Bilkent University

Department of Computer Engineering

Senior Design Project

ErasmusAga

Group Number 2B

Project Design Report - Iteration 2

Group Members:

Doruk Altan 21401362 doruk.altan@ug.bilkent.edu.tr

Asım Bilal Ak 21802887 bilal.ak@ug.bilkent.edu.tr

Gökberk Altıparmak 21901798 g.altiparmak@ug.bilkent.edu.tr

Furkan Yıldırım 21902514 f.yildirim@ug.bilkent.edu.tr

Alp Tuğrul Ağçalı 21801799 tugrul.agcali@ug.bilkent.edu.tr

Instructor: Eray Tüzün

Teaching Assistant: Mert Kara

1. Introduction 4
1.1 Design Goals 4
1.1.1 Simplicity 4
1.1.2 Security 5
1.1.3 Maintainability 5
1.1.4 Performance 5
2. System Architecture 5
2.1. Subsystem Decomposition 5
2.2. Hardware/Software Mapping 8
2.3. Persistent Data Management 9
2.4. Access Control and Security 9
2.5. Boundary Conditions 11
2.5.1. Initialization 11
2.5.2. Termination 11
2.5.3. Failure 11
3. Low Level Design 11
3.2 Final Object Design 13
3.3 Layers 17
3.3.1 User Interface Management Layer 17
3.3.2. Web Server Subsystem 17
3.3.3 Data Management Layer 20
3.4 Design Patterns 21
3.4.1 Facade Design Pattern 21
3.4.2 Builder pattern 21
3.5 Packages 22
3.6.1 User Interface Layer Class Interfaces 23
3.6.1.1 Login 23
3.6.1.2 AdminNavBar 24
3.6.1.3 AdminProfilePage 24
3.6.1.4 AdminUsersPage 25
3.6.1.5 AdminUniversitiesPage 26
3.6.1.6 AdminApplicationPage 26
3.6.1.7 StudentProfilePage 27
3.6.1.8 StudentNavBar 27
3.6.1.9 StudentApplicationPage 28
3.6.1.10 CourseCoordinatorProfilePage 28

3.6.1.11 CourseCoordinatorNavBar 29
3.6.1.12 CourseCoordinatorCoursesPage 29
3.6.1.13 AdministratorProfilePage 30
3.6.1.14 AdministratorNavBar 30
3.6.1.15 AdministratorApplicationPage 31
3.6.1.16 Erasmus administratorUniversitiesPage 31
3.6.2 Web Server Layer Class Interfaces 32
3.6.2.1 Application User LoginService 32
3.6.2.2 ApplicationOperationsController 32
3.6.2.3 ApplicationOperationsImp 33
3.6.2.4 ManageProfileController 34
3.6.2.5 ManageProfileImp 35
3.6.3 Data Management Layer Class Interfaces 35
3.6.3.1 User 35
3.6.3.2 AppAdmin 36
3.6.3.3 Erasmus Administrator 36
3.6.3.4 Course Coordinator 36
3.6.3.5 Student 37
3.6.3.6 Comment 37
3.6.3.7 Comment Type 38
3.6.3.8 Application 38
3.6.3.9 Duration 39
3.6.3.10 Status 39
3.6.3.11 StatusType 40
3.6.3.12 University 40
3.6.3.13 Course 41
3.6.3.14 Department 41

4. Improvement Summary 42
4.1 General 42
4.2 Low-level Design 42
4.3. Glossary and References 42

1. Introduction

ErasmusAga is a web based application. In that system, Bilkent University students will control their applications. Also, they can see their missing information and they can upload their missing information thanks to pdf files. Moreover, students can communicate with administrators via the comments in ErasmusAga. For administrators and course coordinators To-Do list is provided by the program for checking their works for that time period. Also administrators can request a file which is needed for a specific student and also, students can communicate with course coordinators via comment and can upload that syllabus of the wanted course. Course coordinators can respond according to whether it is appropriate or not for the requirements of the Erasmus in Bilkent.

1.1 Design Goals

In the ErasmusAga application our design will be very simple and straightforward. Performance of our application will be high because of the CUBA platform.

1.1.1 Simplicity

In the Erasmus application simplicity is enhanced because of the design of the program. Thanks to the Cuba platform which uses the most common design of web sites design understanding of program flow will be better. In addition to the platform, the design of ErasmusAga includes many options in the same page. For example, Erasmus administrators navigate features from the left dashboard. Also, when Erasmus administrator logs in, he/she directly sees the To Do list and almost all duties can be done in this to do list without navigating any dashboard activities.

1.1.2 Security

For this application security is crucial because all of the student information will be stored in that application so any malicious action will be problematic because this application also serves for Bilkent University. In addition to this, Application will only provide pdf documents for uploading so any problematic files will be blocked by default.

1.1.3 Maintainability

For ErasmusAga application maintainability is vital because it should store data and delete data for a long time. It is because this application will be used in the future. Because of CUBA platform authorization can be given easily by admin without any piece of code. In future authorization problems can be solved by this feature.

1.1.4 Performance

In this application we use a simple interface so the program will not be forced to load huge data for the interface, this does not mean the interface will be ugly. Also, Our database system can store data compatible with OOP so applications can store data as an object. Therefore Any request from the database will require a short time.

2. System Architecture

2.1. Subsystem Decomposition

This section is dedicated to the process of decomposing the system into minor subsystems so that the design of the system is easily understandable. Breaking the problem into smaller modules, we can progress much faster in the initial implementation or any future modifications, therefore allowing us to meet our maintainability expectations. We will

decompose our system into a three layered architecture which consists of the Interface Layer, Web Server Layer, and the Data Layer.

The Interface Layer is responsible for displaying information to the user and allowing the user to interact with the system. This layer serves as the boundary object that contains the interface classes which make up the web pages of the application. Pages are bound to their respective controllers in the Web Server Layer that process the desired interactions from the user.

The Web Server Layer is responsible for managing interactions with users. It contains the controller and service classes for their respective counterparts in the Interface Layer. This layer is where the operations take place that are needed to fulfill the requested interactions of the user. For instance, when the user indicates that he wants to upload a file by interacting with the File Upload Interface in the Interface layer, the File Upload Controller in the Web Server Layer manages this interaction by applying the required functions.

The Data Layer contains the database and is responsible for maintaining the data of the system. It also includes the Entity subsystem that contains the persistent entity classes. The Web Server Layer collaborates with this layer to perform its functionalities. Data Layer either sends data to the Web Server Layer for use in a method or receives data to store.

Notice that every layer in the subsystem decomposition only communicates with other layers that are its direct hierarchical parent or child. In other words, The Interface Layer cannot directly interact with the Data Layer, it must first go through the Web Server Layer. This design choice is to make the system more stable, modifiable and extendable. The model below is a visual representation of the proposed system decomposition.

