Bilkent University

Department of Computer Engineering

CS319 Term Project

Section 3

Group 3A

Bilasmus

Design Report

Group Members:

Alp Afyonluoğlu (22003229)
Emre Karataş (22001641)
Erkin Aydın (22002956)
Parsa Keihan (22001422)
Selin Bahar Gündoğar (22001514)

Instructor: Eray Tüzün
Teaching Assistant: Mert Kara

Contents

Design Report

Bilasmus

1. Introduction

1.1. Purpose of the system

Bilasmus is a web-based project to help both Bilkent University students and faculty members during Erasmus and exchange programs. Bilasmus is used after the approval of students. The program's primary goal is to minimize paperwork as much as possible, and ease communication between students and faculty members. Bilasmus has different user types, such as Erasmus/Exchange coordinators, outgoing/incoming students, admin, department secretary, and the international student office. Thus, essential processes such as the course approval period and its paperwork are done in the Bilasmus system. The system also provides and reflects any update on each period of the Erasmus/Exchange process. The website also provides an in-system messaging platform and a To-Do list for each user as a reminder.

1.2. Design goals

The application offers a sustainable, user-friendly, and functional user interface. As there are users of different kinds, Bilasmus will address to users with all users, even those with low knowledge of computer and technology organization. Also, it is essential to note that requirements from the analysis report form the design goals and purposes of the system; therefore, the system should be secure and maintainable as well. It should also have the most efficient performance for the best experience. Though, the most important two non-functional requirements are usability and reliability in our project.

1.2.1. Usability/User-friendliness

Bilasmus is a program that addresses a variety of actors: outgoing students, incoming students, Faculty Board Executive Committee representative, coordinators, instructors, International Student Office, admin, and department secretary. Thus, not everyone may be familiar with the newest technologies or may want to spend so much time on the application. Therefore, the program is planned to be designed as simple and usable as possible with the most user-friendly interface possible. This is to say that the features used in the system, such as buttons, sections, upload/download fields, toolbars, etc., should be easy-to-use. Each section of the application will have text sections for those unfamiliar, and there is a help button to let the user contact the responsible if there is a problem or question. Every clickable section (such as buttons) is appropriately designed, and each text field or input section is adequately aligned. The color design of the program is also helpful for the users as they are matched with the general concept of design. For example, delete buttons are colored red, and download buttons are colored green. In general, the program will be designed to provide multi-dimensional ease of operation and intelligibility to various users. This section's important features are an easy-to-read font theme and size, aligned text/button/alert fields, and a useful tab for transitions.

1.2.2. Security/Reliability

The system will be as secure as possible as it contains essential documents of Erasmus/Exchange students and actors' personal details. Details such as ID, password, courses, transcripts, etc. will be shared with other actors accordingly if needed. Each actor will see the required information provided for them according to their roll in in the application. This means a student cannot access some other student's transcript. Moreover, the system should ensure that no data will be deleted during any crash. Each completed assignment in the system should be saved and kept as a record in the system as long as needed based on the university's rules.

1.2.3. Sustainability/Maintainability

The system should be supported for all up-to-date web browsers. The locations of every feature in a page, such as buttons, alerts, and text fields should be sustained in all devices. The database should be updated every 30 minutes, and the overall control should be maintained by admins if needed. Also, the system will be designed in a way that anyone as Bilasmus' new developer could build up on it easily. To make the code of the program understandable, comments and reports will be used; therefore, new features can be added faster and easier. As we apply OOP, it ensures that adding new elements or modifying the existing ones will not be a difficulty.

1.2.4. Performance

Overall, the system should be as sufficient as it can be. This means there should not be any crash, data loss, and user dissatisfaction performance wise. This follows that users should run the program fluently and have the best experience provided by the system. Therefore, the number of the users using the system at the same time should not cause a problem in the performance of the system, and anyone with internet access and sustainable web browser should be able to use the system.

2. High-Level Software Architecture

2.1. Subsystem Decomposition

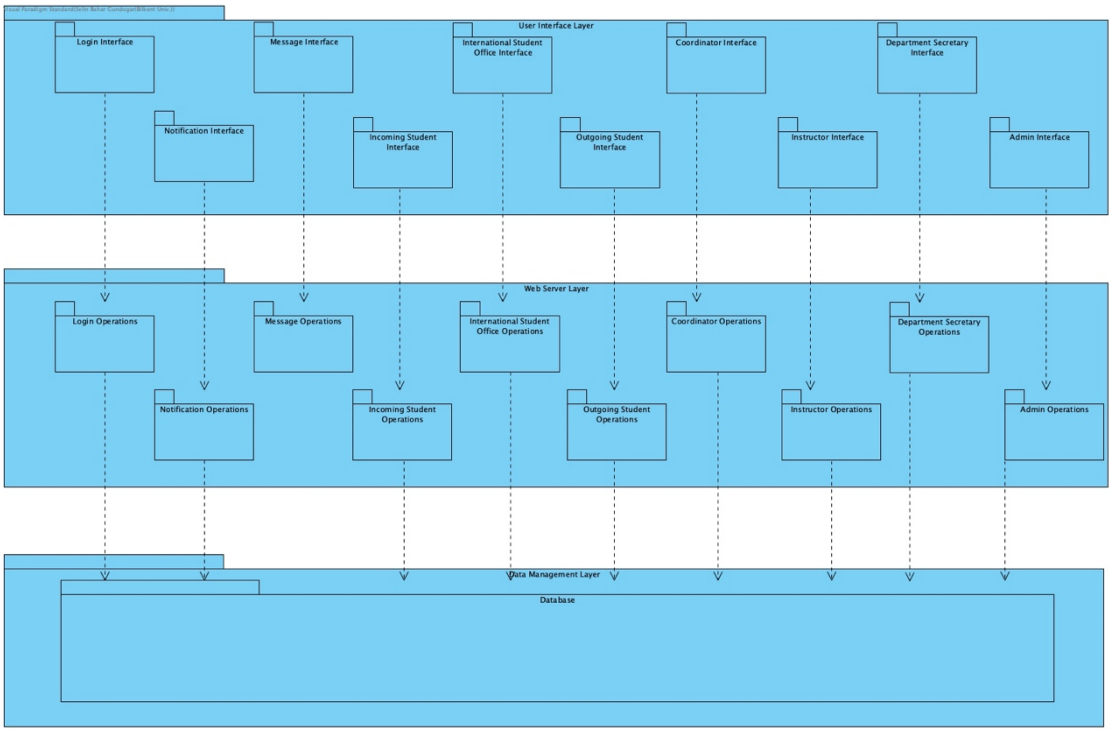

Figure 2.1: 3 Layer Architecture

A three-layer architecture was used in our planning. The User Interface Layer represents the boundary level and is responsible for guiding and showing the user the visuals of the website and takes requests from users and sends it to the controller layer. This layer is purposed to fit into our usability/user-friendliness design goal.

The Middle section is the Web Server Layer which serves as the backend of the code and represents the controller section. It is connected to both the boundary and the entity part of the architecture. This layer matches the sustainability/maintainability and performance criteria of the design goals.

The Data Management Layer contains the database and its repositories contains all information about the users and their files. It represents the Entity layer and is connected to controller. This layer is supposed to match the security/reliability criteria of the design goal.

2.2. Subsystem Explanations

2.2.1. Interface and Web Server Layers

2.2.1.1. Login Interface – Login Operations

Login Interface subsystem is responsible for the login, resetting password, and registering purposes of Bilasmus. The same screen is used for all user types. Users are directed to the reset password page after clicking on the registration link which they receive through email.

The Login Operations subsystem fetches user information from the database and validates if the inputs match with the registered users in the database. Every password change is updated in the database.

2.2.1.2. Notification Interface – Notification Operations

The Notification Interface subsystem contains the screens that the user sees regarding receiving a notification. Users such as coordinators, instructors, international student office, or department secretary automatically receive notifications from the system whenever a student uploads a file that is directed towards the specific user. Moreover, students also receive notifications when a coordinator, instructors, international student office, or department secretary accept, reject, or upload a document related to that student. This notification dropdown can be accessed through the navbar.

The Notification Operations subsystem communicates with the database to detect any changes of a file to send the appropriate notifications to the right users. Moreover, the by sensing a change in the status of a report, an email is sent to the user as well regarding the notification.

2.2.1.3. Message Interface – Message Operations

The Message Interface subsystem is responsible for displaying the incoming and outgoing messages of users. The messages can be accessed from the navbar and the users can view an overview of all chats. These chats can be seen more detailedly if the user clicks on s specific chat and the previously sent messages are shown on the screen.

The Message Operations subsystem is responsible for fetching all messages of a user and receiving any sent messages and storing it in the database. The management subsystem also sends all received messages to the interface subsystem.

2.2.1.4. Incoming Student Interface – Incoming Student Operations

The Incoming Student Interface subsystem contains the course request form creation related screens. The screen allows the user to enter information of Bilkent courses.

The Incoming Student Operations subsystem is responsible for storing the submitted course files and sending the files to the other related users such as the coordinator or department secretary users.

2.2.1.5. International Student Office Interface - International Student Office Operations

The International Student Office Interface subsystem has screens for receiving language proficiency exam results, sending transcripts to the coordinator, adding to do list items to the coordinators and students, and sending the registration link.

The International Student Office Operations subsystem makes the connection to the database to fetch all data regarding the international student office. All language proficiency exam results are fetched and information such as announcing the date of a language proficiency exam taking place is sent to the database.

2.2.1.6. Outgoing Student Interface - Outgoing Student Operations

The Outgoing Student Interface subsystem includes the screens related to the outgoing students such as course request forms, pre-approval upload page, and learning agreement upload page. The user can also reupload or delete these PDF files uploaded to the application.

The Outgoing Student Operations subsystem is responsible for receiving and storing the PDF files to the database. Any changes made to the files are updated in the database as well.

2.2.1.7. Coordinator Interface - Coordinator Operations

The Coordinator Interface subsystem has the screens such as received course request forms, received pre-approval forms, received learning agreement forms, and uploading course transfer forms. The coordinator user also can see the student waitlist excel sheet.

The Coordinator Operations subsystem fetches and retrieves all the files mentioned above to the database and saves any changes made to the database. Moreover, the subsystem also sends any files that need to reach another user to those users. Any courses that are accepted from the outgoing student course request forms for the first time is automatically saved to the database as an accepted course, so that the course information can be used again in the future.

2.2.1.8. Instructor Interface - Instructor Operations

The Instructor Interface subsystem contains the screen of all mandatory course requests coming from outgoing students that the instructor is the course coordinator of. The instructor accepts or rejects the courses through the user interface.

The Instructor Operations subsystem fetches and stores all course requests from the database that are supposed to be received by the instructor. Any accepting or rejecting of a course is administered into the database as well.

2.2.1.9. Admin Interface - Admin Operations

The Admin Interface subsystem has the admin related screens such as the add, update or delete a user features. Moreover, the admin interface includes sending transcripts to the coordinator. All actions done on the application is directed to the Admin Operations subsystem.

The Admin Operations subsystem receives actions done in the interface subsystem and the changes are stored to the database. The database keeps all transcripts.

