Bilkent University

Department of Computer Engineering

Senior Design Project

ErasmusAga

Design Report

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: İdil Hanhan

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.

1.2 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.

1.2.1 User Interface

ErasmusAga application has a very simple user interface because we will use the CUBA platform. It provides us with the most common user interface platform because of this simplicity users can handle all of their work easily.

1.2.2 Simplicity

In the ErasmusAga application all of the works are simple communication and do not have any specific requirements such as direct message. Because ErasmusAga has a comment section, users do not need to specify which topic is problematic because they will comment on that page and administrator or course coordinator will be able to see directly which problem is coming from.

1.2.3 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.

1.2.4 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.

1.2.5 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, Application Layer, and the Data Layer.

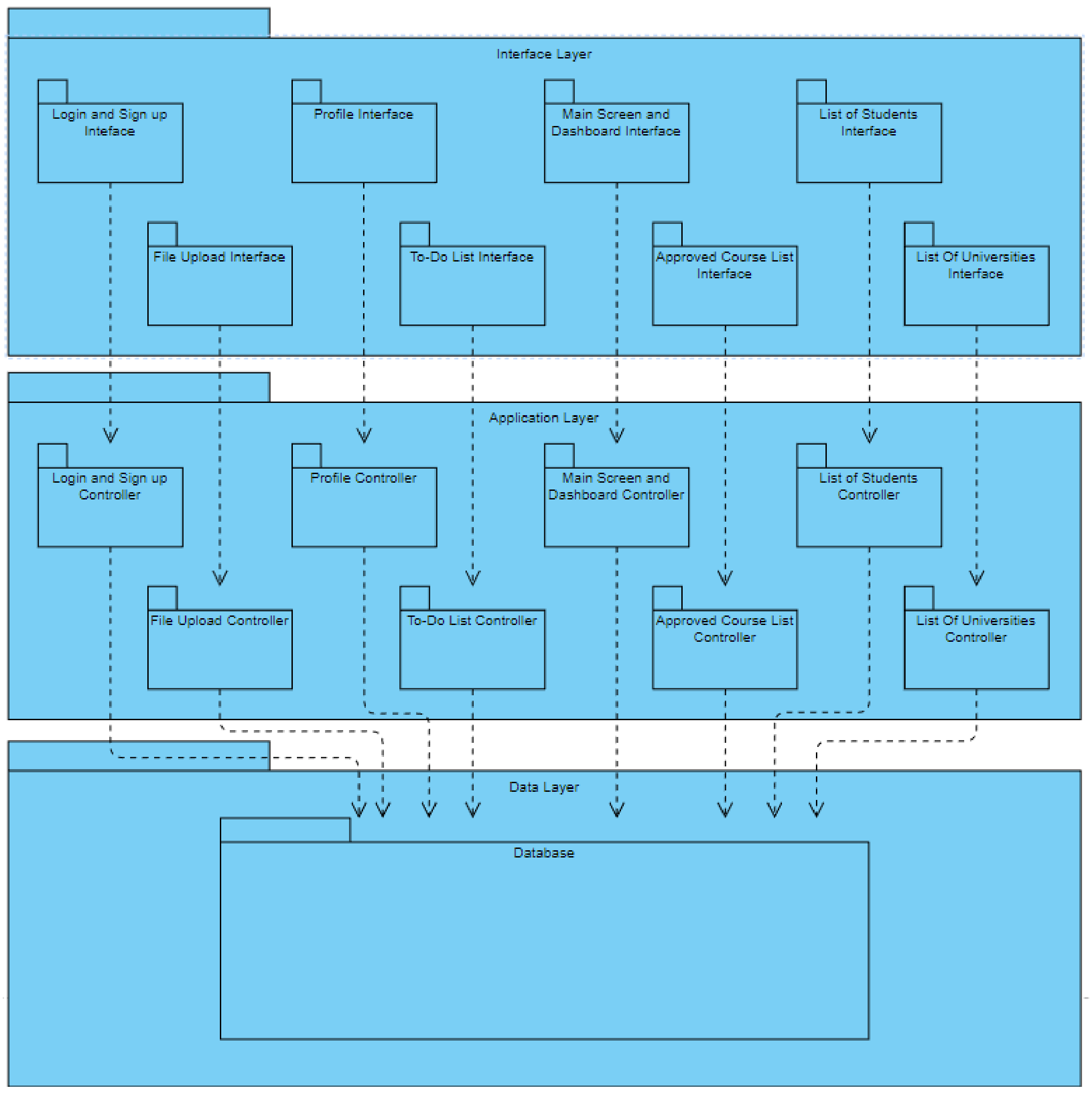
The Interface Layer is responsible for displaying information to the user and allowing the user to interact with the system. This layer serves as the boundary object that contains the interface classes which make up the web pages of the application. Pages are bound to their respective controllers in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