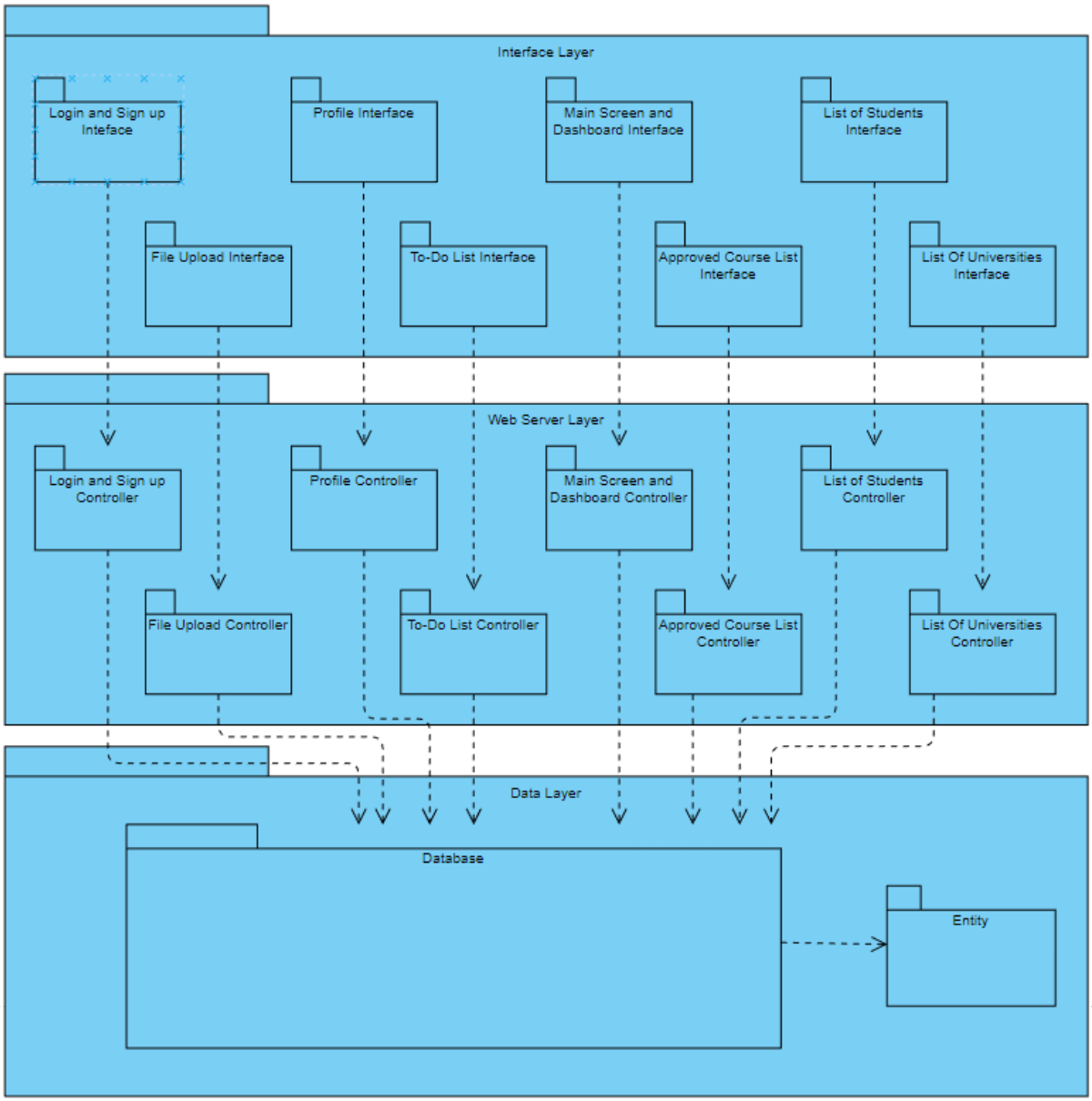

Figure 1 Subsystem Decomposition

2.2. Hardware/Software Mapping

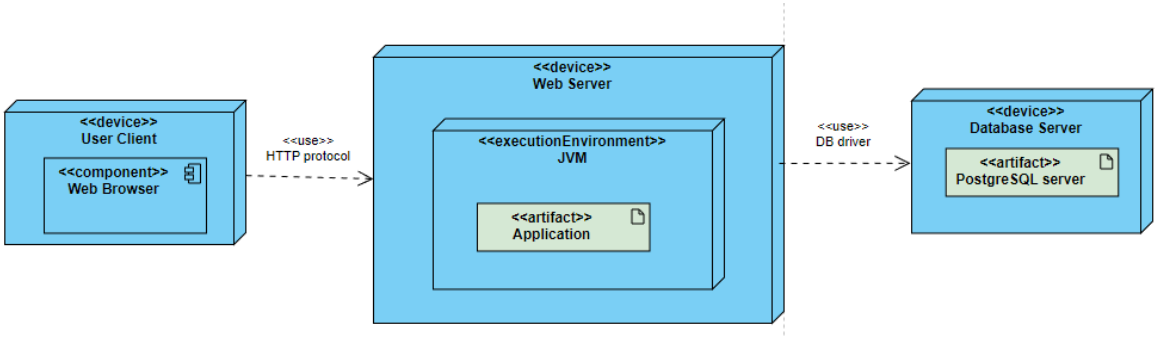

Figure 2 Deployment Diagram

The programming language we use is Java 8 and the framework we use is the Cuba Platform. This project will work on all operating systems such as Mac, Windows, Linux but requires a Java Runtime Environment (JRE). The minimum processor requirement for JRE is a Pentium 2 266 MHz processor and 181MB of disk space. This is a web based project, therefore, web browsers are required for execution on any machine. CUBA platform is compatible with all popular browsers including Google Chrome, Mozilla Firefox, Safari, Opera 15+, Internet Explorer 11, Microsoft Edge. When it comes to frontend, we are using xml and for the database, we are using PostgreSQL. All the current popular web browsers are supported but due to some APIs and libraries, older versions of browsers may be incompatible such as Internet Explorer 8. Also, the CUBA platform supports PostgreSQL for version 8.4 and higher so previous versions will be incompatible.

Our project does not require any specialized hardware components to run successfully. To interact with the system, users only need the common I/O hardware such as a mouse, keyboard and monitor. It is also assumed that the machines have the hardware capability to run the aforementioned web browsers.

2.3. Persistent Data Management

Main data storage device of our project will be a database. We decided to use PostgreSQL as our database since it is a free and open-source system and we have at least some level of familiarity with it compared to other popular database options that also provide SQL compliance and object relation support. The fundamental role of the database will be to set up our objects as tables in the database. These tables include users, universities, courses, uploaded files, grades and roles. Users will be able to edit said tables by invoking the functionalities in the Web Server Layer through interactions with the interfaces in the Interface Layer. The resulting changes are recorded in the database that resides in the Data Layer.

2.4. Access Control and Security

The first way we impose security on our system is the user authentication process. In this step we check the registered credentials of a user to the input credentials during login attempts. If the login is successful, we identify the type of the user, for instance student or Erasmus administrator, and give them the proper permissions. The amount of information users can attain depends on their permission level. Higher permission users can reach more data and functionality compared to lower permission users. This way we ensure that data is only reachable by authorized parties.

Some users do not have the permission to change some data but have the permission to view it. In such cases, it is a point of emphasis in our project to filter any information that should only be viewed by higher permission users. Such information may be passwords or unrelated personal details of users. Lastly, the passwords we keep are all hashed for security purposes.

	Student	**Erasmus administrator**	**Course Coordinator**	**Admin**
Login	x	x	x	x
View approved courses	x	x	x	x
Make comments	x	x	x	x
Change password	x	x	x	x
View Profile	x	x	x	x
Cancel application	x	x		x
Upload pre-approval	x			x
Upload file	x	x		x
Download file	x	x	x	x
View University Requirements	x	x		x
View Student Profile		x		x
View applied university of student		x		x
Change university courses and information		x		x
Edit university requirements		x		x
Add view edit universities		x		x
Confirm application		x		x
Create User				x
Assign change roles				x
Edit user role				x

2.5. Boundary Conditions

2.5.1. Initialization

Ours is an application running on a web server at all times, thus it does not require any installation process. An internet connection and a registered account is everything one needs to use the application. During login, the entered credentials are checked if they are bound to an account in the database. If successful, the user is granted access to all functionalities that are within the scope of their permission. If login is unsuccessful, the user is only granted a restricted view that includes login and signup pages. The information on the pages are all initialized from the database.

2.5.2. Termination

Our system is constantly running and all subsystems are bound to each other. Any spontaneous termination or intentional termination via admin command will result in the entirety of the system terminating.

2.5.3. Failure

Any failures in the system will be handled by our custom handler class which is extended from the DefaultExceptionHandler that CUBA platform offers. If there are any errors during interactions with the database or in tasks like uploading files, the handler will be invoked, developers will be notified and the user will be redirected to the page with an error message.

3. Low Level Design

3.1 Object Design Trade-offs

Memory versus Performance

Because ErasmusAga uses object oriented design, the program will be a little bit slow compared to non-object oriented programs. However, object oriented programming allows

useful memory control. Because a program can store data in object form and tracing, adding and deleting data will be easy because of that approach. Therefore, any adding new data does not require any piece of code. Moreover, adding an object such as student will be easy because it allows directly allocating students data features such as ID number, university and name…

Maintainability versus Performance

ErasmusAga is created with CUBA platform so entity addition and deletion will be easy because of CUBA platform features. Moreover, because ErasmusAga was created by OOP principles, maintainability will be high since any other coder will understand the code easily, so if new coders understand the old code, fixing and software development will also be easy.

Security versus Usability

ErasmusAga allows users only to upload pdf files (only Erasmus administrators can import excel files) so any other file extensions which are dangerous for the software will be blocked. However, this type of feature comes with some problems. For example, users need to change all their files to pdf versions to upload files. Users need to make a small effort but it allows the program to more advantage. It is because in addition to security, Erasmus administrators also can easily control the documents because all of the files will be pdf, documents of a student will be organized and only pdf programs are needed for those files. Today's computer system nearly all of web browsers has pdf reading options so it does not require any other special programs and even many computers have adobe pdf software.

Functionality versus Usability

In ErasmusAga there are many features even if we make them simple. So there can be reduced usability because of many functions. However, because our system is very simple, over a time users will get used to the system thanks to simple design.

3.2 Final Object Design

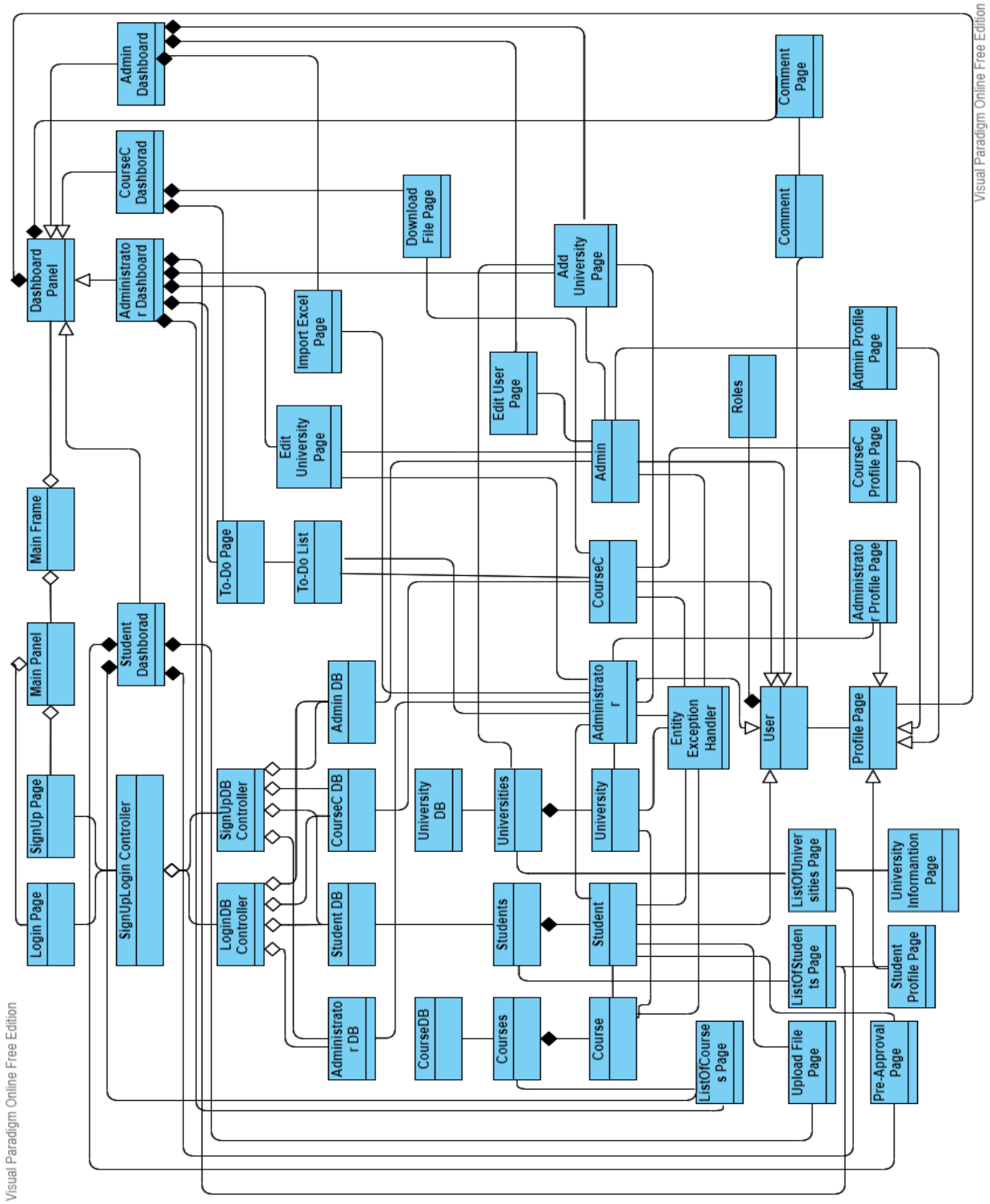

Figure 3 Final Object Design

Left Side of the Diagram:

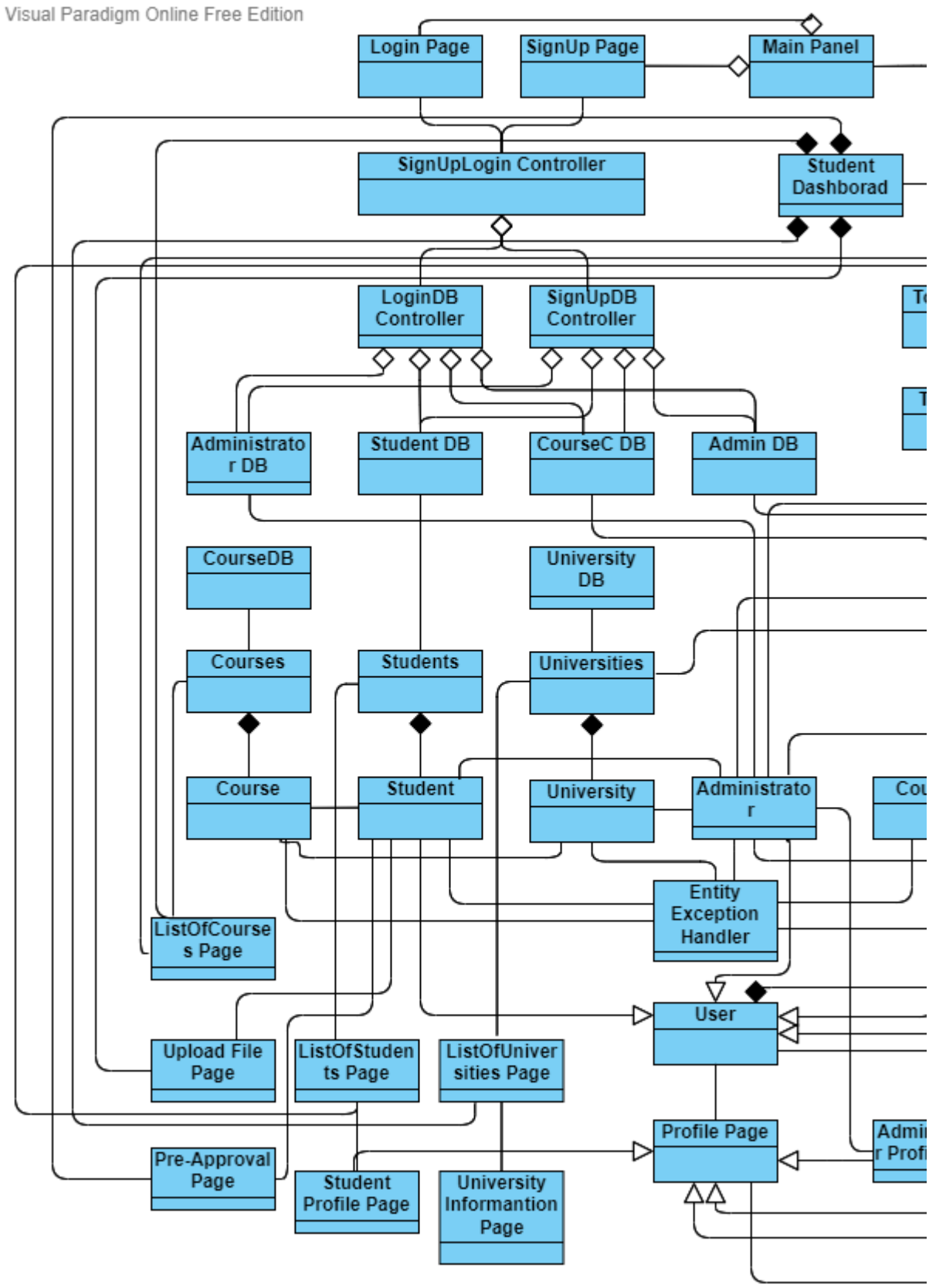

Figure 4 Final Object Design Left Side

Right Side of the Diagram:

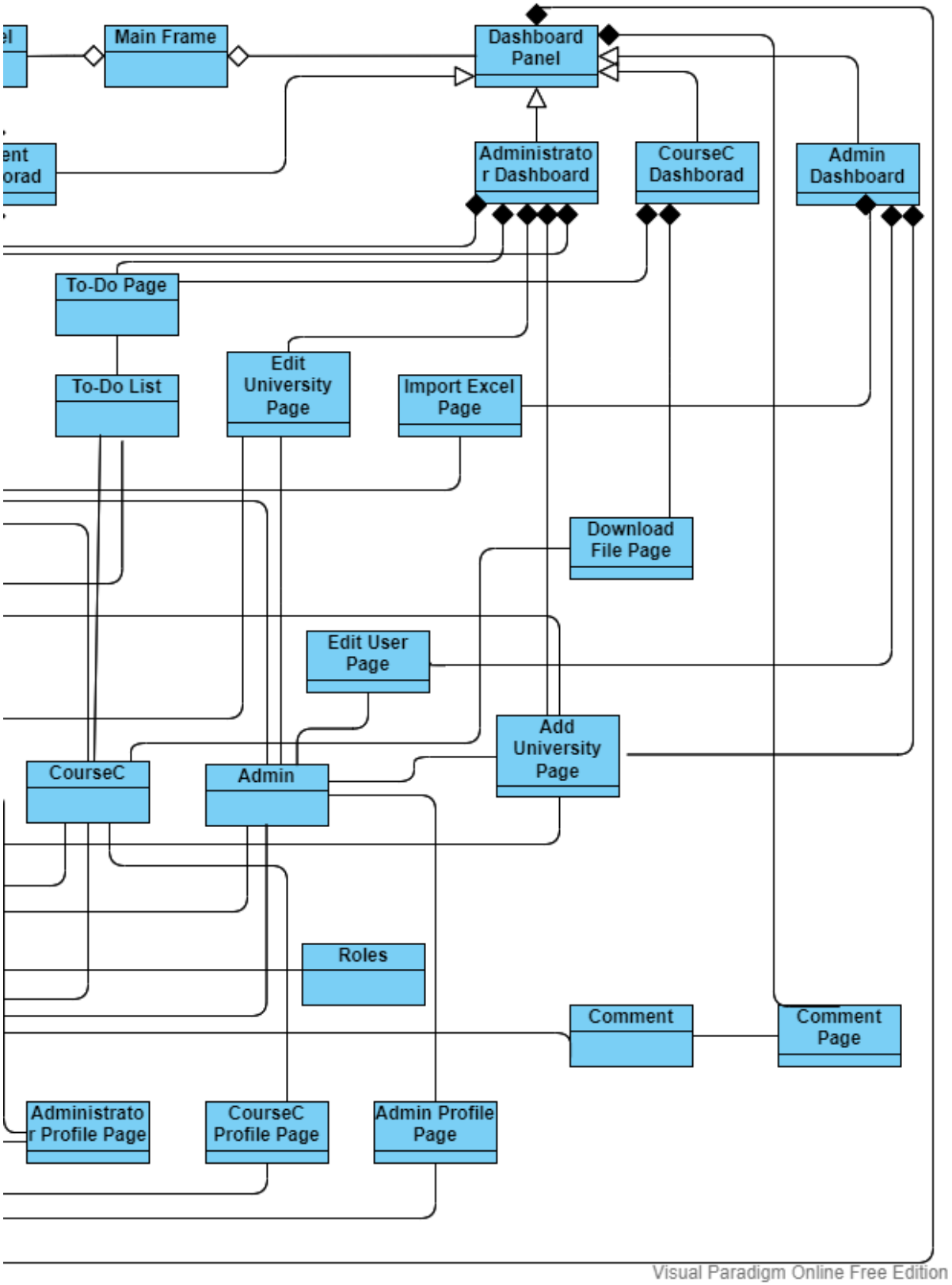

Figure 5 Final Object Design Right Side

3.3 Layers

3.3.1 User Interface Management Layer

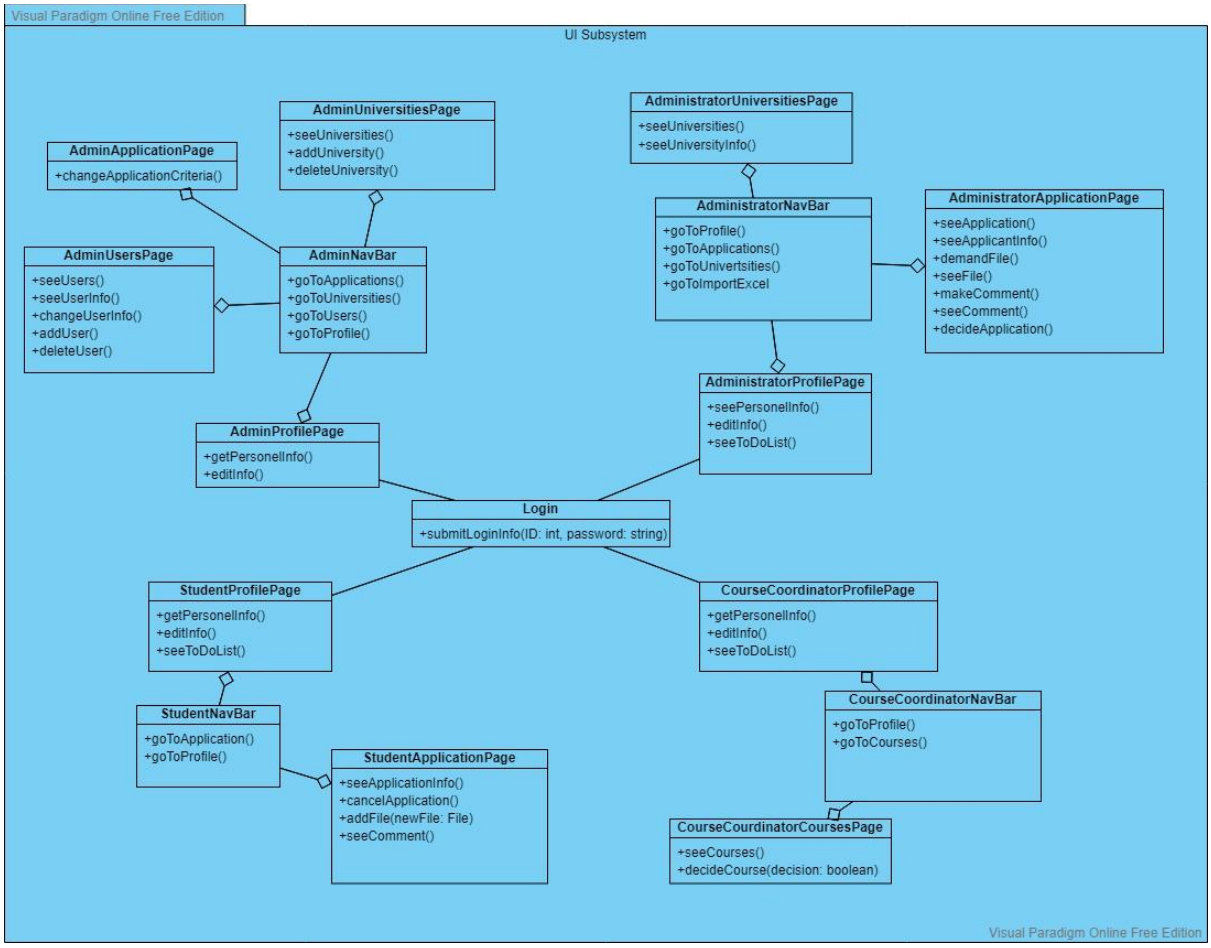

Figure 6 User Interface Management Layer

3.3.2. Web Server Subsystem

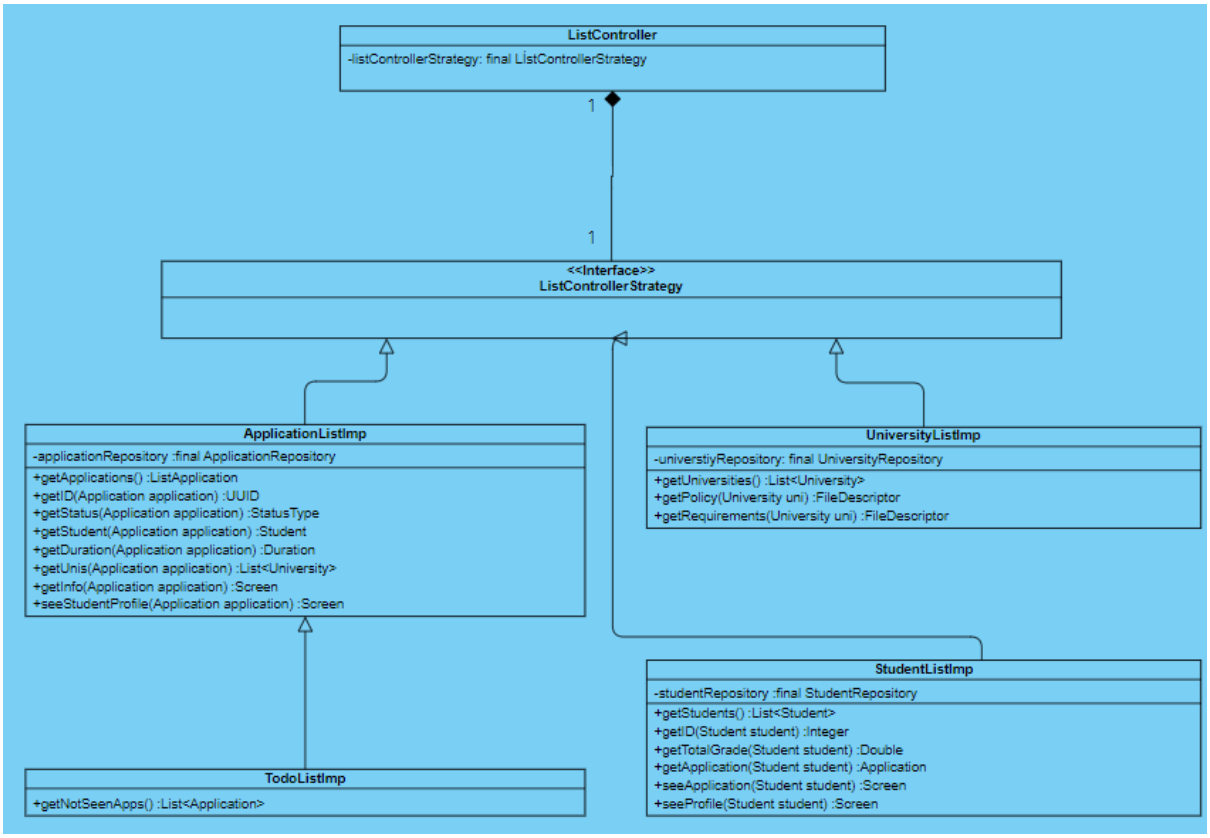

Figure 7 Web Server Subsystem

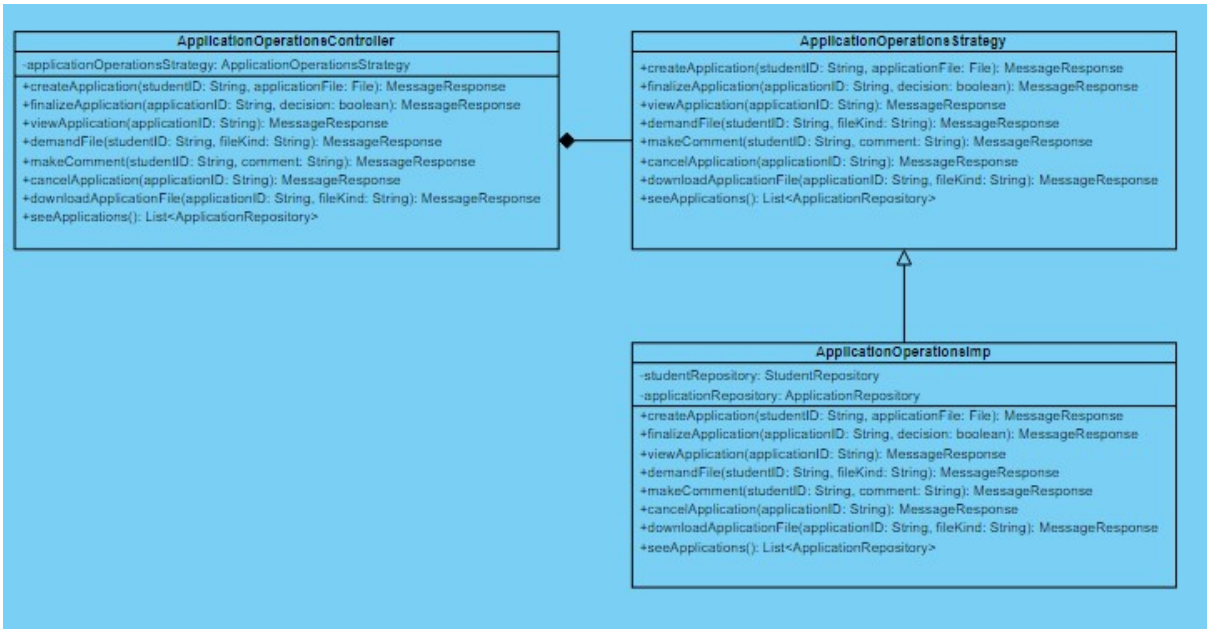

Figure 8 Web Server Subsystem

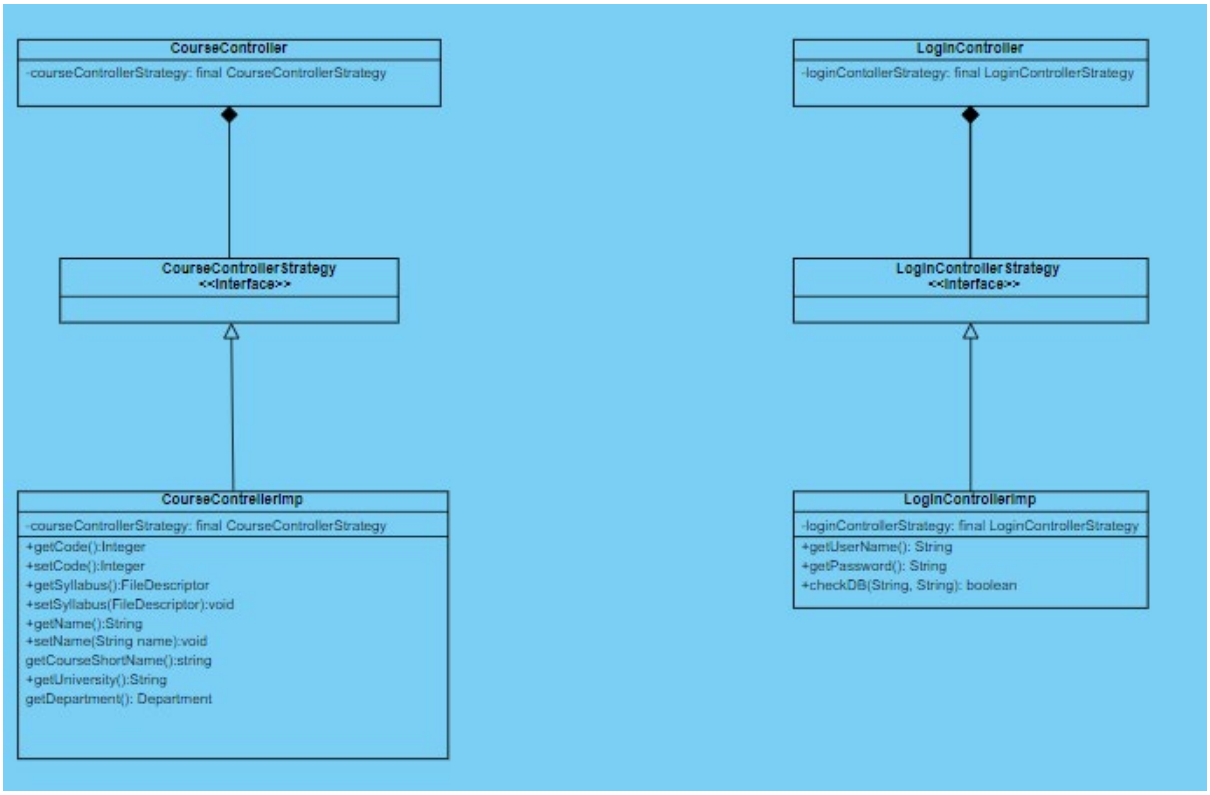

Figure 9 Web Server Subsystem

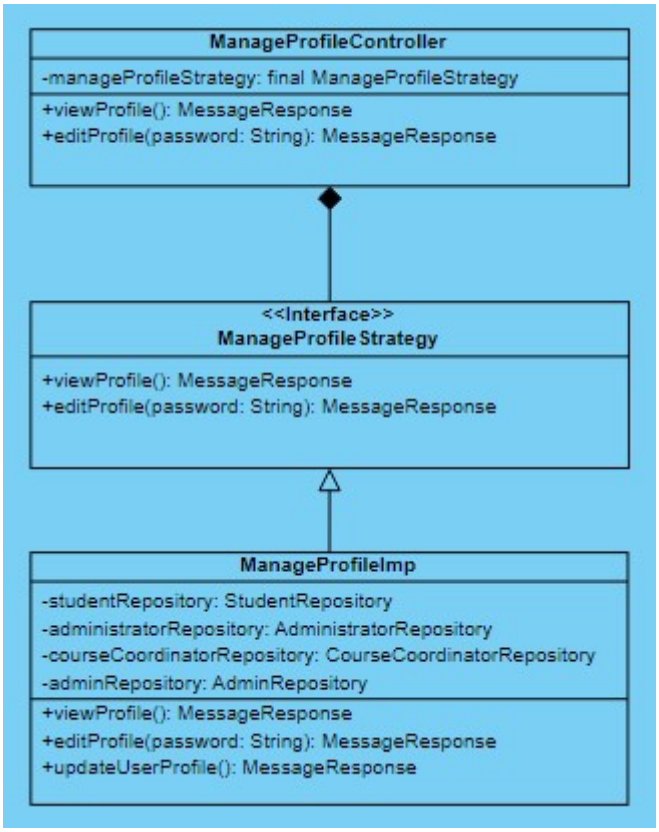

Figure 10 Web Server Subsystem

3.3.3 Data Management Layer

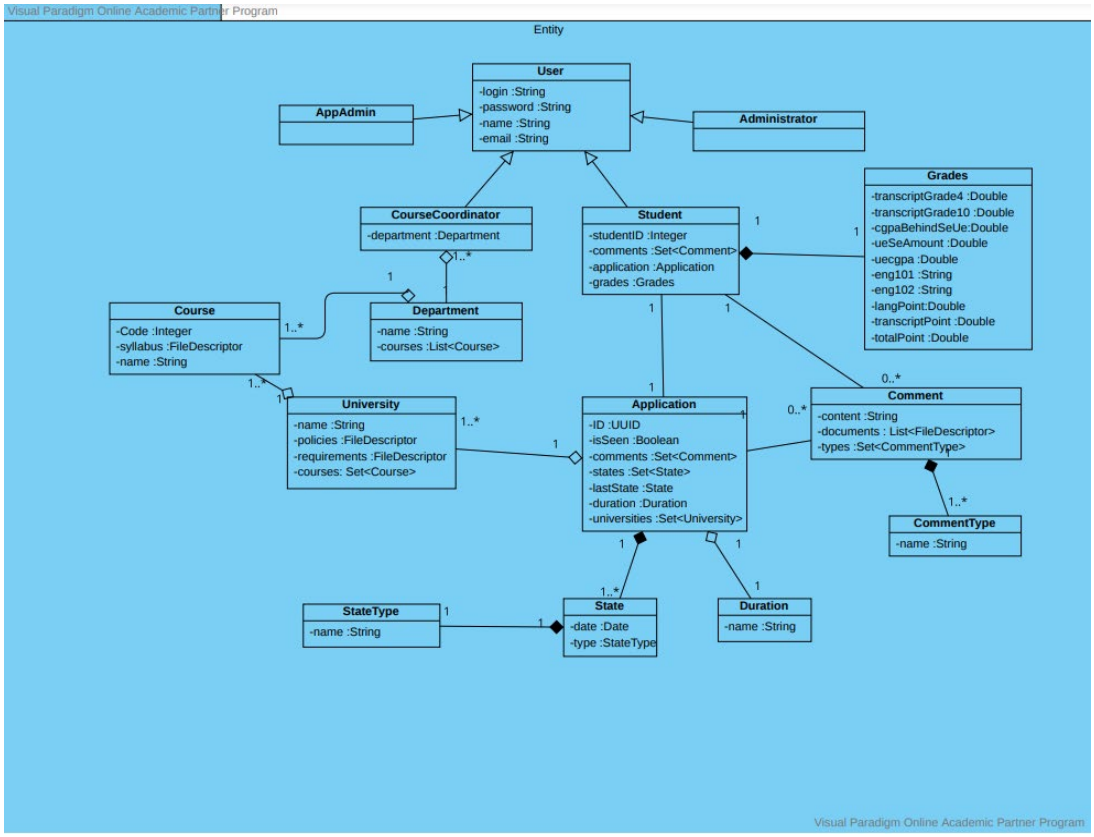

Figure 11 Entity Diagram

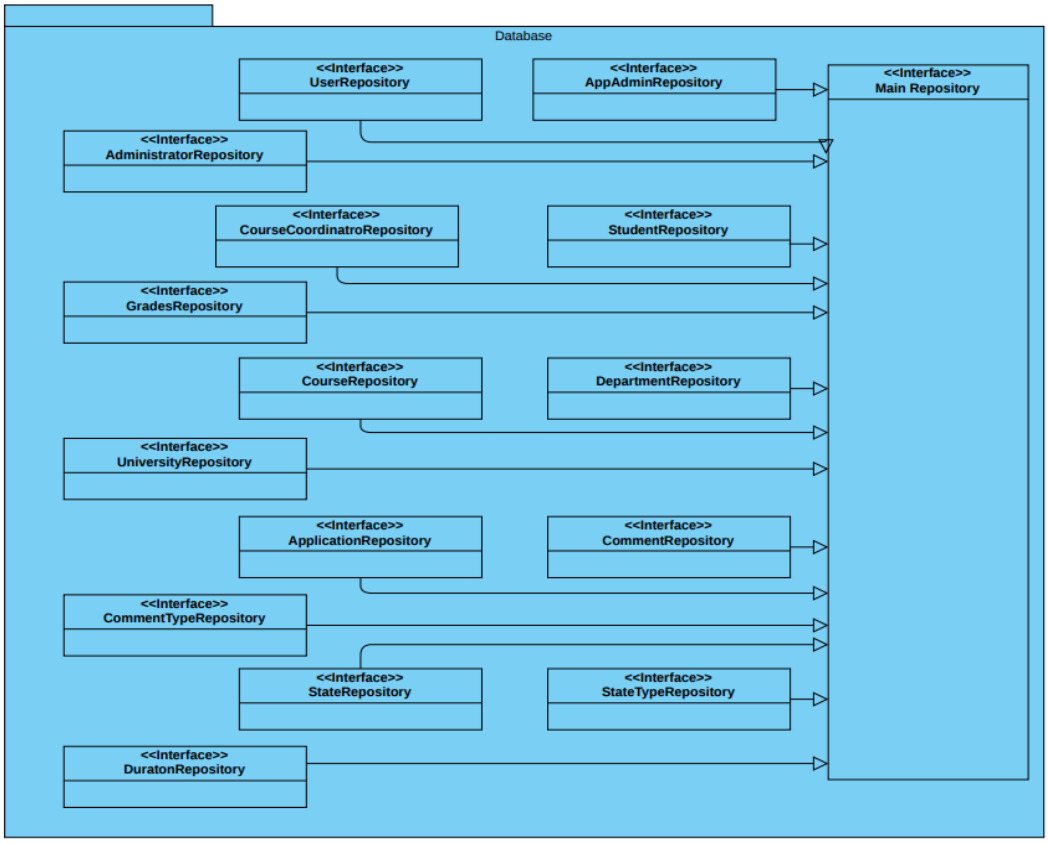

Figure 12 Database Diagram

3.4 Design Patterns

3.4.1 Facade Design Pattern

In our application we used facade design patterns especially for student application flows. In our design application has many relations with other classes such as status, duration, and comment and all of them have different behaviors for handling all of them within student class. We decided to create other classes to modify application properties and functions. For this purpose the application class used to handle all of the functions which students need to do. So in this site the program facade design pattern is used.

3.4.2 Builder pattern

In our application builder design pattern is used for creation of different types of users. For example, admin will create a user and according to information admin gives. Different types of users are created.

3.5 Packages

3.5.1 Packages Used by Developers

3.5.1.1 Entities

This package includes an Entity class containing all of the entities that are used in the application to handle their operations.

3.5.1.2 Login and Signup Controller

This package includes classes which handle the signup and login page actions of the application.

3.5.1.3 Profile Controller

This package includes classes which handle the profile page actions of the application.

3.5.1.4 Main Page and Dashboard Controller