2.3. Data Management Layer

2.3.1. Deployment Diagram

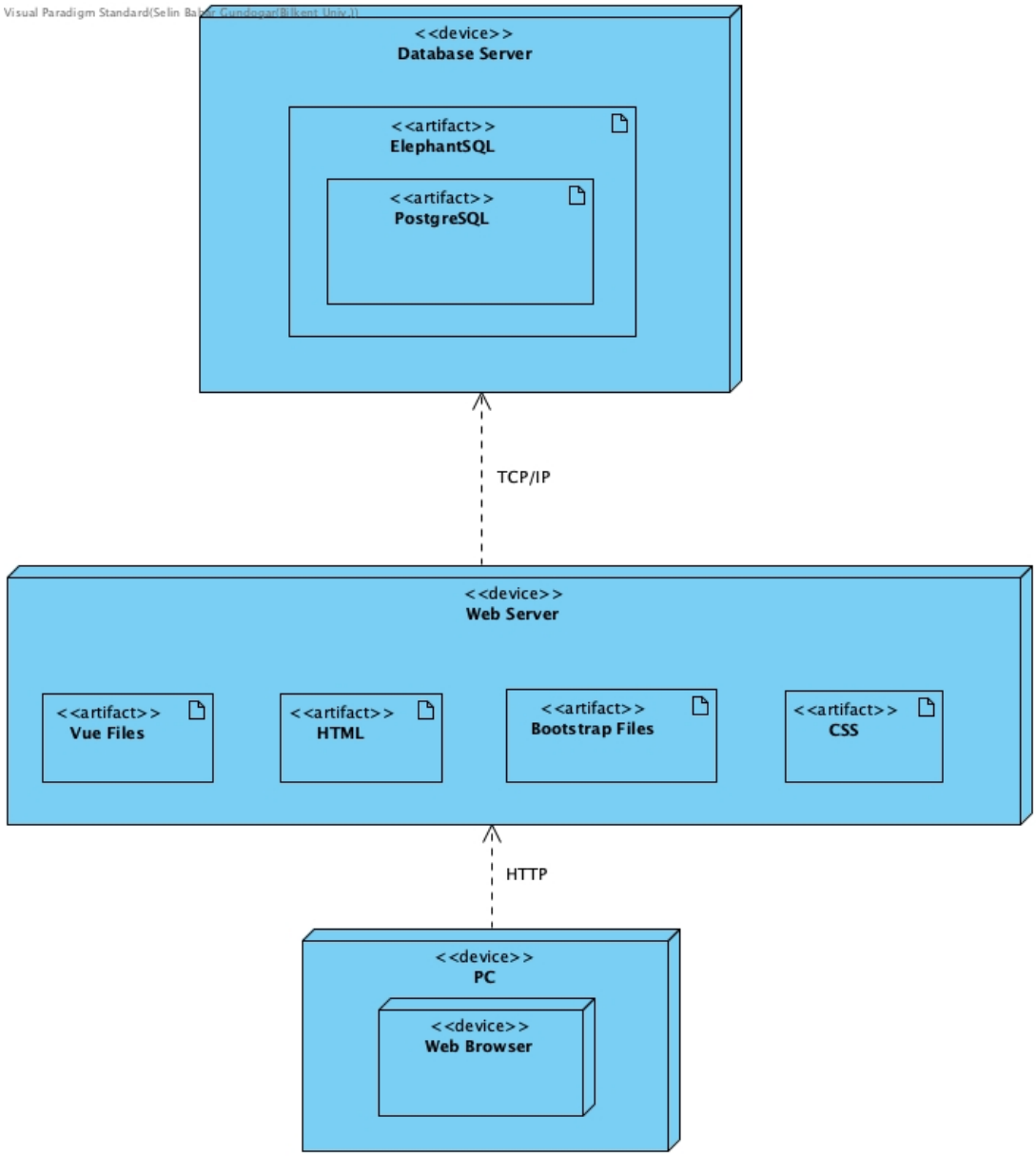

Figure 2.3.1: Deployment Diagram

The Database server node uses the Database subsystem.

The Web Server node uses the Login Operations, Notification Operations, Incoming Student Operations, International Student Office Operations, Outgoing Student Operations, Coordinator Operations, Instructor Operations, and Admin Operations subsystems.

The PC node uses the Login Interface, Notification Interface, Incoming Student Interface, International Student Office Interface, Outgoing Student Interface, Coordinator Interface, Instructor Interface, and Admin Interface subsystems.

2.4. Hardware/Software Mapping

This is a web project. Hence, there are no strict hardware requirements from the user side, any device able to handle modern web browsers should be sufficient. Our program should be able to run stable versions of Google Chrome and Chromium-based browsers, Mozilla Firefox, and Safari. Our project should work on any operating system running these browsers. Typical computer components, such as at least one monitor, keyboard, and mouse, are required to use the web application. Additionally, to host the website, a device with 8GB RAM and an Intel Pentium 4 or an equivalent processor, such as AMD Athlon XP 3200, is sufficient.

In the back end, Node.js version 16.0 is used. In the front end, Vue3 is used, requiring Node.js version 16.0 or higher. Both Node.js and Vue3 are also dependent on npm to work. Hence, Vue3 version 3.2.45, Node.js version 16.0 or a higher version of it, and npm 8.12.1 should be installed on the host machine. Additionally, Bootstrap5 was used in front end, hence Bootstrap5 should also be installed on the hosting machine.

2.5. Persistent Data Management

There were two options for databases: relational databases (SQL) and non-relational databases (NoSQL). There can be, of course, other types of databases, but since we have never heard of them, they were never an option in this limited time. We knew about non-relational databases, and some worked with non-relational databases. However, since non-relational databases are not dynamic, we thought they might create specific problems. That’s why we eliminated non-relational databases.We chose PostgreSQL as it was the one we were most familiar with.

Users' personal data should be persistent throughout the program and should not be modified. This data includes name, surname, ID, e-mail. Relational databases are dependent on SQL queries. As a result of SQL queries, the data in the DB may be changed. To change the data, related SQL query should be embedded into the backend code, connected with related controller classes. If there is no interaction or call to modify data, data should not be modified and be stayed in the DB as it is. This process is accompanied by PostgreSQL and node-postgres module created for Node.js.

2.6. Access Control and Security

BILASMUS	Incoming Student	Outgoing Student	Coordinator	Instructor	Admin	Faculty Committee Board	Department Secretary	International Office
Login Page	Login (id: integer, email:string)	Login (id: integer, email: string)	Login (id: integer, email: string)	Login (id: integer, email: string)	Login (id: integer, email:string)	Login (id: integer, email:string)	Login (id: integer, email:string)	Login (id: integer,email:string)
Login Failed Page	Notify ()	Notify ()	Notify ()	Notify ()	Notify ()	Notify ()	Notify ()	Notify ()
Reset Password Page	ResetPassword (email: string)	ResetPassword (email: string)	ResetPassword (email: string)	ResetPassword (email: string)	ResetPassword (email: string)	ResetPassword (email: string)	ResetPassword (email: string)	ResetPassword (email: string)
Profile Page	goCourseRequest () sendMessage (message: String) Logout ()	goCourseRequest () goPreApprovalForm () goNotifications () goMessage (message: String) goHelp () Logout ()	goCourseRequest () goMessage (message: String) goNotifications () goHelp () Logout ()	goCourseRequest () goMessage (message: String) goNotifications () goHelp () Logout ()	goCourseRequest () goMessage (message: String) goNotifications () goHelp () Logout ()	goCourseRequest () goMessage (message: String) goNotifications () goHelp () Logout ()	goCourseRequest () goMessage (message: String) goNotifications () goHelp () Logout ()	goCourseRequest () goMessage (message: String) goNotifications () goHelp () Logout ()
Notifications Page	listNotifications (NotfList: ArrayList)	listNotifications (NotfList: ArrayList)	listNotifications (NotfList: ArrayList)	listNotifications (NotfList: ArrayList)	listNotifications (NotfList: ArrayList)	listNotifications (NotfList: ArrayList)	listNotifications (NotfList: ArrayList)	listNotifications (NotfList: ArrayList)
Help Page	askQuestion (q: String)	askQuestion (q: String)	askQuestion (q: String)	askQuestion (q: String)	askQuestion (q: String)	askQuestion (q: String)	askQuestion (q: String)	askQuestion (q: String)
Message Page	sendMessage (Message msg) deleteMessage (Message msg) viewAllMessages () seeMessage (Message msg) replyMessage (Message msg)	sendMessage (Message msg) deleteMessage (Message msg) viewAllMessages () seeMessage (Message msg) replyMessage (Message msg)	sendMessage (Message msg) deleteMessage (Message msg) viewAllMessages () seeMessage (Message msg) replyMessage (Message msg)	sendMessage (Message msg) deleteMessage (Message msg) viewAllMessages () seeMessage (Message msg) replyMessage (Message msg)	sendMessage (Message msg) deleteMessage (Message msg) viewAllMessages () seeMessage (Message msg) replyMessage (Message msg)	sendMessage (Message msg) deleteMessage (Message msg) viewAllMessages () seeMessage (Message msg) replyMessage (Message msg)	sendMessage (Message msg) deleteMessage (Message msg) viewAllMessages () seeMessage (Message msg) replyMessage (Message msg)	sendMessage (Message msg) deleteMessage (Message msg) viewAllMessages () seeMessage (Message msg) replyMessage (Message msg)
Detailed Info Page		Search (input: String) checkDocument (doc: Document)						
Outgoing Student Main Page		Search (input: String) checkDetail (crs: Course)						
		checkSllyabus (sls: Sllyabus)						
Course Request Page – Outgoing Students		SetID (id: int) setCourseName (name: String) setLink (link: String) setType (c: Course) setCourseToSatisfy (c: course) setSllyabus (s: Sllyabus) add (request: CourseRequest) update (request: CourseRequest) send (request: CourseRequest)						
Language Proficiency Page		setDocument (d: Document) update (d: Document)						
Pre-Approval Form Page		choose (d: Document) delete (d: Document) download (d: Document) seeStatus ()						
File Upload Page		upload (d: Document) delete (d: Document) download (d: Document) seeStatus () seeStatus ()						
Learning Aggrement Page		choose (d: Document) delete (d: Document) download (d: Document) seeStatus ()						
Main Page – Incoming Students	seeCourses (this: IncomingStudent) checkDetails (c: Course) checkSllyabus (c: Course)							
Course Request Page –	SetCourseID(id:int)							

Incoming Students	setName (name: String) add (c: Course) update (c: Course) delete (c: Course) submit (d: Document)							
Main Page - Coordinator			checkCourseApprovals (d: Document) approve (d: Document) reject (d: Document) seeStatus (d: Document) goPreApproval () goRequest ()					
Course Request Page- Coordinator			checkCourseApprovals (d: Document) approve (d: Document) reject (d: Document) seeStatus (d: Document) goPreApproval () search (input: String)					
View Course Request- Coordinator			approve (d: Document) reject (d: Document) seeStatus (d: Document) search (input: String) goLink (c: Course) goSllyabus (s: Sllyabus)					
View Pre-Approval - Coordinator			approve (d: Document) reject (d: Document) seeStatus (d: Document)					
View Learning Aggrement - Coordinator			approve (d: Document) reject (d: Document) seeStatus (d: Document)					
Course Transfer Form			seePending () upload (d: Document) save (d: Document) search (d: Document)					
Main Page - International Office								sendTranscript(id:int) addTitle (str: String) addDate (d: Date) setReminder (flag: boolean) setTodoTarget (flag: Boolean)
								sendRegLink (id:int, flag: boolean)
View Language Proficiency – International Office								Submit (d: Document) Edit (s: Student)
Main Page- Department Secretary							approve (d: Document) reject (d: Document) seeStatus (d: Document) sendMessage (t: Student)	
Course Request Page – Department Secretary							approve (d: Document) reject (d: Document)	
Main Page- Faculty Committee Board						View (d: Document) setSignature(user:user)		
Course Transfer Form Page - FCB						approve (d: Document) reject (d: Document) check (d: Document) search (d: Document)		
View Pre-Approval Form – FCB						approve (d: Document) reject (d: Document) see (d: Document)		
Main Page - Instructor				approve (d: Document) reject (d: Document) seeStatus (d: Document) goLink (c: Course) goSllyabus (s: Sllyabus)				
Main Page- Admin					Add (user: User) setName (name: String) setID(id:int)			
					setEmail (mail: String) setType (flag: boolean) delete(user:user) update(user:user)			

Figure 2.6 Access Control Matrix

Security and reliability of the data is guaranteed in the software by using default packages of PostgreSQL. The data is obtained from Bilkent University Erasmus+ website, which is real data and consistent with each other and the reality. The data obtained is kept at local server, which is protected by at least three different powerful

passwords. Also, every log on the PostgreSQL is kept and if an unexpected log is observed, the data administrator will be notified to take necessary precautions. All the users in the system can login and sign out from the application. Depending on their role, they can see reports submitted to them, the reports that they need to submit, notifications, messages from other users. Furthermore, every user may reach the help section in the navigation bar to tell them how to use the application.

