Bilkent University

Department of Computer Engineering

CS 319 Term Project

ErasmusNET

Group 3F

Arda Baktır, Beyza Çağlar, Gülin Çetinus, Mehmet Kağan İlbak, Zeynep Selcen Öztunç

Instructor: Eray Tüzün
Teaching Assistant(s): Metehan Saçakçı, Emre Sülün, Muhammed Umair Ahmed, İdil Hanhan and Mert Kara

Design Report

December 11, 2022

Contents

1. Introduction

1.1 Purpose of the system

ErasmusNET is a web-based manager for the Erasmus program, aiming to aid outgoing students, incoming students, department coordinators, and international office coordinators with their responsibilities. The system's primary purpose is to make every part of the Erasmus procedure easier by uniting every operation the students and the coordinators have to perform so that every problem can be dealt with in one place. For instance, the students can upload multiple required files while making an application or make an appointment with the coordinators to discuss any issues they may have during the application process. The coordinators can directly resolve the student's problem or arrange a meeting. They can also see and approve all the outgoing students' pre-approval forms in one place without the risk of them being scattered or lost in the hundreds of emails sent to the coordinators.

1.2 Design Goals

The design goals of ErasmusNET are inspired by the non-functional requirements specified in the Analysis report. These goals are essential specifications because they are aimed at enhancing the user's experience.

1.2.1. Reliability

The program should run without errors on all platforms for a reliable result. An unreliable system may result in significant delays and other issues. If there is an error on the website, all the data entered or edited by the user should be saved in the database without losses. The data in the system must be correct, and false or incompetent inputs should be warned.

1.2.2. Security

ErasmusNET contains sensitive user information along with usernames and passwords. Therefore, the system must be secure. Firebase encrypts each request to access data on the database. Moreover, the requests between the back-end and front-end software will also be encrypted to provide more protection.