The Application 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 Application 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. The Application Layer collaborates with this layer to perform its functionalities. Data Layer either sends data to the Application 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 Application Layer. This design choice is to make the system more stable, modifiable and extendable. The figure below is a visual representation of the proposed system decomposition.


Interface Layer
Login and Sign up Inteface
Profile Interface
Main Screen and Dashboard Interface
List of Students Interface
File Upload Interface
To-Do List Interface
Approved Course List Interface
List Of Universities Interface
Application Layer
Login and Sign up Controller
Profile Controller
Main Screen and Dashboard Controller
List of Students Controller
File Upload Controller
To-Do List Controller
Approved Course List Controller
List Of Universities Controller
Data Layer
Database

2.2. Hardware/Software Mapping

This project will be implemented with the programming language Java. It will work on all operating systems such as Mac, Windows, Linux but requires a Java Runtime Environment (JRE). This is a web based project, therefore, web browsers such as Google Chrome, Safari, Firefox,etc.. are required for execution on any machine.

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, 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 students, universities, etc. Users will be able to edit said tables by invoking the functionalities in the Application 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 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.

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 created by OOP principles maintainability will be high since any other coder will be understand the code easily so if new coder understand the old code fixing and software development will also easy.

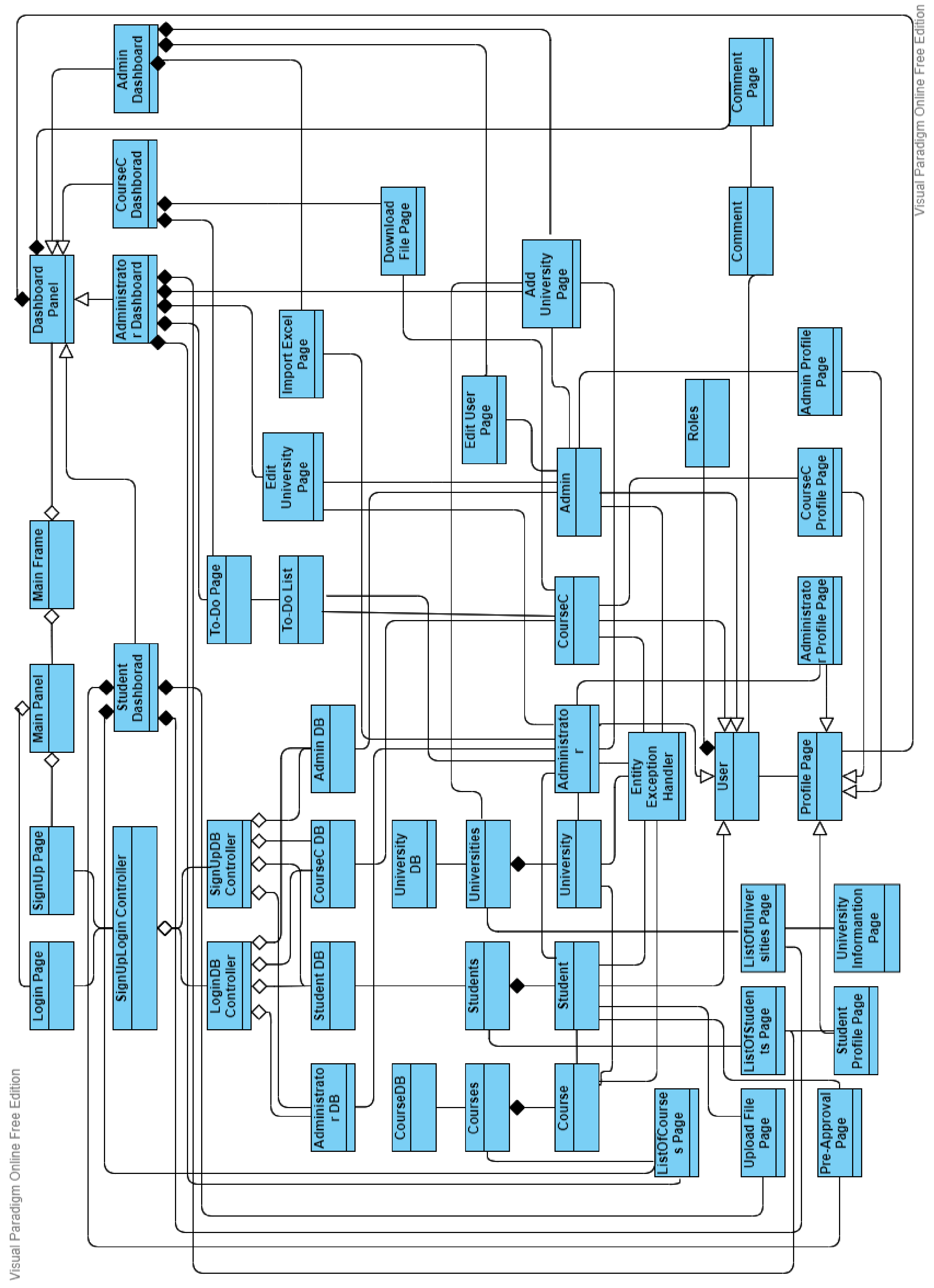
Security versus Usability

ErasmusAga allows users only to upload pdf files (only 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, 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.