Student users are divided up to two different roles, incoming student, and outgoing student. Incoming students coming from abroad universities to Bilkent University submits a form of the courses they wish to take at Bilkent and their request is sent to the coordinator for confirmation. Coordinator evaluates the list and sends it to the department secretary. Based on the coordinator's approval, the incoming students will be registered to their courses by the department secretary. On the other hand, outgoing students submit their course form, pre-approval form, learning agreement, and their language proficiency results before going to Erasmus/Exchange. The coordinator is responsible for checking all of these submitted documents except for the language proficiency result. For the course form, the instructor may also access the course if the course is a mandatory course that has not been approved before. Learning agreement forms are approved or rejected by the faculty executive committee board after it has been approved by the coordinator.

Apart from evaluating students' forms, the coordinator also uploads a course transfer form to the faculty executive committee board. Coordinator also receives the transcripts from the international student office and receives Erasmus student ranking excel sheet from the admin.

Admin staff may add new users to the program. If it is needed, they can change information regarding users. In some cases (if a student cancels his/her placement or for other valid reasons) admin staff may delete users from the system. Admin also sends the list of student raking for Erasmus to the coordinator.

Department secretary receives the list of courses that the incoming students wish to take after the courses have been approved by the coordinator. International student office is responsible for announcing language proficiency exams taking place, processing language proficiency results, and sending the transcripts to coordinators.

Faculty executive committee board is responsible for approving or rejecting learning agreement forms and course-transfer forms.

2.7. Boundary Conditions

2.7.1. Initialization

Bilasmus application is initialized through a database hosted at ElephantSQL, but can also be initialized on a local machine. The initialization of the application requires the tables in the database to be already created and certain data to be already stored in the database such as the list of all previously accepted courses of Erasmus/Exchange universities. The database connection is set-up through Pg Admin 4, via port 5433 by username "postgres" through server name "localhost", when deployed locally. The database's name is BilasmusDB, and it is protected by three different passwords. PostgreSQL 15 is used for the connection.

Additionally, Node.js v16 is used in our application. While the Node.js app will be running on Google Cloud Platform's App Engine and will be accessible online, the

app can also be initialized on a local machine like the database. In order to run the application locally, the zip file containing the project must be unzipped, Node.js version 16 and up must be installed, and PostgreSQL database tables must be ready. The connections between the three layers, which are web, application, and database, must be established correctly. The correct database connection with the back-end must have been set up and the front-end and back-end must have been connected. Then, the program can be run by first installing modules and then starting the application, which can be done by executing the following command in order after navigating to the root path of the unzipped project folder: “npm install”, “node ./bin/www”.

The initialization of the application from the user side includes the admin to register every other user that will be using the application to the system, in which the system automatically sends a URL link through email to registered users of the application. By clicking on the link, users will decide on their passwords on the application and be able to use the application.

2.7.2. Termination

Any resulting subsystem crash will automatically lead to the termination of the application as the system must be working fully right to save any data changes the user may make. The subsystems will communicate each other if one fails; thus, making the whole program stop before notifying the user of a crash. The user will be notified if a system crash does occur and will be prompted to login to the application again. The user will not be able to login if the application is still down and will be notified about it on the application. The system will not abruptly shut down but instead users will be notified of it. Furthermore, if the admin makes any maintenance to the application, then the users will be notified priorly on the application a day before the maintenance occurs.

In order to prevent any crash from happening, a timeout system will be used to gracefully shut down the application. All users connected to the application will be given time to complete their action on the application and they will be notified of the application shutting down. Moreover, a health check route will be added to the back-end to ensure the application is running at all times and the front-end will receive the return status code. Additionally, the application will ensure that there will be no data loss when any unhandled exception or error occurs as it will not cause an immediate shutdown of the application but instead the user will be alerted of an error. In such case, the user will not be able to continue with doing any type of action in the application and the program will log the user out.

2.7.3. Failure

Two recovery layers will be used to handle unexpected exit conditions. First, Node.js exit events, such as ‘exit’, will be listened to and the exit code will be logged for debugging later. The PM2 module will be used to restart the server in case of a crash. Second, for Google Cloud Services, there will be a disaster recovery plan (drp) to implement on the Bilasmus project. Google Cloud Storage will backup the data to the system whenever the crash happens, according to drp.

3. Low-Level Design

3.1. Object Design Trade-Offs

3.1.1. Functionality v. Usability

Our project is supposed to be used by everyone who is involved in the Erasmus project, ranging from incoming and outgoing students to the academic staff such as coordinators, instructors, faculty board committee. This makes the application functional. On the other hand, the application and interface are easy to use. It has been tried to keep a balance between functionality and usability. However, because the application focuses on students' and staffs' use, we focused more on usability.

3.1.2. Rapid Development v. Functionality

Because of time limitations on the development of the application, rapid development is one of the main concerns for the application. This time constraint causes the application to be less functional.

3.1.3. Security v. Usability

When a user logs in to the application, unless the user signs out, the session remains active for one hour. This increases usability but security decreases as there is no need to log in again in one hour. Also, it should be noted that new passwords are sent by e-mail to the user. This increases security of the user.

3.2. Layers

3.2.1. Final Object Design

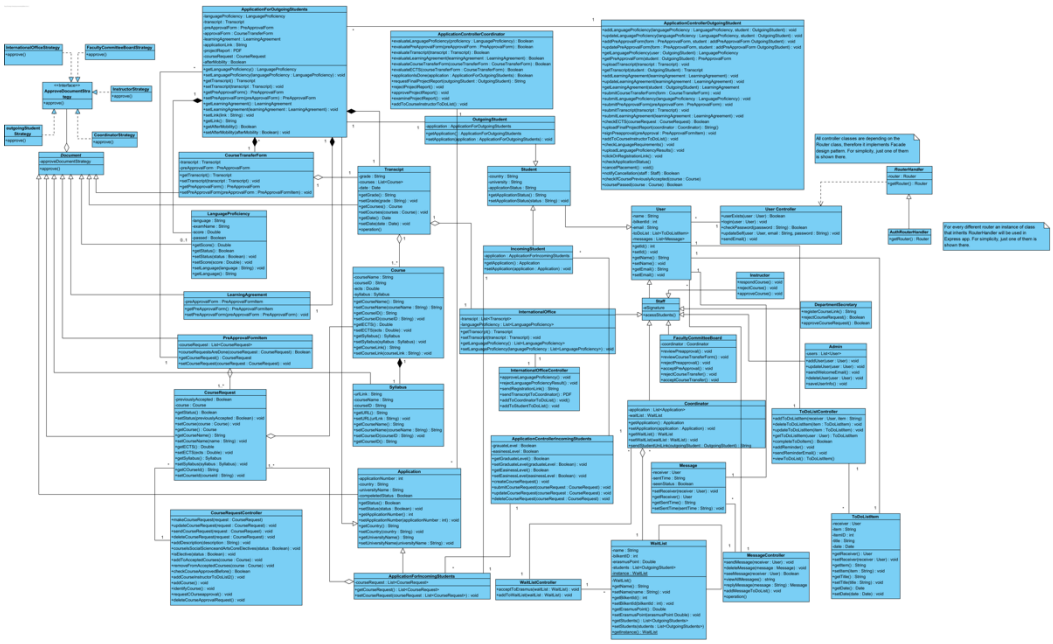

Figure 3.2: Entity-Controller Diagram

Figure 3.2 represents entity-controller diagram of the Bilasmus application. For readability, the diagram split into three different parts.