This package includes classes which handle the main page and dashboard of the application.

3.5.1.5 List of Students Controller

This package includes classes which handle the operations of the Student objects and the list of them used in application.

3.5.1.6 File Upload Controller

This package includes classes which handle the files uploaded to the application.

3.5.1.7 To-Do List Controller

This package includes classes which handle the To-Do list of Student and Administrator objects of the application.

3.5.1.8 Approved Course List Controller

This package includes classes which handle changes in the approved courses list of the application.

3.5.1.9 List of Universities Controller

This package includes classes which handle the operations of the University objects and the list of them used in application.

3.5.1.10 Exception Handler

This package includes classes which handle the exceptions of the application.

3.5.1.11 Database

This package includes classes which handle the operations in order to fetch from and write into the database of the application.

3.5.2 External Library Packages

3.5.2.1 com.haulmont.cuba.gui.notifications

This package includes classes which send notifications.

com.haulmont.cuba.gui.screen

This package includes classes which display the screens.

com.haulmont.cuba.security.auth

This package includes classes which handle the security issues of application

3.6 Class Interfaces

3.6.1 User Interface Layer Class Interfaces

3.6.1.1 Login

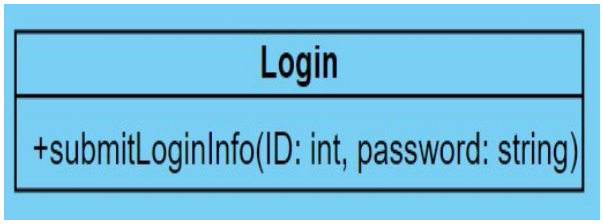

Figure 13 Login Class

Login page that users can log in to the system.

Operations:

public submitLoginInfo(int ID, String password): On click, takes required information and sends it to the database for comparison. According to the information about the role, the program opens the next page.

3.6.1.2 AdminNavBar

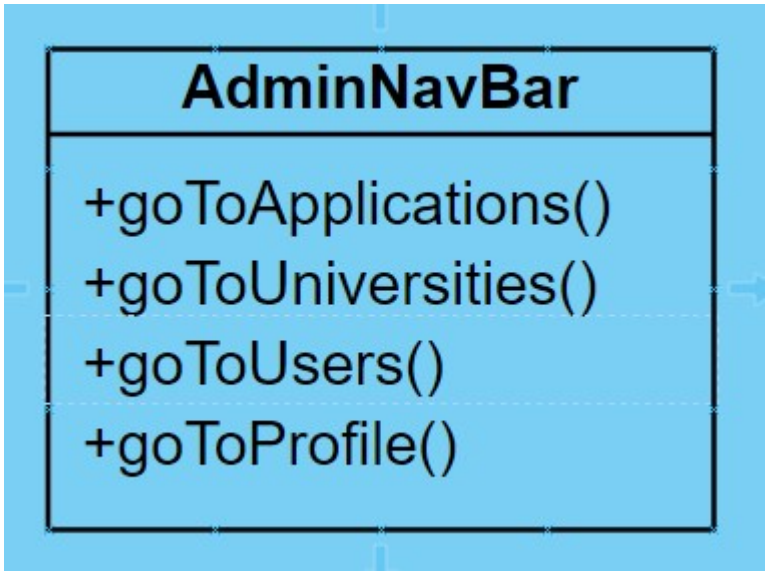

Figure 14 AdminNavBar class

AdminNavBar is a navigation bar for all admin pages.