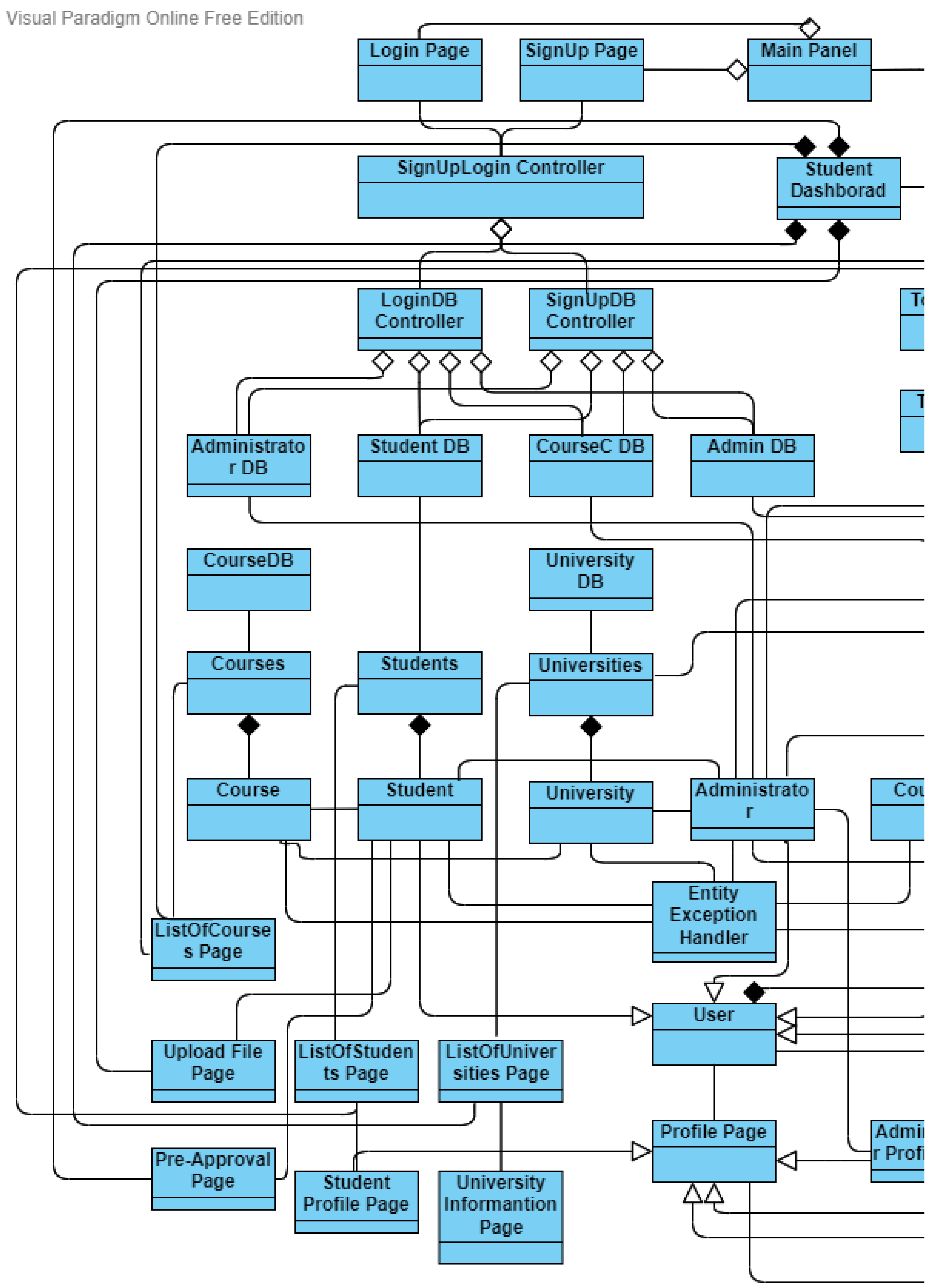
Functionality versus Usability

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.