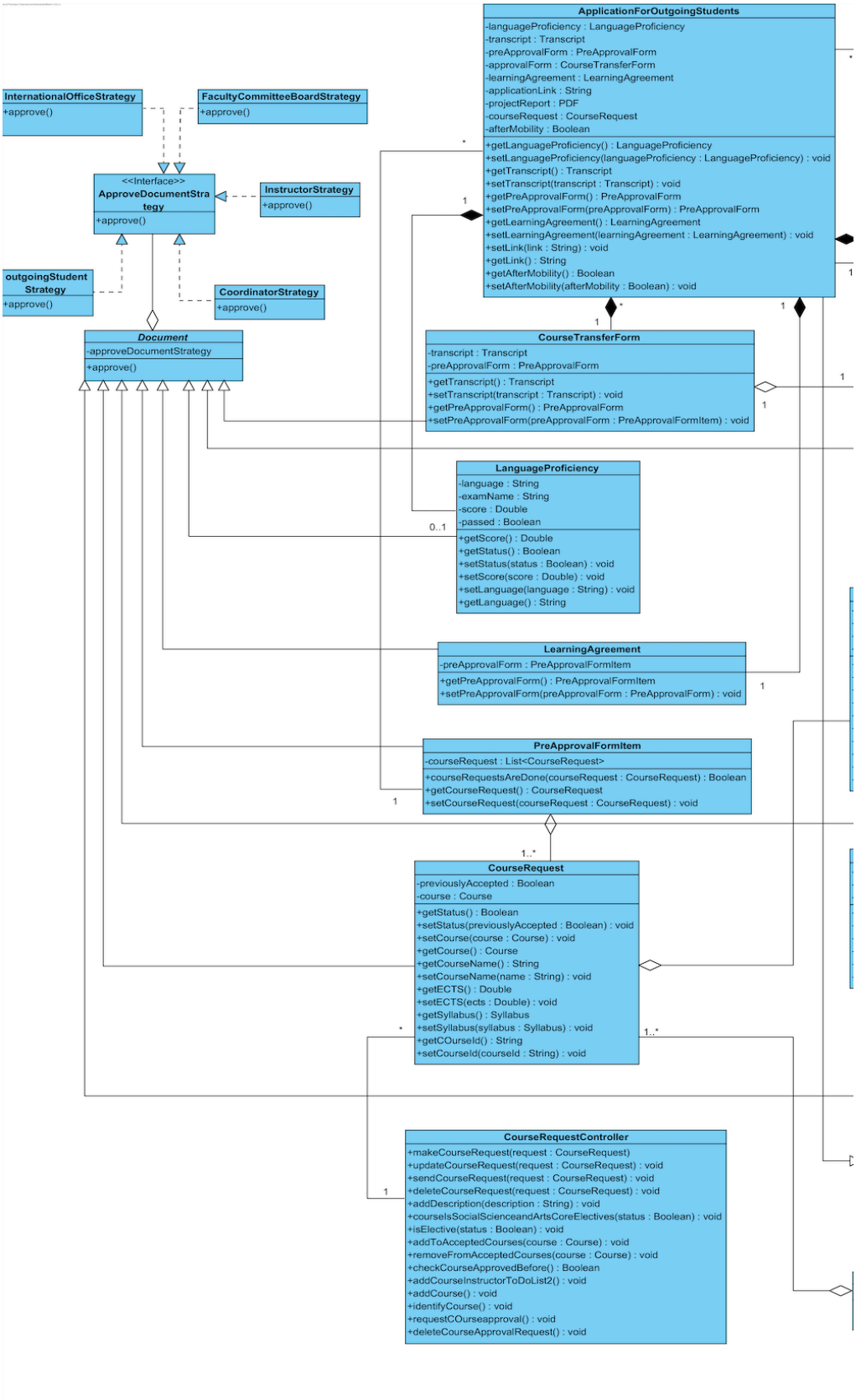

Figure 3.2: Left side of the Final Object Design

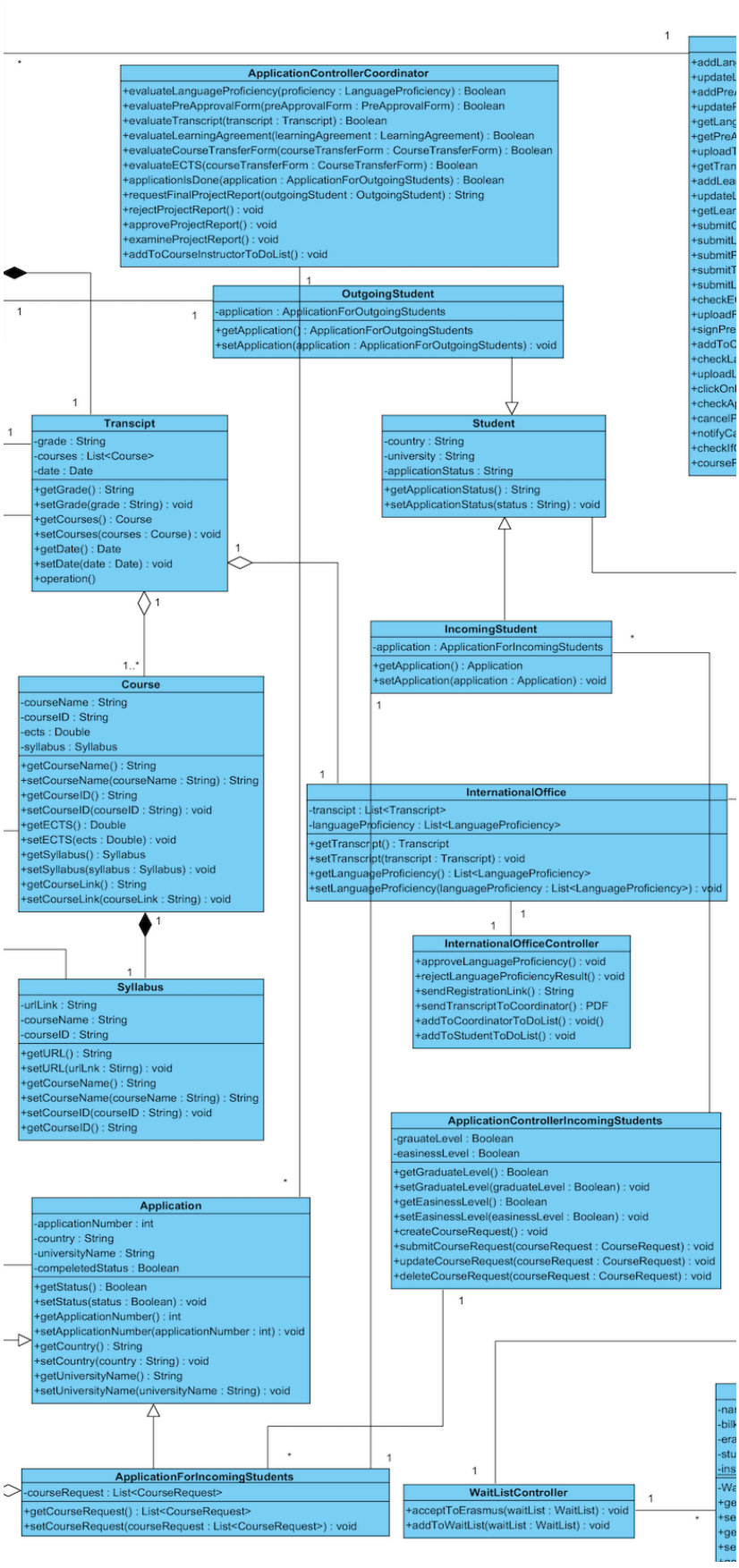

Figure 3.2: Middle side of the Final Object Design

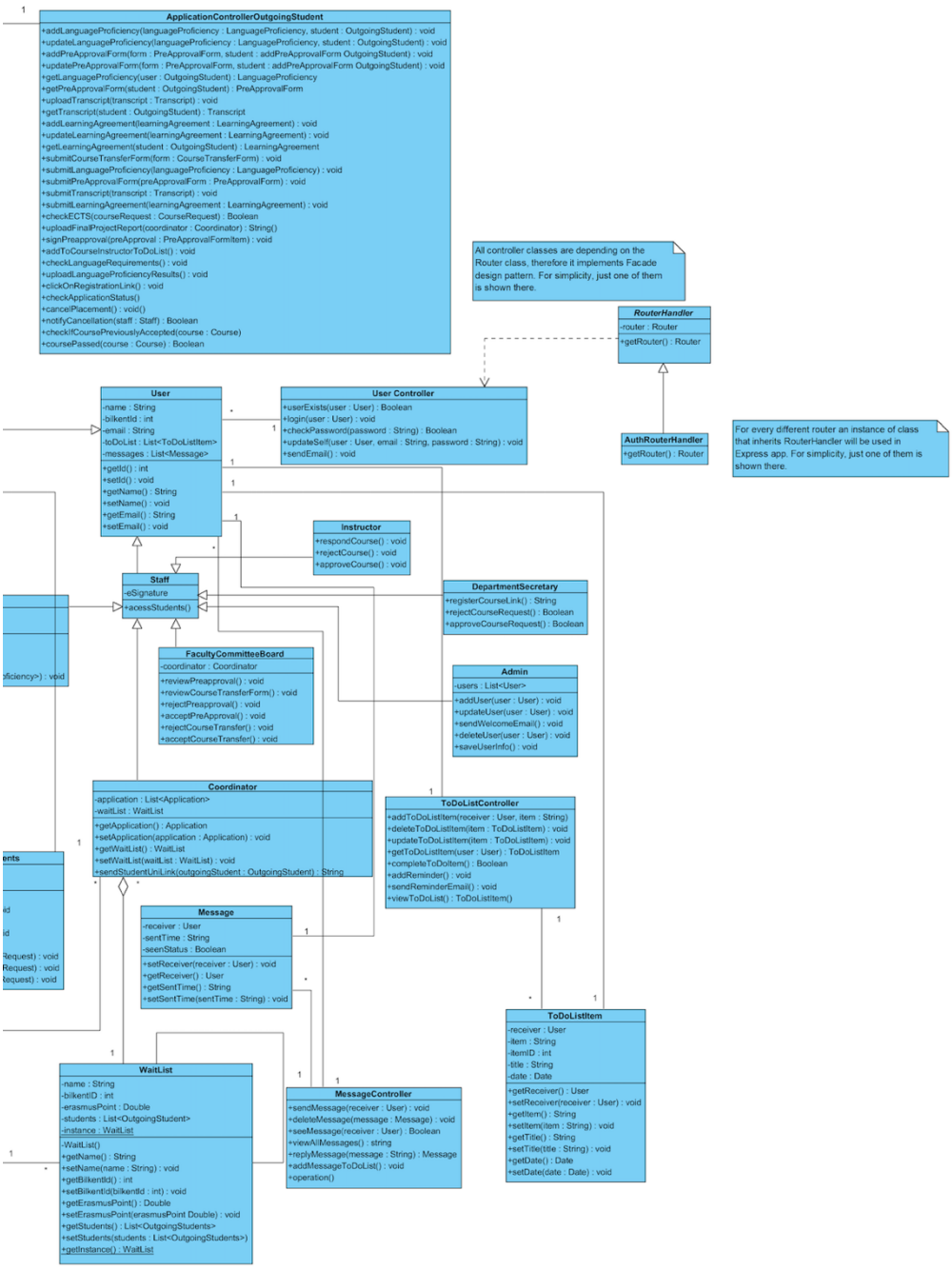

Figure 3.2: Right side of the Final Object Design

3.2.2. Design Patterns

3.2.2.1. Approve Document Strategy Pattern:

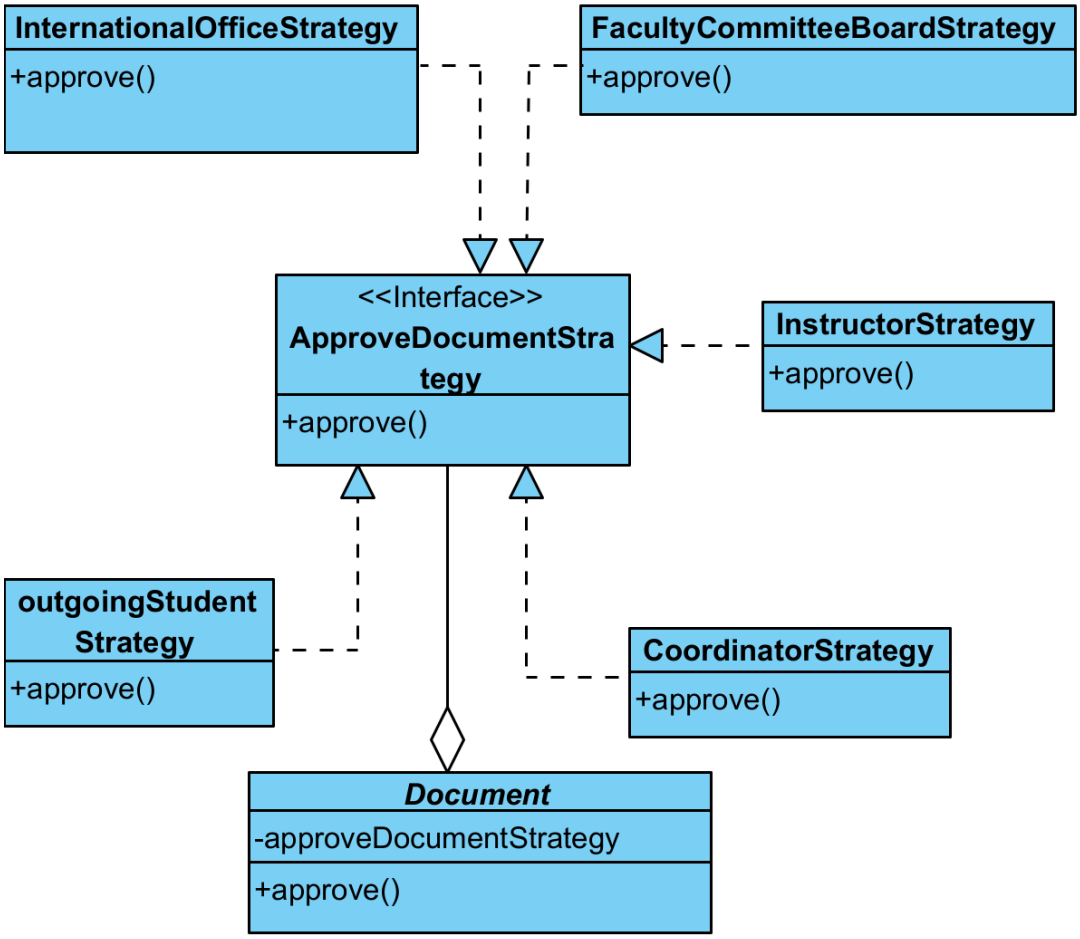

Figure 3.2.2.1: Approve Document Strategy Pattern

Strategy design pattern is used to enable the selection of the approve document algorithm at runtime, based on the role of the user. Related users have ability to approve the documents, however, approving the document algorithm changes according to the user's identity. Document, which is an abstract class, has an abstract method approve for approving methods to be overridden by the subclasses. Also, it holds an instance of approveDocumentStrategy interface, which is the way to apply strategy design pattern. 5 different classes are implementing this interface, depending on the user's identity. This way, different users can approve the documents and this process can be changed during the runtime. For instance, if Pre-Approval form

is approved by the coordinator, coordinator cannot approve this form again. However, in the last process of Erasmus, application is approved by the faculty committee board, which they can do even though coordinators cannot do at this period. All in all, strategy design pattern makes the approve document algorithms reusable in all related role classes and also enables some users to choose between strategies with different responsible at runtime during Erasmus process.

