasmusCourseById(id : Long)
<<interface>> CandidateListRepository
+selectCandidateListById(id : Long)

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.