2. High-Level Software Architecture

2.1 Subsystem Decomposition

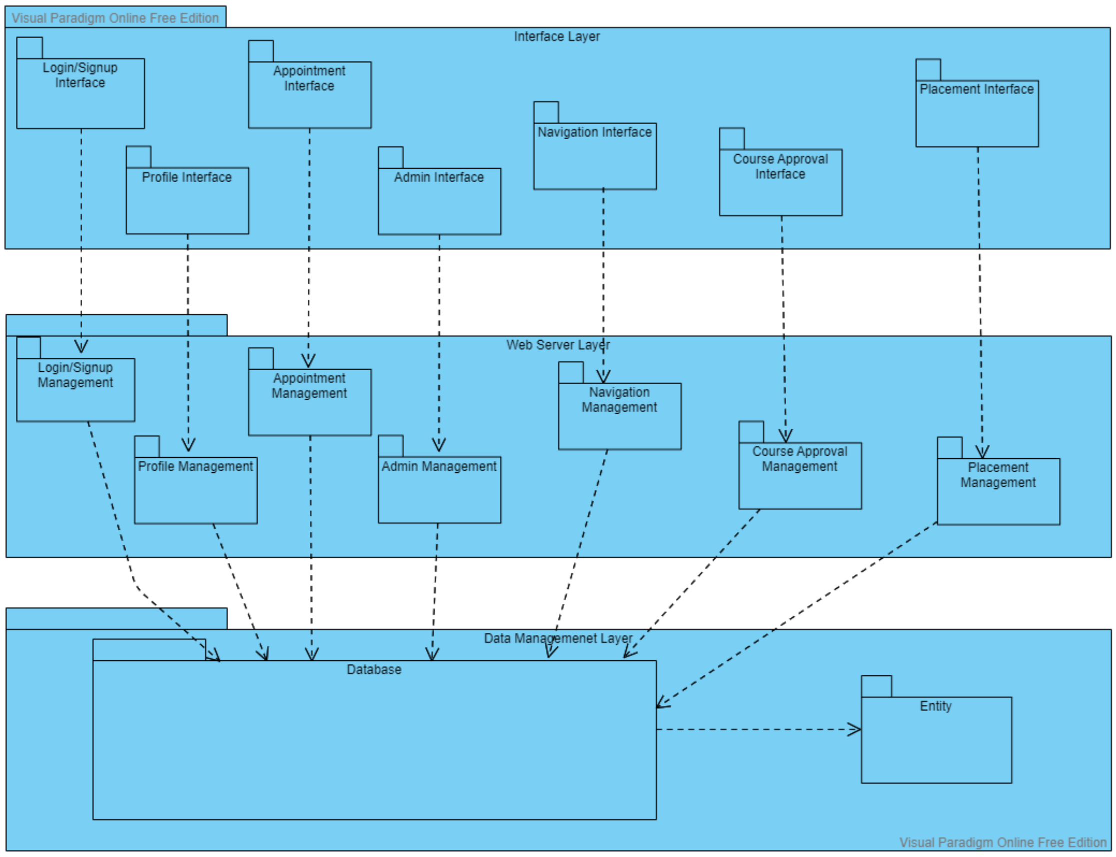

Figure 1: Subsystem Decomposition of ErasmusNET

Our project will consist of three layers: Interface Layer, Web Server Layer, and Data Management Layer. The Interface Layer has the pages for ErasmusNET. This layer corresponds to the boundary objects of the project. The user can interact with the pages to use the offered functionalities. Each UI element will be designed based on usability and extensibility design goals.

The Web Server Layer contains all the subsystems to manage the Interface Layer in the backend. For example, Student List Management will fetch the necessary student information from the database to be displayed in the Interface Layer. It has classes and interfaces for viewing and managing documents and personal information of students. Like the Student List Management, other subsystems in this layer manage different parts of the ErasmusNET, and all of these subsystems depend on the database for fetching data. The separation of these subsystems provides a maintainable and efficient system.

The Data Management Layer has the Database and Entity subsystems. The Database subsystem communicates with the database, whereas the Entity has Entity classes and communicates with the Entity objects.

2.2 Hardware/Software Mapping

ErasmusNET is a web-based software that runs on a web server. There is no need for special-purpose hardware. The software should be available as long as the users have a device connected to the Internet, have an up-to-date web browser, and have an up-to-date operating system. A minimum of 8 GB RAM and an Intel Pentium 4 or later processor is enough to host our website. Node.js should be downloaded to host the website since the React framework is used for the frontend of the project. Because React supports browsers such as Edge, Opera, Google Chrome, and Safari, the program can be run on these browsers. React depends on npm packages, and Node.js version 18.12.1 and npm version 9.2.0 is used while building the project.

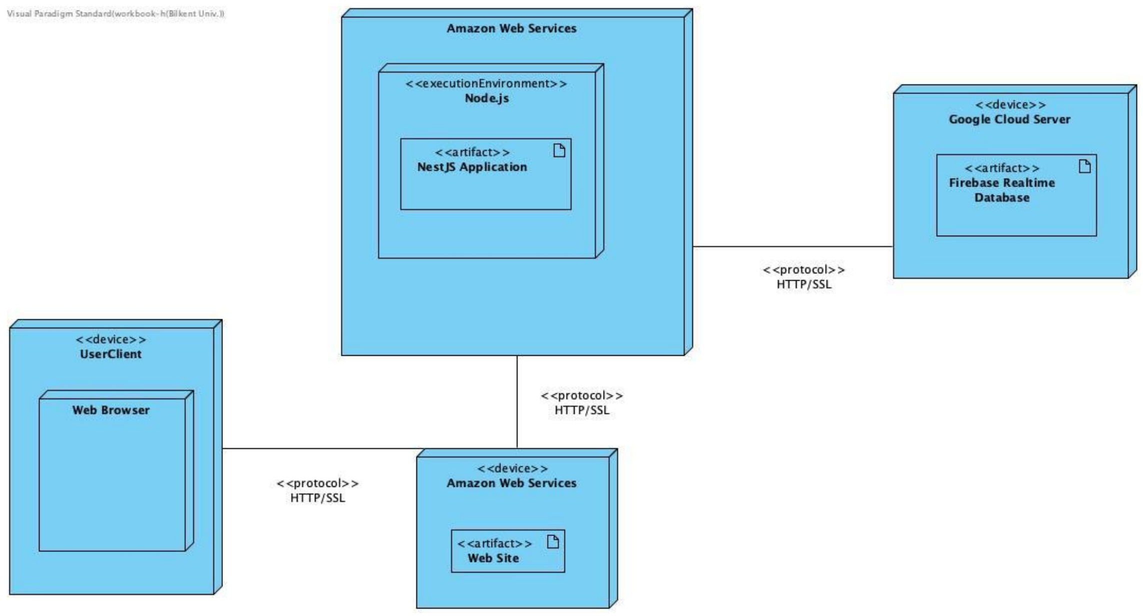
Visual Paradigm Standard(workbook-h(Bilkent Univ.))
Amazon Web Services
<<executionEnvironment>> Node.js
<<artifact>> NestJS Application
<<device>> Google Cloud Server
<<artifact>> Firebase Realtime Database
<<protocol>> HTTP/SSL
<<device>> UserClient
Web Browser
<<protocol>> HTTP/SSL
<<protocol>> HTTP/SSL
<<device>> Amazon Web Services
<<artifact>> Web Site