3.2.2.2. WaitList Singleton Design Pattern

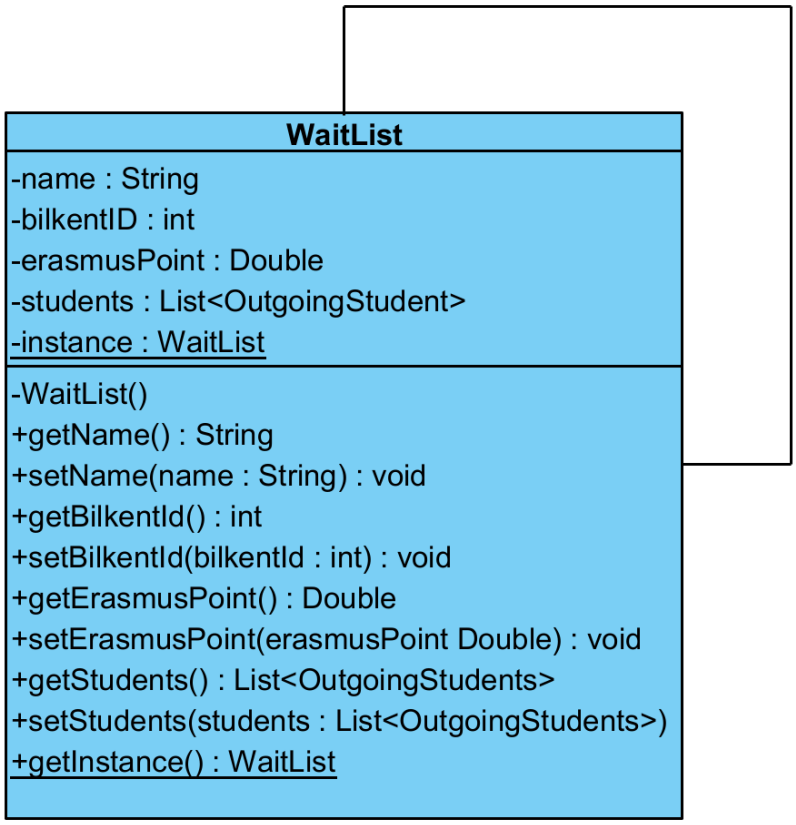

Figure 3.2.2.2: WaitList Singleton Design Pattern

Singleton is a creational design pattern that lets someone to ensure that a class has only one instance, while providing a global access point to this instance. In the Bilasmus application, there is only one waitlist. Regardless of the content of the waitlist (which includes students who are waiting to have a right to get into Bilasmus app, if someone cancels his/her placement), there is always just one list. This class's constructor should be private, and this class should include abstract instance type of WaitList to implement Singleton design pattern. Also, this class is connected to itself to use itself during the Erasmus process.

3.2.2.3. Router Handler Façade Design Pattern

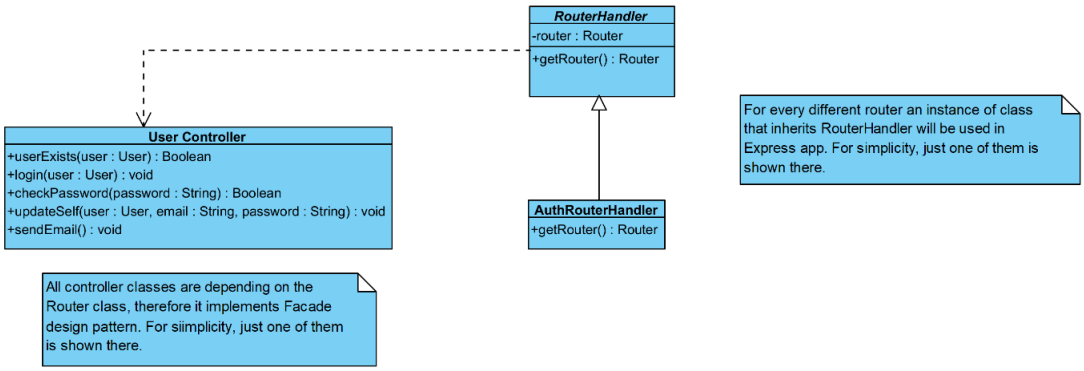

Figure 3.2.2.2: Router Handler Façade Design Pattern

Facade is a structural design pattern that provides a simplified interface to a library, a framework, or any other complex set of classes. As Bilasmus application is being developed on a web server, to handle requests coming from/to server should be handled by the routers. In Node.js, routers are handled with Express. Router objects belong to sophisticated Express framework. To handle routers for functionalities of the program, an abstract class named RouteHandler is created, which has getter method for the router object. This abstract class is connected to all controller classes in the program, but for the simplicity, just UserController class is shown there. For every functionality, there should be a new class to handle router for that function. These classes must inherit abstact class HandleRouter. For simplicity, just AuthHandler is shown there. This class implements the abstract method coming from its parent.

3.2.3. User Interface Management Layer

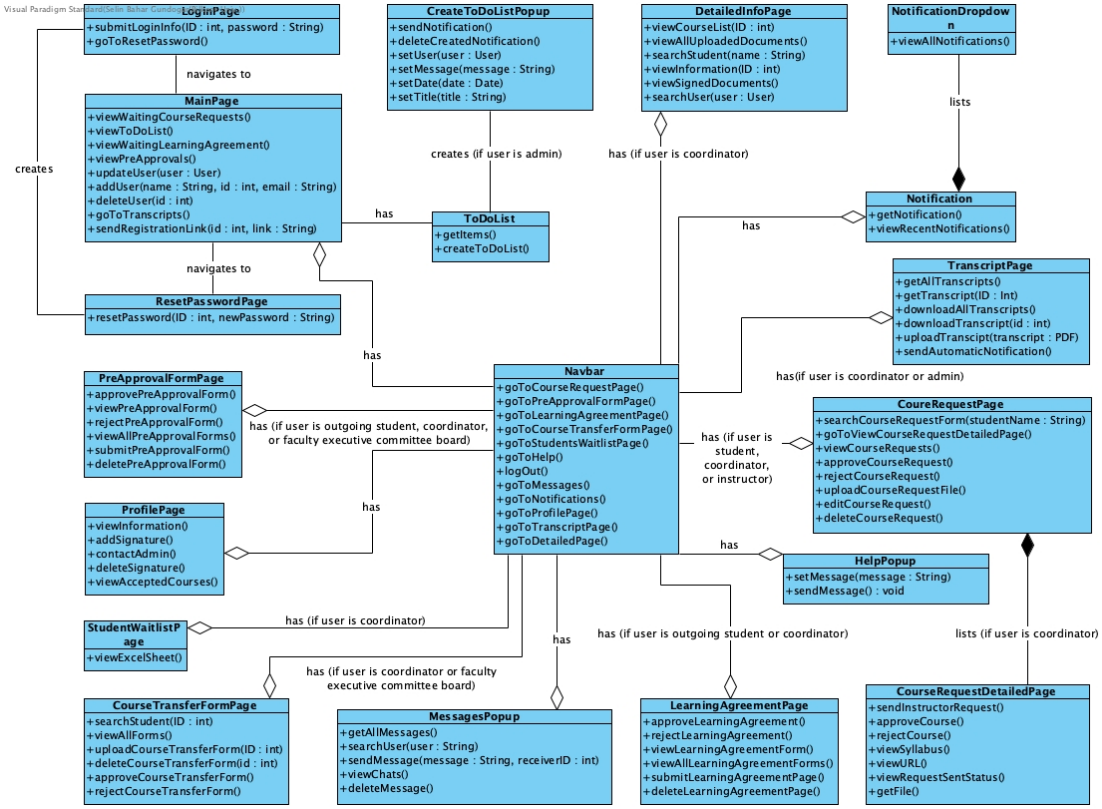

Figure 3.3.1: User Interface Management Layer

3.2.3.1. User Interface Class Explanations

LoginPage

The LoginPage is the initial page that opens up when the user opens the application. The user can go to the MainPage after logging in successfully or can change his/her password by going to the ResetPasswordPage.

Methods

submitLoginInfo(ID:int, password:String): Sends the inputs to the backend to authenticate if the user entered information correctly.

Other methods are methods that are implemented to navigate the user to the MainPage and ResetPasswordPage.

ResetPasswordPage

The ResetPasswordPage is accessed through the LoginPage or the registration link that is sent to the user through email. Once the user enters the new password, he/she is directed to the MainPage.

Methods

resetPassword(ID:int, password:String): Sends the inputs to the backend to update the backend.

Other methods are methods that are implemented to navigate the user to the MainPage.

MainPage

The MainPage is accessed through the LoginPage or the ResetPasswordPage and it contains the information related to the specific type of user. Different type of users has different user interfaces; though, all have a navbar that the user utilizes to move within the application.

Methods

viewWaitingCourseRequests(): Displays the awaiting course requests that are coming from students. Coordinator, instructor, department secretary specific.
viewToDoList(): Displays the to-do list of users.
viewPreApprovals(): Displays the Pre-Approval forms to the user. Specific to the coordinator and faculty executive committee board.
updateUser(name:String, id:int, email:String): Updates the information of a user. Specific to the admin user.
addUser(name:String, id:int, email:String):Adds a user to the application and an email is sent to the added user's email. Specific to the admin user.
deleteUser(id:int):Deletes a user. Specific to the admin user.
goToTranscripts():Takes the coordinator to the sent transcripts page. Specific to the coordinator.
sendRegistrationLink(id:int, link:String):Sends Erasmus registration link to the outgoing students. Specific to the admin.

PreApprovalPage

The PreApprovalPage is accessed through the Navbar. The user interface displays a page for the Pre-Approval document to be uploaded, updated, deleted, or viewed and approved/rejected. This page is only available to the outgoing students, coordinator, and the faculty executive committee board.

Methods

approvePreApprovalForm(): Approves the Pre-Approval Form. Specific to the coordinator and faculty executive committee board.
viewPreApprovalForm(): Displays the Pre-Approval Form. Specific to the outgoing student, coordinator, and executive committee board.
rejectPreApprovalForm(): Rejects the Pre-Approval Form. Specific to the coordinator and faculty executive committee board.
viewAllPreApprovalForms(): Displays all Pre-Approval Forms. Specific to the outgoing student, coordinator, and executive committee board.
submitPreApprovalForm(): Submits PreApproval Form. Specific to the outgoing student.
deletePreApprovalForm(): Deletes the uploaded Pre-Approval Forms. Specific to the outgoing student.

ProfilePage

The ProfilePage is accessed through the Navbar. The ProfilePage consists of displaying information about the user.

Methods

viewInformation(): Loads the information of a user onto the page.
addSignature(): User uploads a PDF of his/her signature.
contactAdmin(): Sends a request directly to the admin.
deleteSignature(): Deletes the PDF signature that was uploaded.
viewAcceptedCourses(): Displays the excel sheet of all accepted courses when the user clicks on the previously accepted courses button.

StudentWaitlistPage

The StudentWaitlist is accessed through the Navbar. This page is only available to the coordinator.

Methods

viewExcelSheet(): Displays the excel sheet of all students that are on the Erasmus wait list.

CourseTransferFormPage

The CourseTransferPage is accessed through the Navbar. This page is only available to the coordinator and the faculty executive committee board. This page contains the course transfer forms.

Methods

searchStudent(id:int): Returns the searched student's course transfer form.
viewAllForms(): Displays all uploaded course transfer forms.
uploadCourseTransferForm(id:int): Sends the course transfer form of an outgoing student. Specific to the coordinator.
deleteCourseTransferForm(id:int): Delete the uploaded course transfer form. Specific to the coordinator.
rejectCourseTransferForm(): Approves a course transfer form. Specific to the faculty executive committee board.
approveCourseTransferForm(): Rejects a course transfer form. Specific to the faculty executive committee board.