Operations:

public goToApplications(): On click, opens the application management page.

public goToUniversities(): On click, opens the universities page.

public goToUsers(): On click, opens the users page.

public goToProfile(): On click, opens the admin profile page.

3.6.1.3 AdminProfilePage

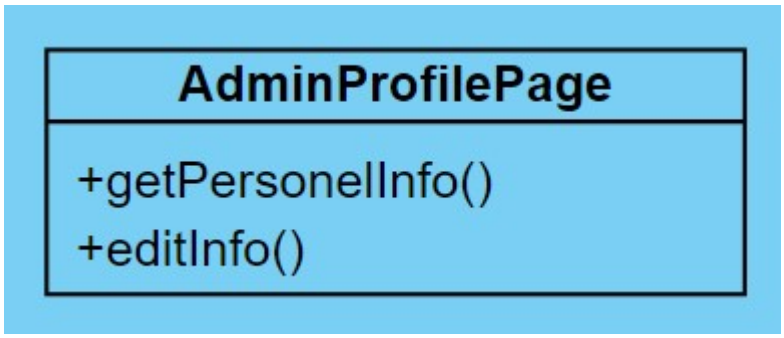

Figure 15 AdminProfilePage class

AdminProfilePage gives information about admins and admins can change information about them.

Operations:

public getPersonelInfo(): Shows information of admin.

public editInfo(): On click, admins can change their information.

3.6.1.4 AdminUsersPage

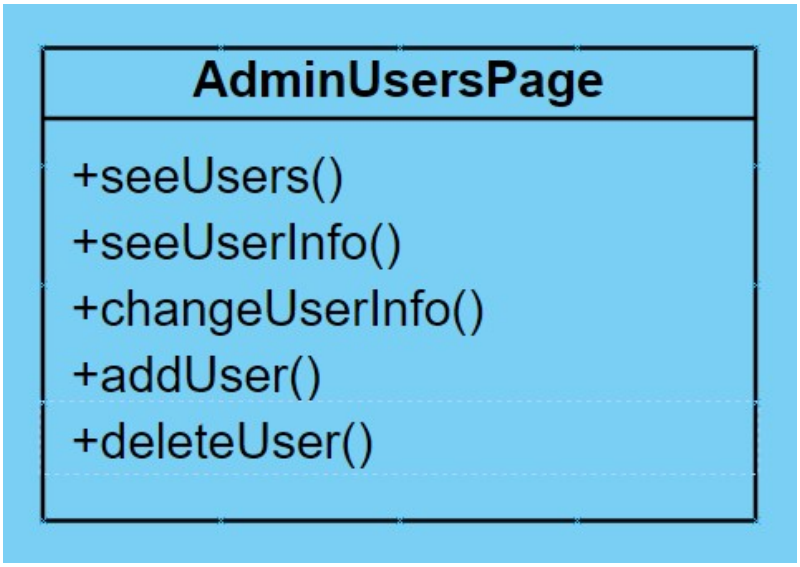

Figure 16 AdminUsersPage class

AdminUsersPage gives the authority about users. They can see, add or delete a user.

Operations:

public seeUsers(): On click, admin can see a brief information of users.

public seeUserInfo(): On click, admins can see the information of a selected user.

public changeUserInfo(): On click with necessary information, admins can change the information of the selected user.

public addUser(): On click with necessary information, admins can add a user to a system.

public deleteUser(): On click with necessary information, admins can delete a user to a system.

3.6.1.5 AdminUniversitiesPage

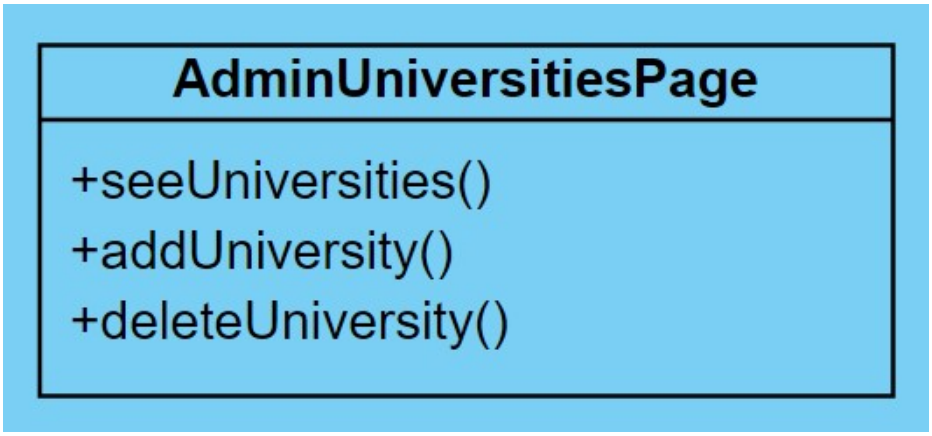

Figure 17 AdminUniversitiesPage Class

AdminUniversitiesPage gives the authority about universities. They can see, add or delete a university.

Operations:

public seeUniversities(): On click, admin can see the information of a user.

public addUniversity(): On click with necessary information, admins can add a university to the system.

public deleteUniversity(): On click with necessary information, admins can delete a university from the system.

3.6.1.6 AdminApplicationPage

Figure 18 AdminApplicationPage class

AdminApplicationPage helps admins to change the application criteria.

Operations:

public changeApplicationCriteria: Admins can change the application criteria from this page.

3.6.1.7 StudentProfilePage

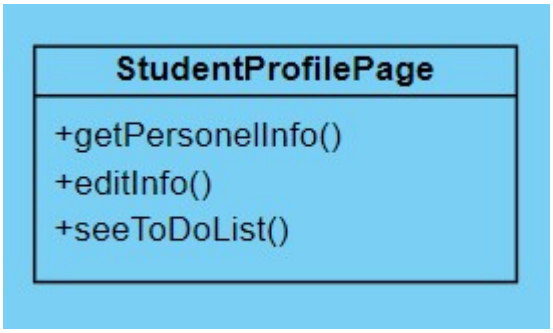

Figure 19 StudentProfilePage Class

StudentProfilePage gives information about students and students can change information about them.

Operations:

public getPersonelInfo(): Shows information of the student.

public editInfo(): On click, the student can change their information.

public seeToDoList(): Students can see their To-Do list.

3.6.1.8 StudentNavBar

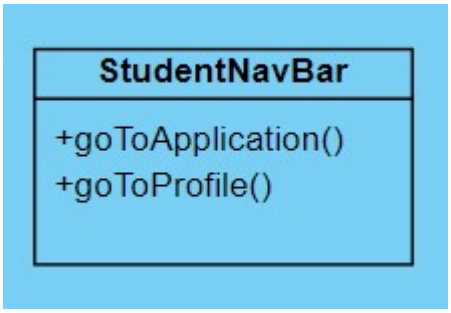

Figure 20 StudentNavBar class

StudentNavBar is a navigation bar for all admin pages.

Operations:

public goToApplication(): On click, opens the application page for the student.

public goToProfile(): On click, opens the admin profile page.

3.6.1.9 StudentApplicationPage

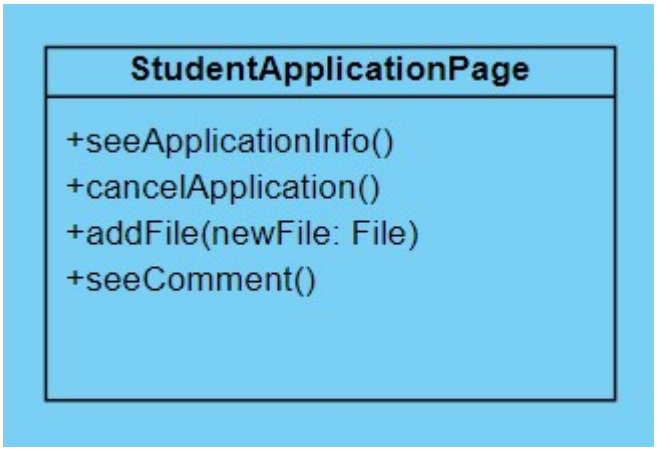

Figure 21 StudentApplicationPage Class

StudentApplicaitonPage is shows the information about the student's application.

Operations:

public seeApplicationInfo(): Shows the application information.

public cancelApplication(): Cancel the application.

public addFile(): Students can add the desired file from Erasmus administrators.

public seeComment(): Shows the comment from Erasmus administrators.

3.6.1.10 CourseCoordinatorProfilePage

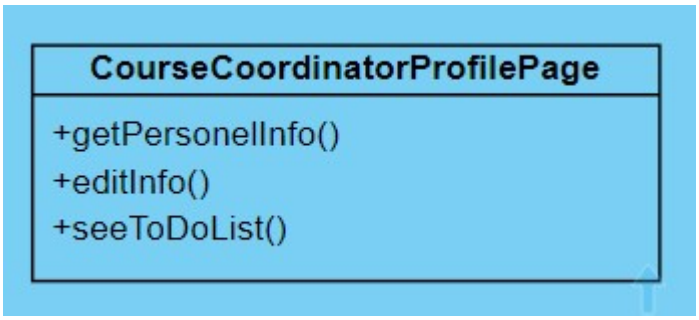

Figure 22 CourseCoordinatorProfilePage class

CourseCoordinatorProfilePage gives information about course coordinators and course coordinators can change information about them.

Operations:

public getPersonelInfo(): Shows information of course coordinators.

public editInfo(): On click, course coordinators can change their information.

public seeToDoList(): Course coordinators can see their To-Do List.