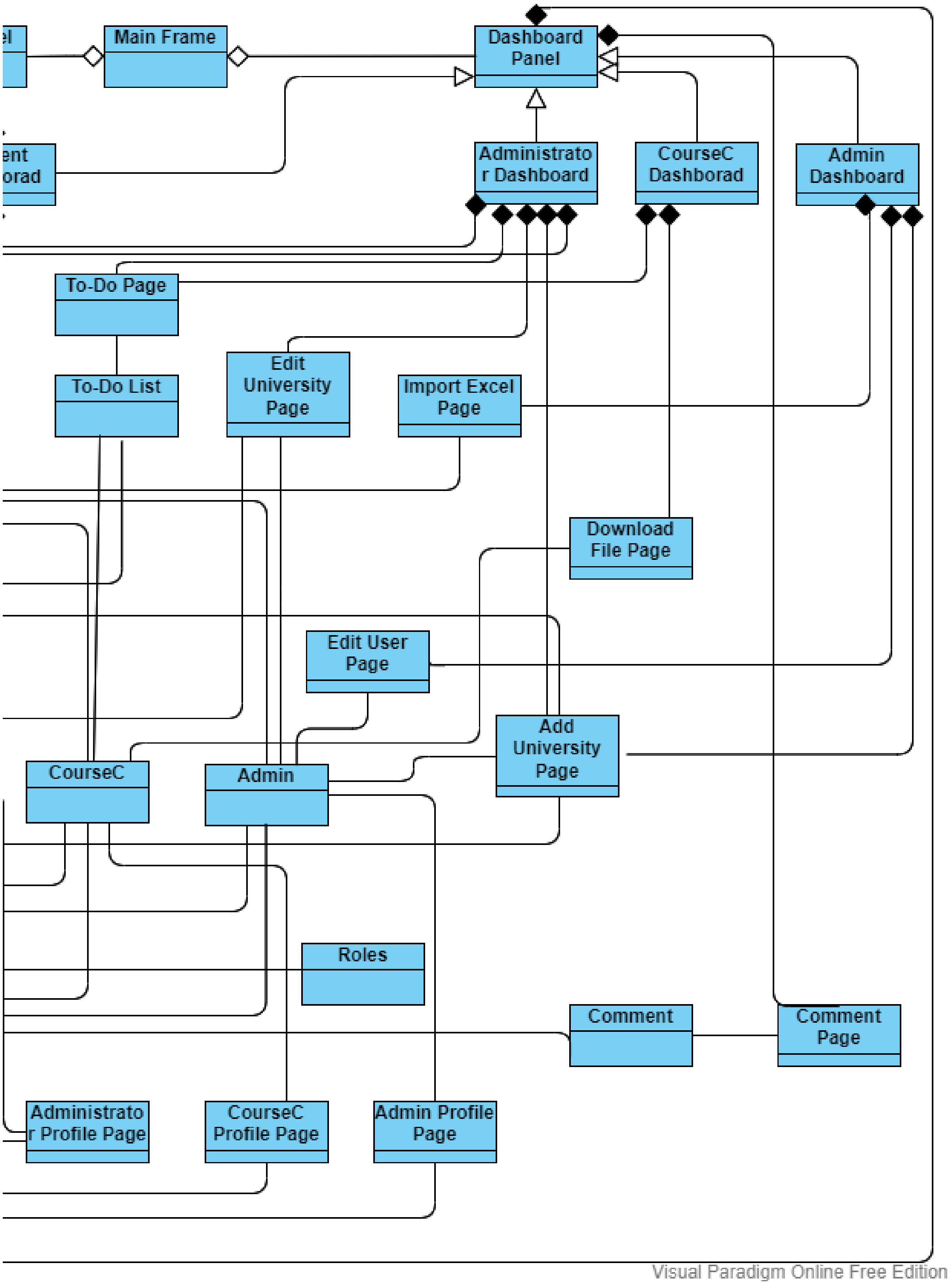
3.2 Final Object Design



Left Side Of The Diagram:



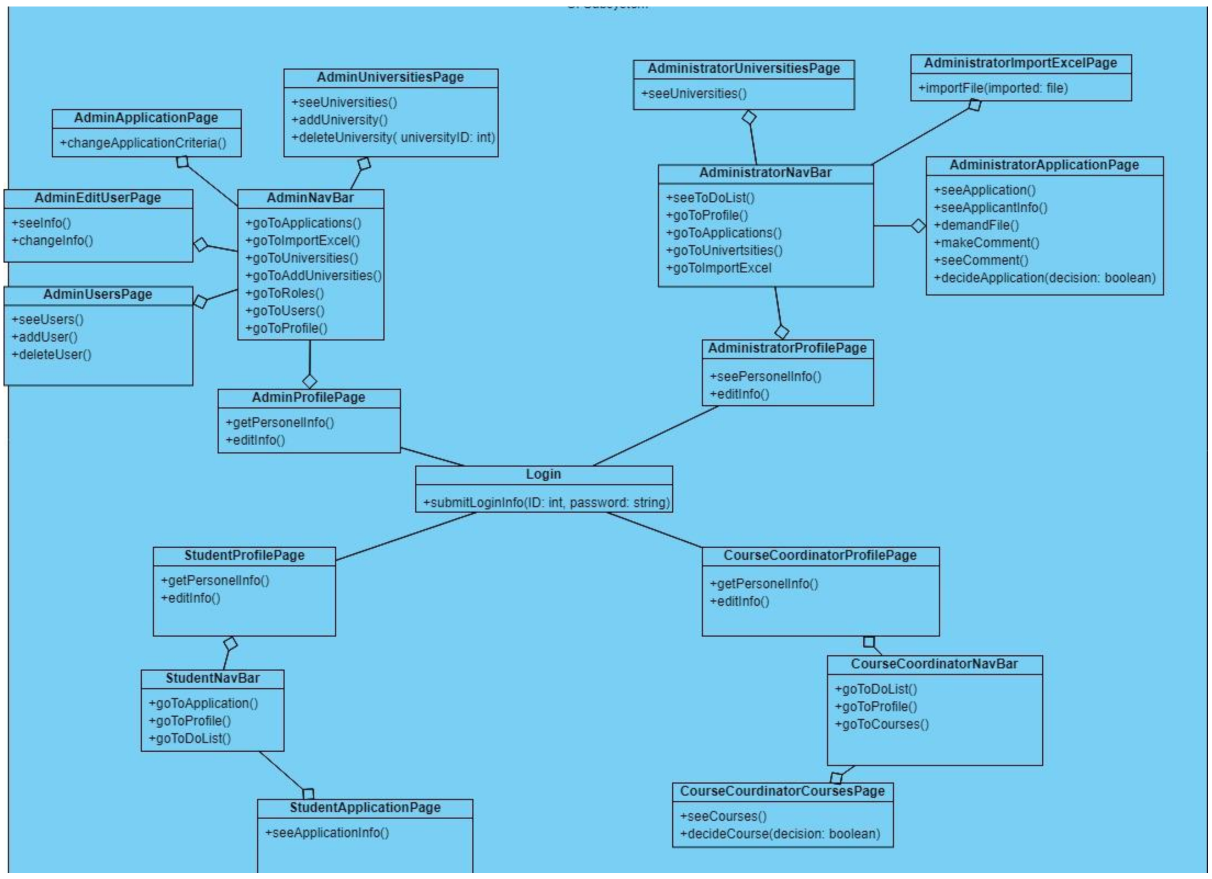
Right Side Of The Diagram:



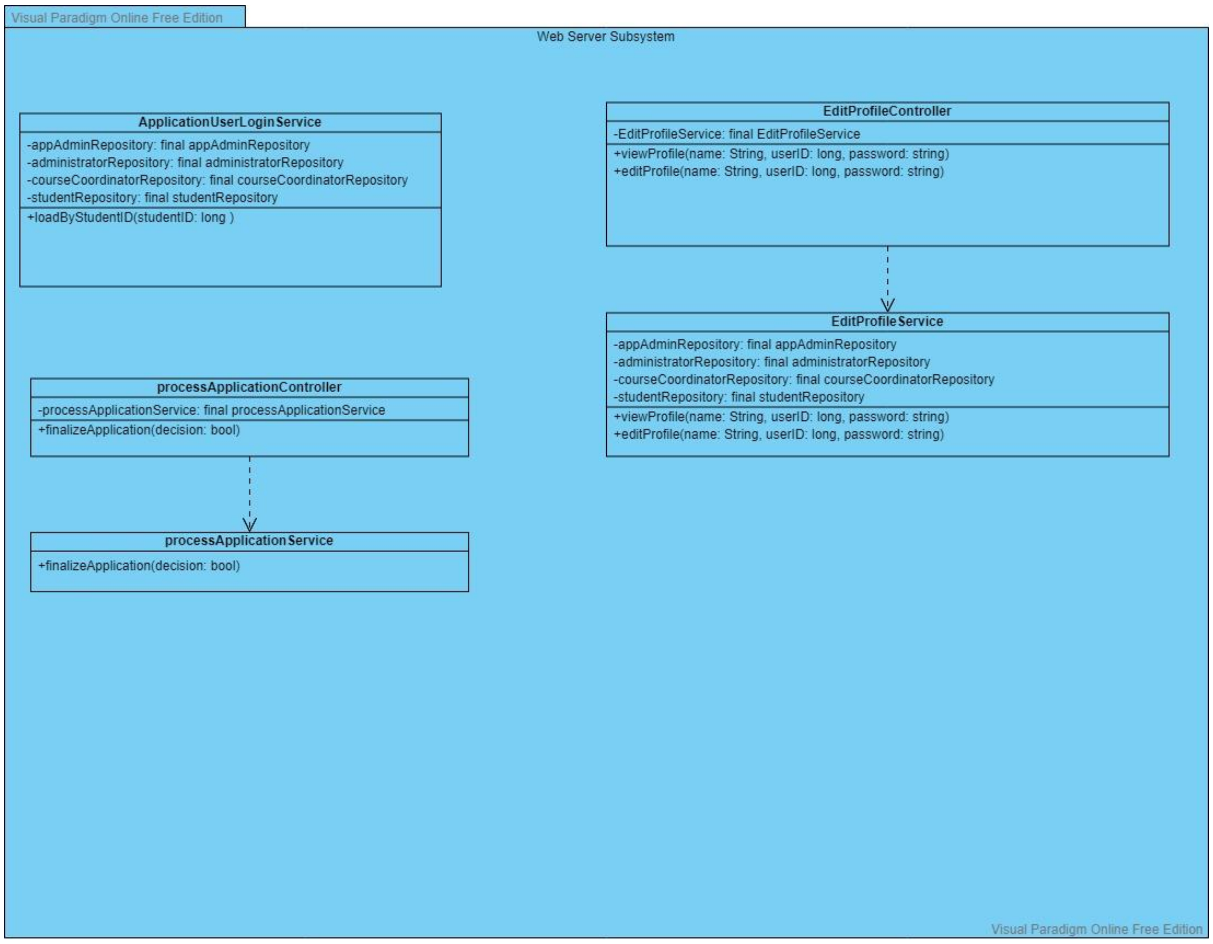
el
Main Frame
Dashboard Panel
ent orad
Administrator Dashboard
CourseC Dashborad
Admin Dashboard
To-Do Page
To-Do List
Edit University Page
Import Excel Page
Download File Page
Edit User Page
Add University Page
CourseC
Admin
Roles
Comment
Comment Page
Administrator Profile Page
CourseC Profile Page
Admin Profile Page
Visual Paradigm Online Free Edition