Navbar

The navbar is accessed in all pages for all users. However, the elements of a navbar changes based on user type. Also, navbar allows navigation to each page listed as navbar element.

Methods

All functions starting with goTo…() are to navigate to the keyword after the phrase "goTo". These are items listed on a navbar and functions change based on the user type.
logout(): safely exits the user from the system.

ToDoList

ToDoList is accessed from MainPage and is a shared UI for all user types. It shows items or activites to be done and let the admin create a new ToDoList.

Methods

getItems(): shows the items or activities to be done as a list provided by admin.
createToDoList(): creates a new ToDoListPopup. Specific to the admin.

CreateToDoListPopup

All features below are specific to admin user type
This UI is navigated from ToDoList of an admin only. It provides features to create a new Item for a user by admin.

Methods

sendNotification(): sends notification to the user.
DeleteCreatedNotification(): removes the notification sent previously if no longer needed.

setUser(user: User): lets the admin decide which user to create the item for.
setMessage(message: String): lets the admin set the message of the content of the item to be created.
setDate(date: Date): lets the admin create a deadline for example by setting a date of the item to be done.
setTitle(): lets the admin set the Title of the Content of the item.

MessagePopup

This popup is provided to all user types. It allows any two user interact via in-system messaging.

Methods

getAllMessages(): gets both new and already sent messages of a chat.
searchUser(user: User): searches for a specified user.
sendMessage(Message: String, receiverID: int): sends a message as a string to a specified userID.
viewChats(): shows a table of all previous chats with different users.
deleteMessages(): deletes messages.

DetailedInfoPage

All features below are specific to coordinator user type

This page shows a detailed info of a student to the coordinator if clicked on navbar. This page is an item of a navbar of coordinator only.

Methods

viewCourseList(ID: int): shows a list of courses of a student specified by ID.
viewAllUploadedDocuments(): shows all uploaded documents of all students
seacrhStudent(user: User): seacrhs for a specified student as a user.
viewInformation(ID: int): shows information about a specified student.
viewSignedDocuments(): shows all signed documents in the system.

Notification

A UI for all notifications which is navigated from navbar for all users.

Methods

getNotification(): gets notification of a specified user.
viewRecentNotifications(): shows all recent notifications.

NotificationDropDown

A list dropdown for all notifications.

Methods

viewAllNotifications(): shows all notifications.

TranscriptPage

All features below are specific to coordinator or admin user type
A page to upload a PDF transcript of a student.

Methods

getAllTranscripts(): gets all transcripts that were uploaded and saves to the system.
getTranscript(ID: int): gets the transcript of a specified student and saves to the system.
downloadAllTranscripts(): downloads transcript of all students in the system. Specific to the coordinator.
downloadTranscript(ID: int): downloads transcript of a specified student in the system. Specific to the coordinator.

uploadTranscirpt(transcript: PDF): uploads a pdf transcript of a specified student in the system. Specific to the admin.
sendAutomaticNotification(): sends an automatic notification. Specific to the coordinator.

CourseRequestPage

All features below are specific to coordinator, instructor, or a student user type

A page to request a new course or approve the requested course based on the user type.

Methods

searchCourseTransferForm(studentName: String): searches for a specified student's course transfer form. Specific to the coordinator.
goToViewCourseTransferForm(): navigates to a second page(CourseRequestDetailedPage) for a detailed list of requested courses. Specific for the coordinator.
viewCourseRequest(): shows information about a requested course. Specific to the coordinator and instructor.
approveCourseRequest(): approves a course request. Specific to the coordinator and instructor.
rejectCourseRequest(): rejects a course request. Specific to the coordinator and instructor.
uploadCourseRequestFile(): adds a new course request file. Specific to the
editCourseRequest(): updates a course request accordingly. Specific to the student.
deleteCourseRequest(): removes a course request accordingly. Specific to the student.

CourseRequestDetailedPage

*All features below are specific to coordinator *

A detailed course request page provided to the coordinator. It also allows an interaction between the instructor and the coordinator.

Methods

sendInstructoreRequest(): sends the request to the Instructor to get validation of a new course.
approveCourse(): approves a course.
rejectCourse(): rejects a course.
viewSyllabus(): shows syllabus of a course.
viewURL(): shows a url of a specified course for detailed information.
viewRequestSenStatus(): shows the status of a course request.
getFile(): shows general information as a list for a course specified for a student.

HelpPopup

*All features below are specific to all user types *

A help popup to assist users if a problem exists.

Methods

setMessage(message: String): shows a blank box to write a message.
sendMessage(): sends the written message to the responsible.

learningAgreementPage

All features below are specific to coordinator or an outgoing student

A page to download, upload, reject, approve, or view learning agreement(s) based on the user type.

Methods

approveLearningAgreement(): approves a learning agreement. Specific to the coordinator.

rejectLearningAgreement(): rejects a learning agreement. Specific to the coordinator.

viewLearningAgreementForm(): shows a learning agreement. Specific to the coordinator and the student.

viewAllLearningAgreementForms(): shows all learning agreements as a list. Specific to the coordinator.

submitLearningAgreementPage(): lets the student upload a leaning agreement.

deleteLearningAgreementPage(): lets the student remove a leaning agreement.

3.2.4. Web Server Management Layer

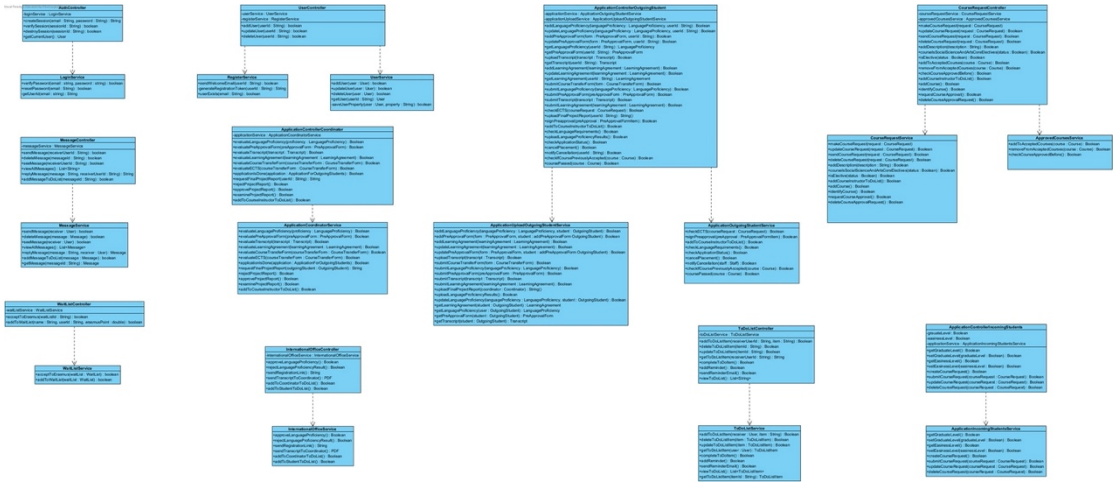

Figure 3.3.2: Web Server Management Layer

This represents the web server management layer of our project. It links the User Interface Layer and the data management layer and is responsible for all requests to get connected to each other between the frontend and the backend.