3.6.1.11 CourseCoordinatorNavBar

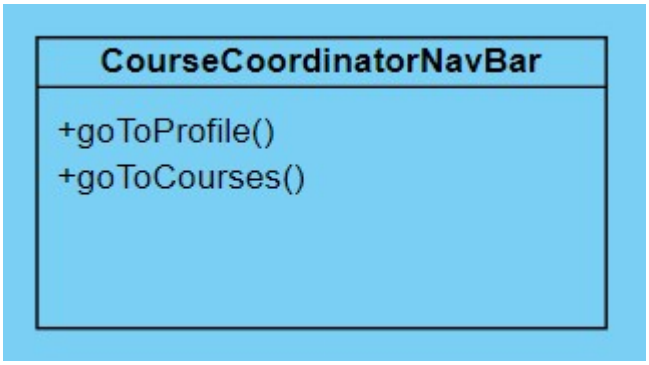

Figure 23 CourseCoordinatorNavBar Class

CourseCoordinatorNavBar is a navigation bar for all course coordinator pages.

Operations:

public goToProfile(): On click, opens the course coordinators profile page.

public goToCourses(): On click, opens the courses page for the course coordinators.

3.6.1.12 CourseCoordinatorCoursesPage

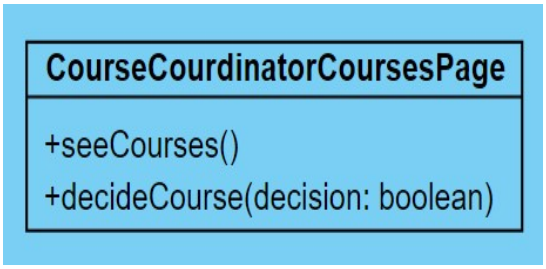

Figure 24 CourseCoordinatorCoursesPage class

CourseCoordinatorCoursesPage shows the information about the courses and gives course coordinators authority to decide on waiting courses.

Operations:

public seeCourses(): Shows the application information.

public decideCourse(boolean decision): According to selection, the course can be included in the syllabus for the department.

3.6.1.13 AdministratorProfilePage

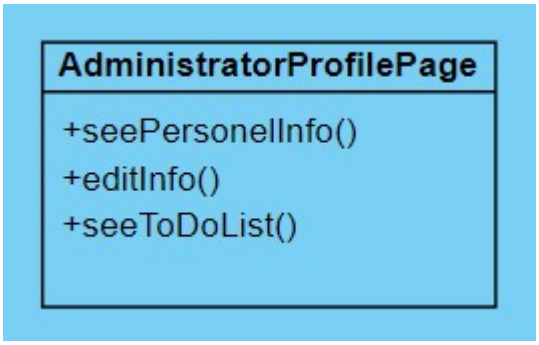

Figure 25 AdministratorProfilePage Class

AdministratorProfilePage gives information about Erasmus administrators and Erasmus administrators can change information about them.

Operations:

public getPersonelInfo(): Shows information of Erasmus administrators .

public editInfo(): On click, Erasmus administrators can change their information.

public seeToDoList(): Erasmus administrators can see their To-Do List.

3.6.1.14 AdministratorNavBar

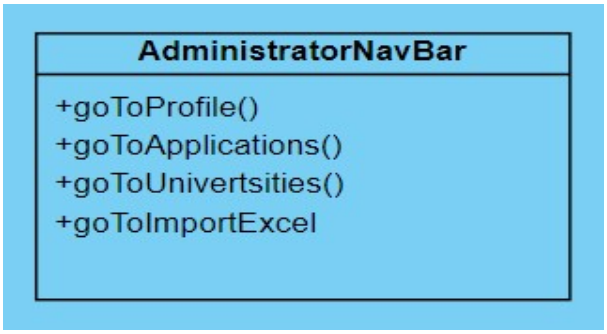

Figure 26 AdministratorNavBar class

AdministratorNavBar is a navigation bar for all Erasmus administrator pages.

Operations:

public goToProfile(): On click, opens the admin profile page.

public goToApplications(): On click, opens the application management page.

public goToUniversities(): On click, opens the universities page.

3.6.1.15 AdministratorApplicationPage

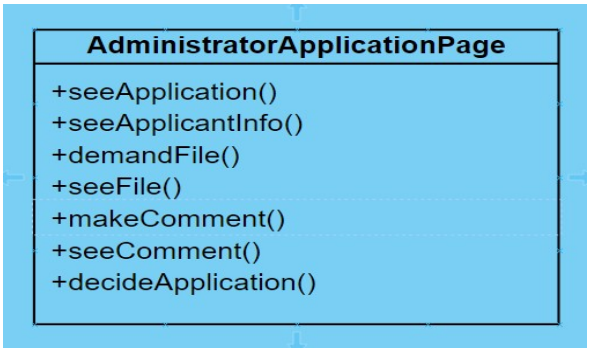

Figure 27 AdministratorApplicationPage class

AdministratorApplicationPage can give the authority to Erasmus administrators to do operations on applications.

Operations:

public seeApplications(): Shows the brief information about applications.

public seeApplicationInfo(): On click, shows the information of the selected application.

public demandFile(): Demands file from selected student for selected feature.

public seeFile(): On click, shows the selected file from the selected application.

public makeComment(): Make comment to the selected application.

public seeComment(): See comment which is made to the selected application.

public decideApplication(): Decide application whether it is appropriate to approve.

3.6.1.16 AdministratorUniversitiesPage

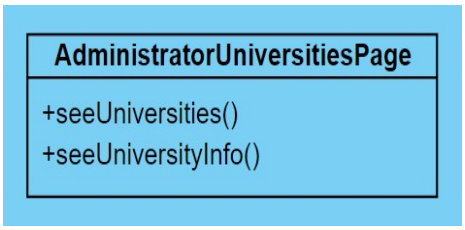

Figure 28 AdministratorUniversitiesPage class

AdministratorUniversitiesPage shows contracted universities with Bilkent University.

Operations:

public seeUniversities(): Shows the brief information about universities.

public seeUniversitityInfo(): Shows the information about the selected university.

3.6.2 Web Server Layer Class Interfaces

3.6.2.1 Application User LoginService

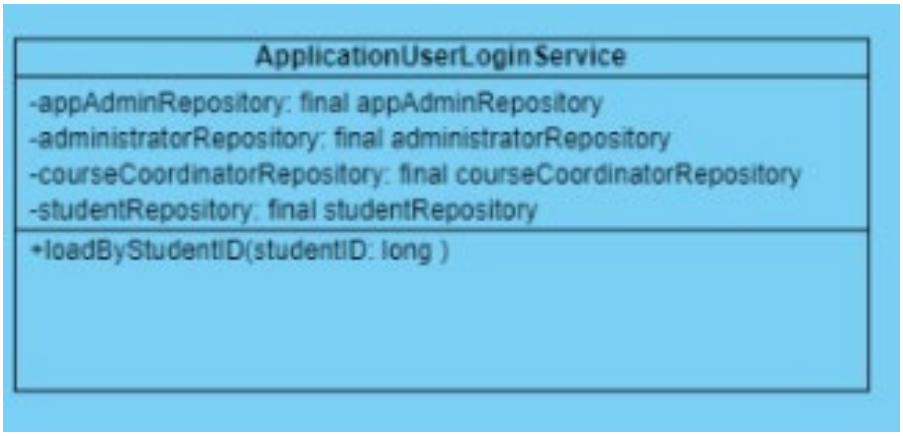

Figure 29 ApplicationUserLoginService class

3.6.2.2 ApplicationOperationsController

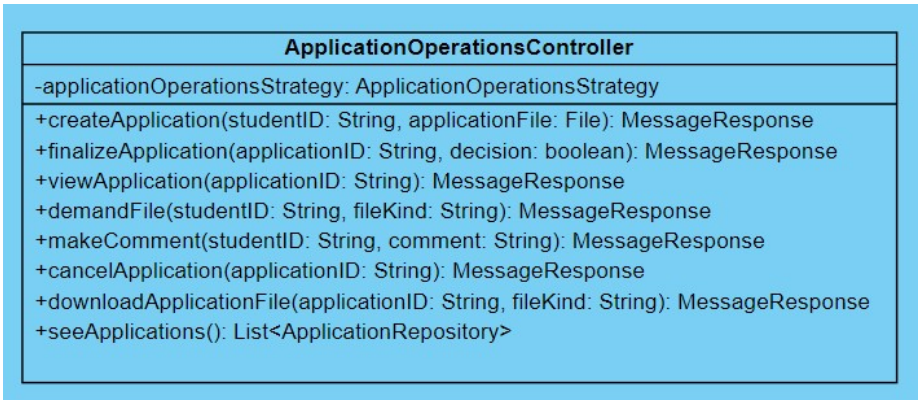

Figure 30 ApplicationOperationsController class

This class controls the application operations and sends these activities to the service.

Operations:

public MessageResponse createApplication(String: studentID, File applicationFile): This method creates an application.

public MessageResponse finalizeApplication(String: applicationtID, boolean decision): This method helps Erasmus administrators to finalize an application.

public MessageResponse viewApplication(String: applicationID): This method shows application documents.

public MessageResponse demandFile(String: studentID, String fileKind): This method helps Erasmus administrators to demand a file from a desired student.

public MessageResponse makeComment(String: studentID, String comment): This method helps Erasmus administrators to make a comment about an application.

public MessageResponse cancelApplication(String: applicationID): This method helps students to cancel their applications.

public MessageResponse downloadApplication(String: applicationID, String fileKind): This method helps students or Erasmus administrators to download students' application documents.

3.6.2.3 ApplicationOperationsImp

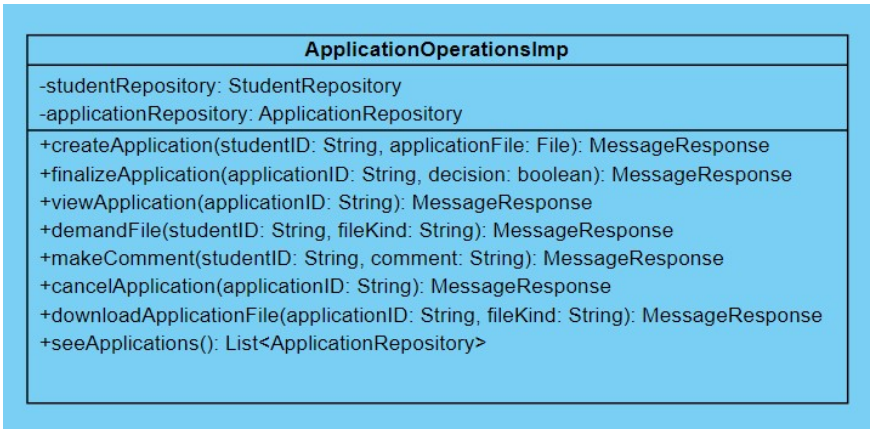

Figure 31 ApplicationOperationImp class

This class takes functions from the controller and implements functions with data that came from the database.