3.3 Layers

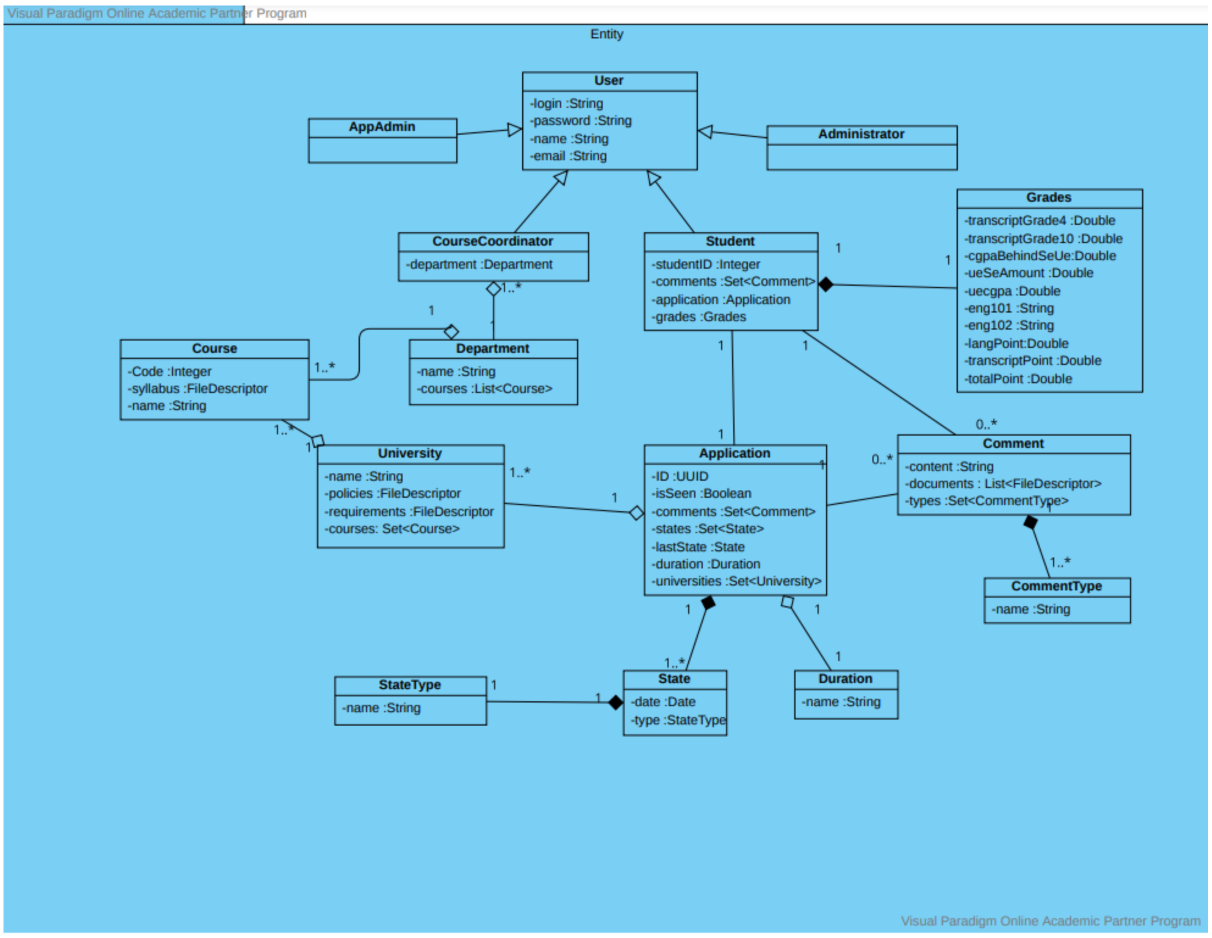
3.3.1 User Interface Management Layer

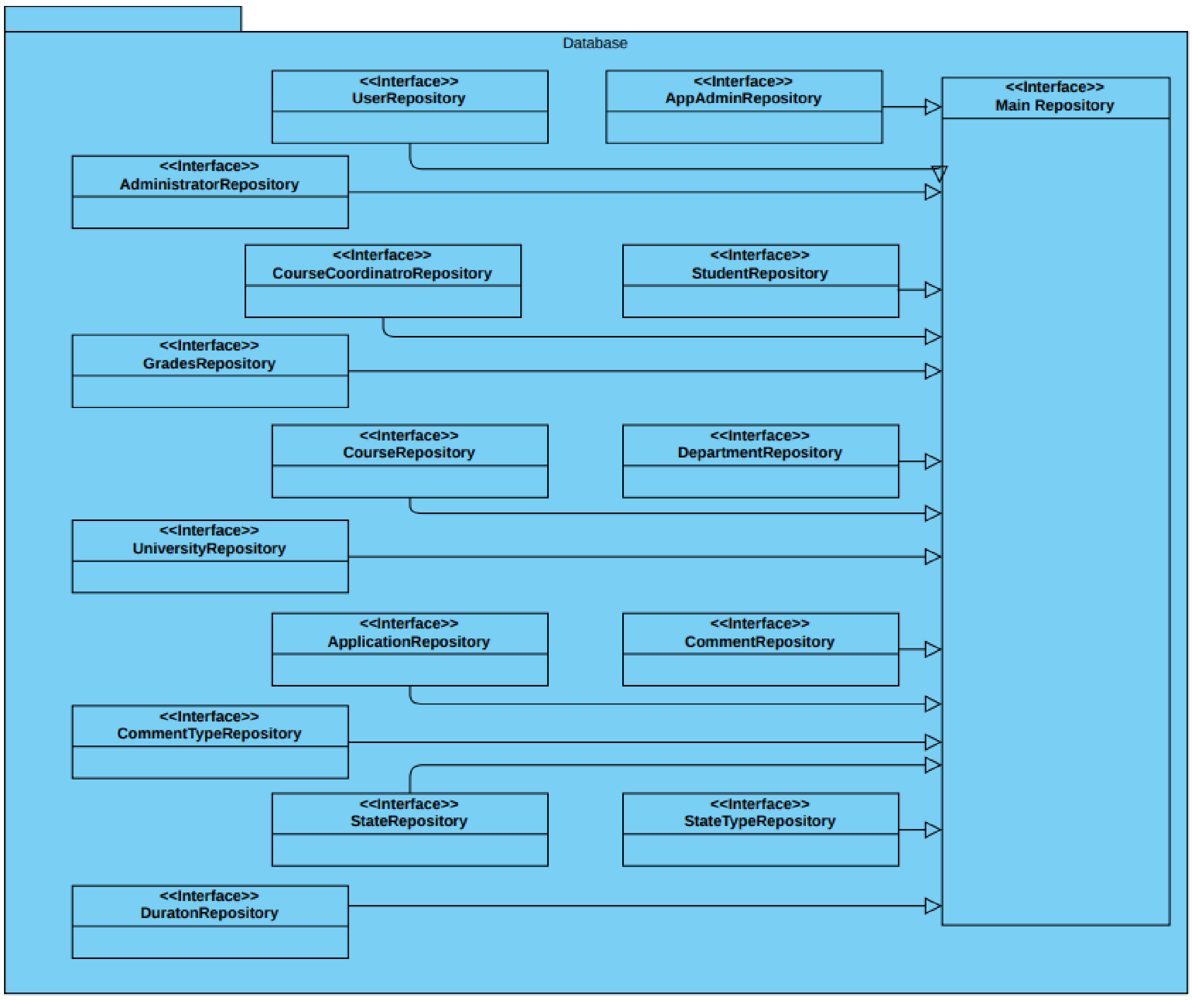


3.3.2. Web Server Layer


Visual Paradigm Online Free Edition
Web Server Subsystem
ApplicationUserLoginService
-appAdminRepository: final appAdminRepository
-administratorRepository: final administratorRepository
-courseCoordinatorRepository: final courseCoordinatorRepository
-studentRepository: final studentRepository
+loadByStudentID(studentID: long)
EditProfileController
-EditProfileService: final EditProfileService
+viewProfile(name: String, userID: long, password: string)
+editProfile(name: String, userID: long, password: string)
EditProfileService
-appAdminRepository: final appAdminRepository
-administratorRepository: final administratorRepository
-courseCoordinatorRepository: final courseCoordinatorRepository
-studentRepository: final studentRepository
+viewProfile(name: String, userID: long, password: string)
+editProfile(name: String, userID: long, password: string)
processApplicationController
-processApplicationService: final processApplicationService
+finalizeApplication(decision: bool)
processApplicationService
+finalizeApplication(decision: bool)
Visual Paradigm Online Free Edition

3.3.3 Data Management Layer




Database
<<Interface>>
UserRepository
<<Interface>>
AppAdminRepository
<<Interface>>
Main Repository
<<Interface>>
AdministratorRepository
<<Interface>>
CourseCoordinatroRepository
<<Interface>>
StudentRepository
<<Interface>>
GradesRepository
<<Interface>>
CourseRepository
<<Interface>>
DepartmentRepository
<<Interface>>
UniversityRepository
<<Interface>>
ApplicationRepository
<<Interface>>
CommentRepository
<<Interface>>
CommentTypeRepository
<<Interface>>
StateRepository
<<Interface>>
StateTypeRepository
<<Interface>>
DuratonRepository