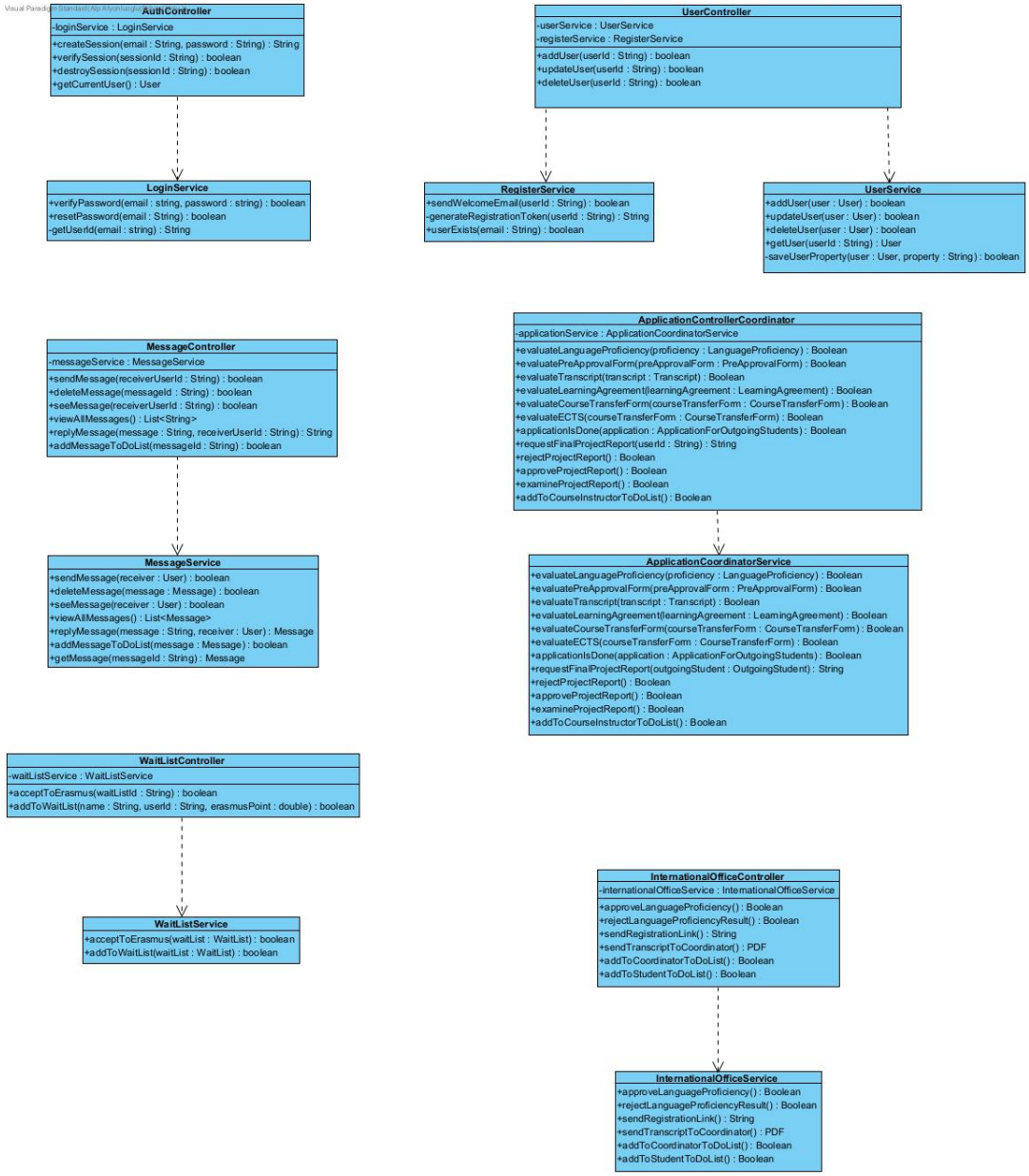

Figure 3.3.2: Left side of Web Server Management Layer

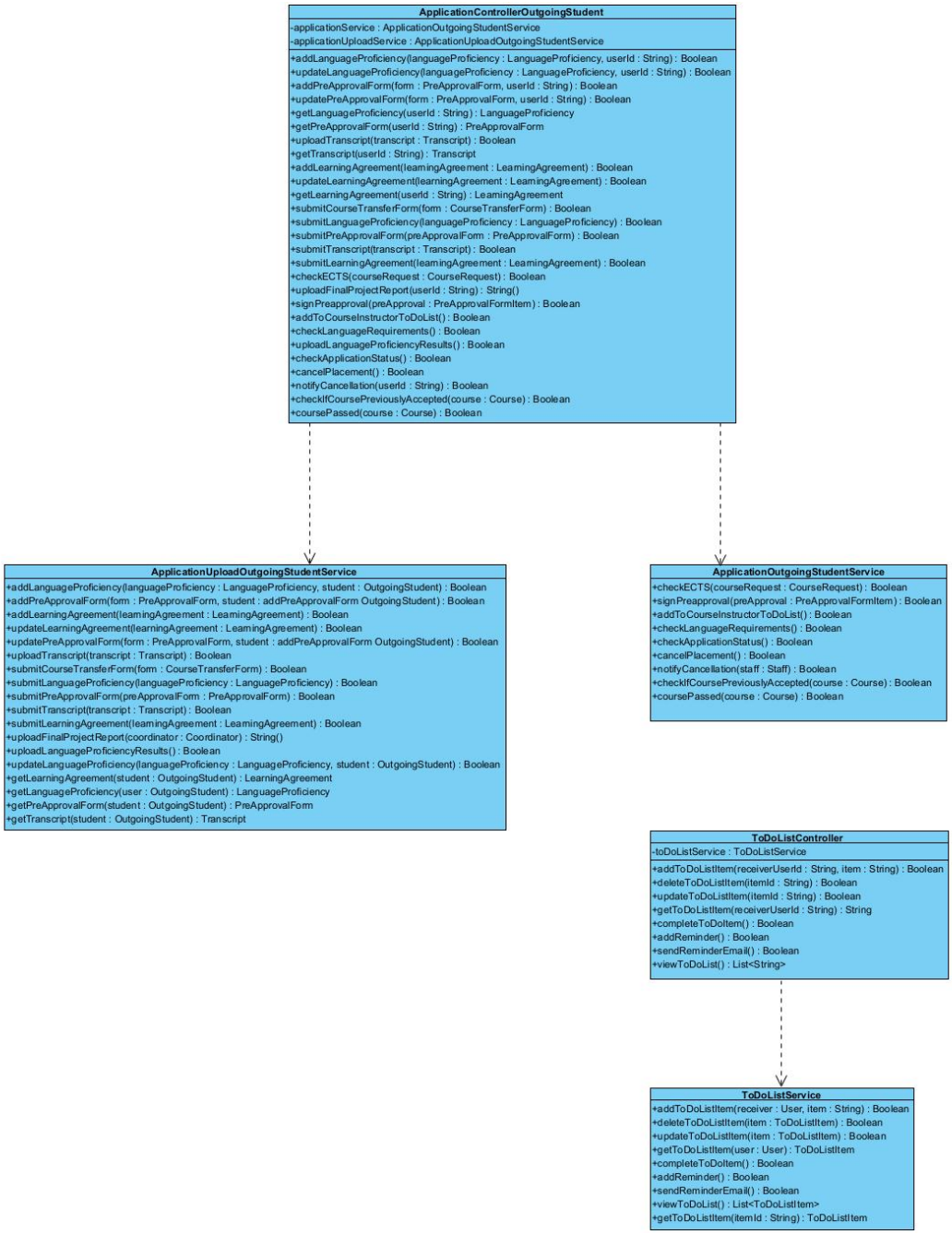

Figure 3.3.2: Middle of Web Server Management Layer

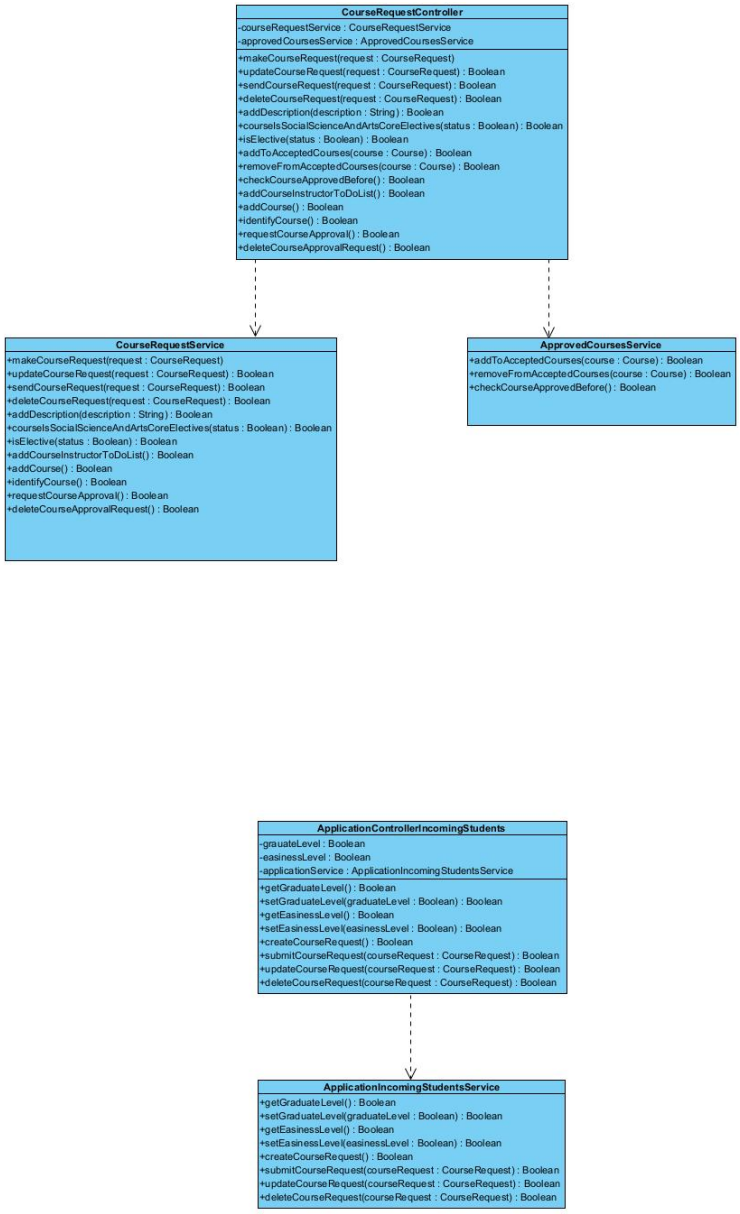

Figure 3.3.2: Right side of Web Server Management Layer

3.2.5. Data Management Layer

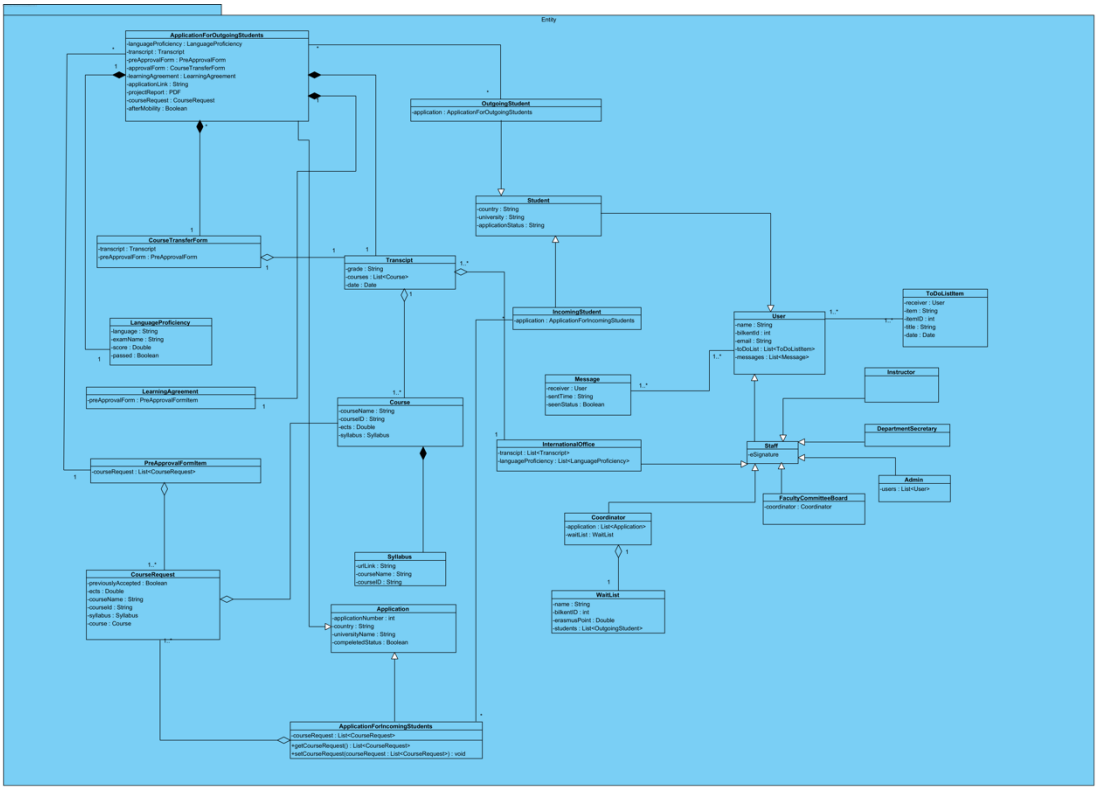

Figure 3.3.3: Entity Classes

In Figure 3.3.3, in addition to what have been shown in the figure, each class have empty and parametrized constructors and getter and setter methods of attributes. Note that operations are not shown in the figure for simplicity.