Operations:

public MessageResponse createApplication(String: studentID, File applicationFile): This method creates an application.

public MessageResponse finalizeApplication(String: applicationtID, boolean decision): This method helps Erasmus administrators to finalize an application.

public MessageResponse viewApplication(String: applicationID): This method shows application documents.

public MessageResponse demandFile(String: studentID, String fileKind): This method helps Erasmus administrators to demand a file from a desired student.

public MessageResponse makeComment(String: studentID, String comment): This method helps Erasmus administrators to make a comment about an application.

public MessageResponse cancelApplication(String: applicationID): This method helps students to cancel their applications.

public MessageResponse downloadApplication(String: applicationID, String fileKind): This method helps students or Erasmus administrators to download students' application documents.

3.6.2.4 ManageProfileController

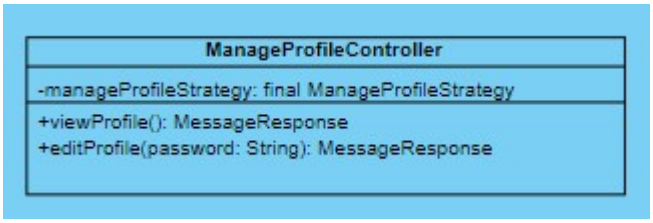

Figure 32 ManageProfileController class

This class controls the application operations and sends these activities to the service.

Operations:

public MessageResponse viewProfile(): This method shows the profile.

public MessageResponse editProfile(String: password): This method helps users to edit their profiles.

3.6.2.5 ManageProfileImp

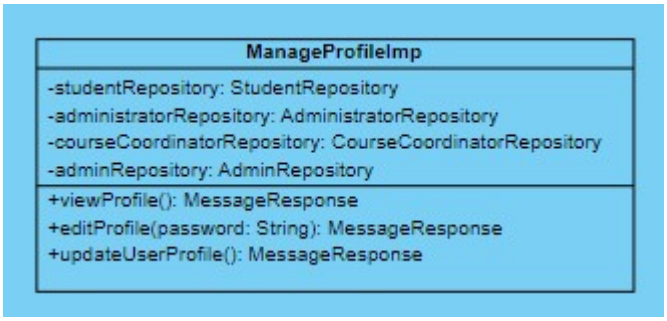

Figure 33 ManageProfileImp class

This class takes functions from the controller and implements functions with data that came from the database.

Operations:

public MessageResponse viewProfile(): This method shows the profile.

public MessageResponse editProfile(String: password): This method helps users to edit their profiles.

public MessageResponse updateUserProfile(): This method updates a user profile..

3.6.3 Data Management Layer Class Interfaces

Important Note: All entity classes whose information is provided in this section have an empty constructor to initialize, and a setter and a getter function for each attribute of these entity classes.

3.6.3.1 User

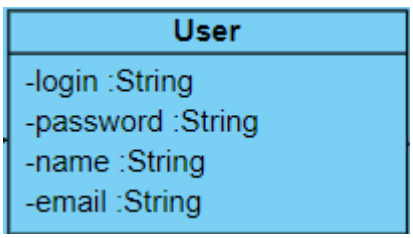

Figure 34 User class

This class holds fundamental information about the program's users.

Attributes:

- **private String login :** Username of user's that they use for log in.
- **private String password:** Password of user for logging in.
- **private String name:** Name of the user.
- **private String email:** Email address of the user.

Constructor:

Constructor which has all attributes for initializing class.

3.6.3.2 AppAdmin

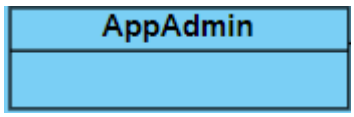

Figure 35 AppAdmin class

This is the class of Application Admin.

Attributes : Empty.

Constructor: Empty.

3.6.3.3 Erasmus Administrator

Figure 36 Administrator class

This is the class of Erasmus administrators.

Attributes : Empty.

Constructor: Empty.

3.6.3.4 Course Coordinator

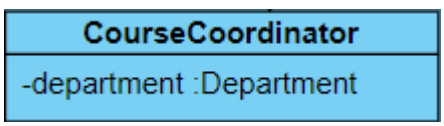

Figure 37 CourseCoordinator class

This is the class of Course Coordinators.

Attributes :

- **private Department department :** Department of the instructor.

Constructor:

Constructor which has all attributes for initializing class.

3.6.3.5 Student

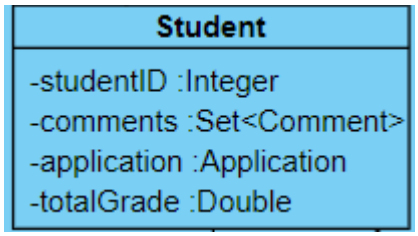

Figure 38 Student class

This is the class of Students.

Attributes :

- **private Integer student ID:** Bilkent ID of student.
- **private Set<Comment> comment:** Comments that are done by students.
- **private Application application:** Erasmus application of the student.
- **private Double totalGrade:** Total grade of the student

Constructor:

Constructor which has all attributes for initializing class.

3.6.3.6 Comment

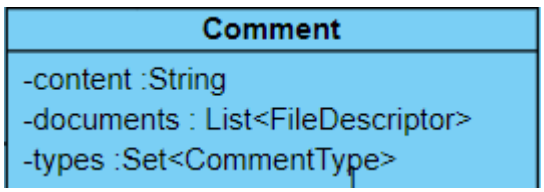

Figure 39 Comment class

This is the class of Comments.

Attributes:

- **private String content:** Content of the comment.

- **private List<FileDescriptor> documents:** Documents that are attached to comment:
- **private Set<Comment Type> types:** Type of comment such as Course Request, pointing out false information.

Constructor:

Constructor which has all attributes for initializing class.

3.6.3.7 Comment Type

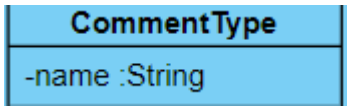

Figure 40 CommentType class

This is the class of comment types.

Attributes:

- **private String name:** Type of the comment.

3.6.3.8 Application

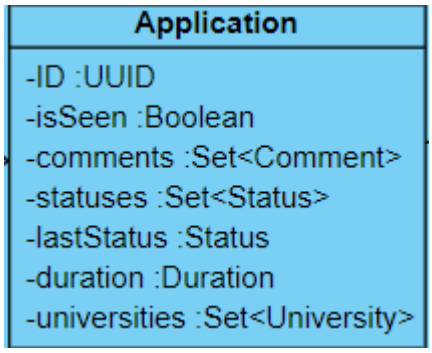

Figure 41 Application class

This is the class of Applications.

Attributes:

- **private UUID ID:** unique ID of the application.
- **private boolean isSeen:** information that checks if there is a new comment or update on application status.
- **private Set<Comment> comments:** Comments that are done to applications.

- **private Status lastStatus:** Status of the application such as being evaluated, approved, rejected, in reserve.
- **private Set<Status> statuse**s: All status the application has from beginning to chasing past records.
- **private Duration duration :** Duration of application (Fall and Spring)
- **private Set< University> universities:** Universities that are applied.

Constructor:

Constructor which has all attributes for initializing class.

3.6.3.9 Duration

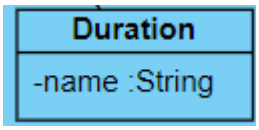

Figure 42 Duration class

This is the class of Duration..

Attributes:

- **private String name:** Name of the Duration (Fall, Spring)

Constructor:

Constructor which has all attributes for initializing class.

3.6.3.10 Status

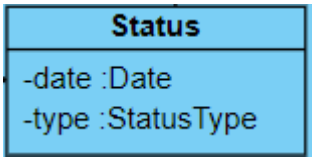

Figure 43 Status class

This is the class of Status.

Attributes:

- **private String date:** Date of the Status.

- **private StatusType type:** Type of the Status.

Constructor:

Constructor which has all attributes for initializing class.

3.6.3.11 StatusType

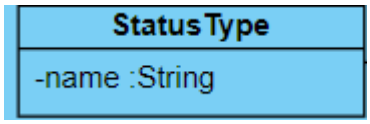

Figure 44 StatusType class

This is the class of status types.

Attributes:

- **private String name:** Name of the Status type.

Constructor:

Constructor which has all attributes for initializing class.

3.6.3.12 University

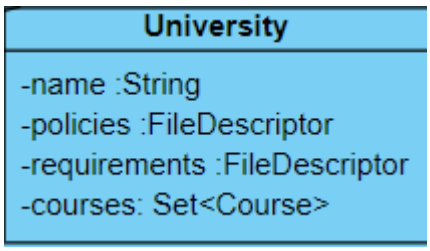

Figure 45 University class

This is the class of Universities.

Attributes:

- **private String name:** Name of the Status type.

Constructor:

Constructor which has all attributes for initializing class.

3.6.3.13 Course

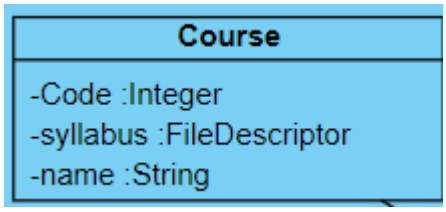

Figure 46 Course class

This is the class of Course.

Attributes:

- **private Integer code:** Course code.
- **private FileDescriptor syllabus:** Syllabuses of the course.
- **private String name:** Name of Course

Constructor:

Constructor which has all attributes for initializing class.

3.6.3.14 Department

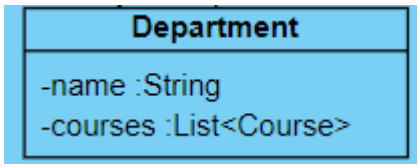

Figure 47 Department class

This is the class of the department The courses and course coordinators belong to a department.

Attributes:

- **private List<Course> courses:** name of the courses that belong to this department.
- **private String name:** Name of department

Constructor:

Constructor which has all attributes for initializing class.

4. Improvement Summary

4.1 General

- Figure captions are added.
- Web server subsystems are created and modified.
- All class interfaces and their explanations are added with their methods and attributes.

4.2 Low-level Design

Functions and attributes of the entities added

4.3. Glossary and References

The references have been formatted using the IEEE Citation Style

[1] *Including the library - Cuba platform. developer's Manual*. [Online]. Available: https://doc.cuba-platform.com/manual-latest/jespa_lib.html. [Accessed: 11-Dec-2022].

[2] Haulmont, “Product customizations with Application Components,” *High productivity application development platform*. [Online]. Available: https://www.jmix.io/cuba-platform/guides/product-customizations-with-app-components. [Accessed: 11-Dec-2022].

[3] *UML class diagram tutorial*. [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/. [Accessed: 11-Dec-2022].

.