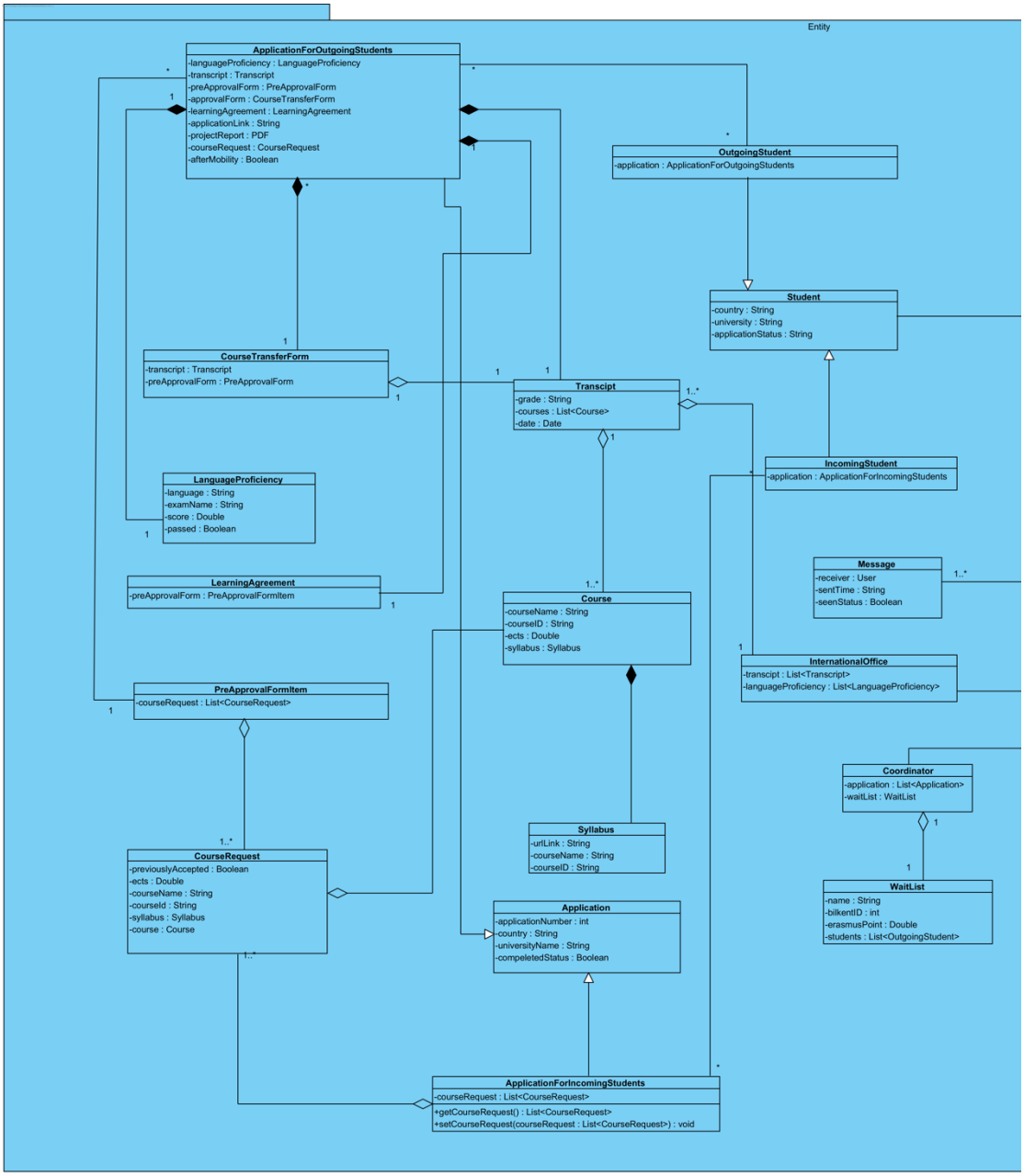

Figure 3.3.3: Left side of Entity Classes

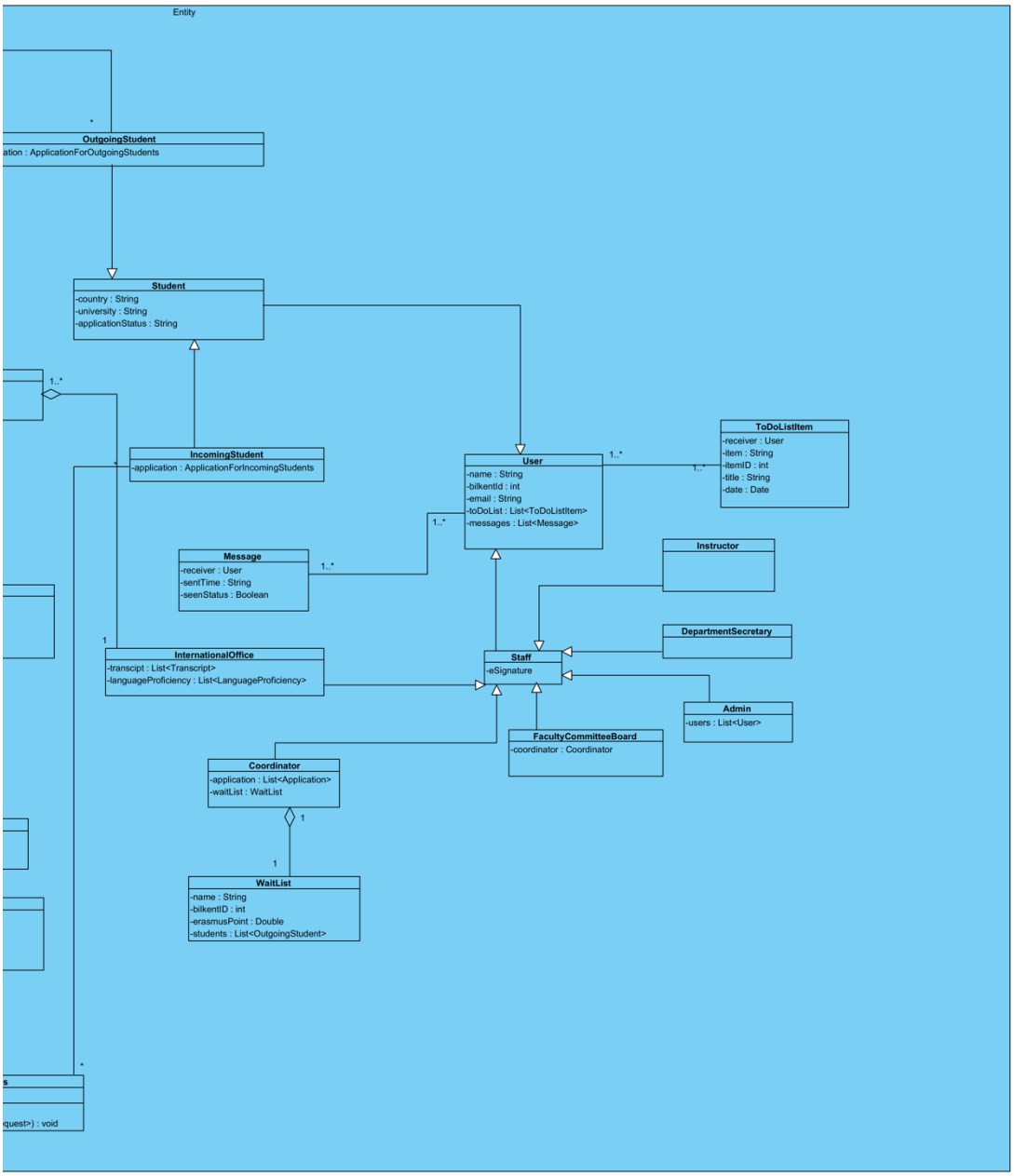

Figure 3.3.3: Right side of Entity Classes

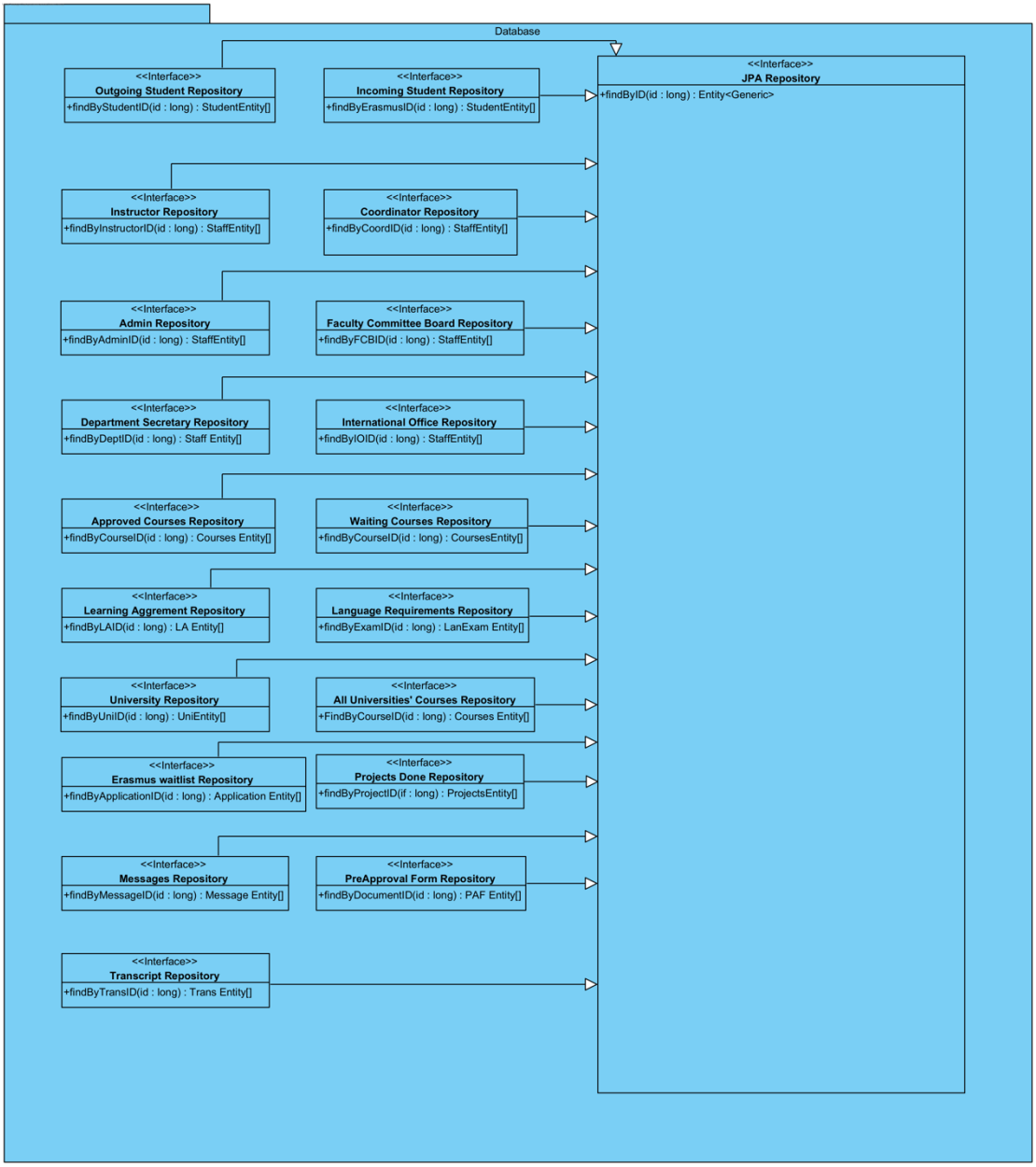

Figure 3.3.3: Data Management Layer

This layer represents the Data Management Layer and it consists of the database. This layer handles all requests and changes of the database by inheriting the JPA Repository in its repository interfaces.

3.3. External Packages

3.3.1. Frontend

3.3.1.1. Vue-pdf

This package allows pdf files to be downloaded and uploaded on the website.

3.3.1.2. Vue-signature

This package allows users to sign documents with an e-signature.

3.3.1.3. Vue-upload-image

This package allows users to upload images on the website to update profile photos.

3.3.2. Backend

3.3.2.1. express

Express is a web application framework built to be used with Node JS.

3.3.2.2. express-subdomain

This package allows creating and managing subdomains and related routing.

3.3.2.3. http-errors

This package allows creating and handling error cases.

3.3.2.4. express-session

This package creates sessions once a user logs in, and stores session-related data separately for each user.

3.3.2.5. node-postgres

This package will be used to communicate with the postgresql database.

3.3.2.6. express-fileupload

This package will be used to handle the server-side of a file-upload task and save the files that are sent via a form.

3.3.2.7. pbkdf2-password

This package will be used to calculate hash values of passwords.

3.3.2.8. nodemailer

This package will be used to send automated emails.

3.4. Internal Packages

3.4.1. routes

The routes package contains router files that map API endpoints with controller functions.

3.4.2. public

This package contains separate sub-packages/sub-folders for static resources, such as css files and image/document resource files.

3.4.3. controllers

This package contains controller classes that are called by routers or other controllers. Files in this package call service classes when required, and are used to determine the structural logic of how the classes work.

3.4.4. services

This package contains service classes that handle the async tasks done in the background. Classes of this package are called by controllers when a task, that is generally long and takes much time, is chosen to be passed to the background services.

3.4.5. models

This package contains model classes that determine data that need to be stored for a class type, in a dedicated database table.

3.4.6. database

This package contains database-related classes that are used when querying database entries to insert, select, update, or delete data.

4. Improvement Summary

Changes based on the feedback were corrected in the second iteration of the Design Report. Paragraph format was changed to justified and the subsystem diagram was modified and its description was added. A new deployment diagram was added to the report and details of the Hardware/Software Mapping section was included. Moreover, 2.5 Persistent Data management section was improved and an access matrix was added to the 2.6 Access Control and Security section. Additionally, 2.7 Boundary Conditions section was modified based on the feedback and the User Interface Management Layer was redone. Description of the User Interface Management Layer was added below its diagram. Furthermore, the entity-controller diagram of the 3.2.1 Final Object Design section was redesigned to include design patterns. All design patterns were explained in detail below the diagram.

5. Glossary & references

[1] https://app.erasmus.bilkent.edu.tr/
[2] https://w3.bilkent.edu.tr/www/degisim-programlari/