Figure 2: Deployment Diagram of ErasmusNET

2.3 Persistent Data Management

Persistent data is stored in both Firebase Firestore and Firebase Cloud Storage. Data stored in cloud storage includes course syllabuses and relevant student documents that are needed to be stored, such as pre-approval forms and learning agreements. This data is accessed infrequently and is more suitable to store in a storage bucket instead of a database.

User data will be stored in the Firebase Firestore. This database is structured as a document to hold nested data for students, courses, universities, instructors, exchange coordinators, admins, placement list, and so on. The primary keys of these tables will be selected so that it uniquely identifies each record, such as the IDs of users.

2.4 Access Control and Security

Our project provides security and access control. Only the required amount of information is sent to the front end. When any of the users log in to the system, the system matches the credentials of the users to one of the user types. It determines the information that will be available to the user according to that type. All of the users can log in and have their profile page. They also can access the 'About' and 'Contact' pages, in which information about the exchange programs is displayed, and contact information on exchange-related people or institutions is given. The students have access to the student interface, where they can submit their pre-approval form, make appointments with coordinators, and upload syllabi of the courses they want to take. They can also view many pages such as Knowledge Library, Student Stories, and Document Templates.

The coordinators have access to the coordinator interface, where they can approve/ reject applications and appointment requests and view information about students. The information may include the student's placed university, contact information about the student, and the student's first-degree relatives.

Admins have access to the admin interface. They can create, modify and delete deadlines, view every user's info, and modify information about the program, such as the contact numbers and description of the program.

	Student	**Coordinator**	**Instructor**	**Admin**
Login	X	X	X	X
Signup	X	X		X
View Syllabus	X	X	X	X
Add Syllabus	X			X
Approve/Reject Course			X	X

Approve/Reject Pre-approval Form		**X**		**X**
View Profile	**X**	**X**		**X**
Edit Profile	**X**	**X**		**X**
Add Appointment	**X**	**X**		**X**
View Appointment	**X**	**X**		**X**
See Student List		**X**		**X**
Upload Placement List				**X**
View Placement List		**X**		**X**
Add/Remove Deadline				**X**
Edit Deadline				**X**
View About/Contact	**X**	**X**	**X**	**X**
Edit About/Contact				**X**

2.5 Boundary Conditions

2.5.1 Initialization

ErasmusNET runs on the web; therefore, there is no need to install the software. As mentioned above, the only requirement to access the website is a device with Internet access and web browsers.
The program is deployed through Firebase's hosting system and the database will be connected to the Firebase database, instead of a local one, as well.

2.5.2 Termination

ErasmusNET is terminated in stages. First step is to terminate the webapp server. Second step is terminating the backend server. This is a measure to prevent data loss. Final step would be removing the read-write access from Firebase databases to ensure data integrity.

2.5.3 Failure

ErasmusNET's web server will run inside clusters of AWS. In an event of cluster failure, the app will be deployed to another cluster to provide a continuous service. The web app is written carefully to handle the unexpected behavior to prevent crashes. In addition, webpack catches the exceptions and logs them to the developer console for debugging.

Database, and bucket storage is handled by Google Firebase. However, in an event of failure, Node server will prevent further access to those storages for data integrity.

3. Low-Level Design

3.1 Object Design Trade-offs

- **Maintainability versus Performance:** Our project divides each object into its own entity, implementation and controller class. While it can cause a performance reduction due to OOP, it may actually boost performance if multiple services run in parallel. Also this approach is ideal for data abstraction and its maintainability as it can be changed controllably.
- **Memory versus Performance:** Since we are using objects for each datum this can cause a limitation in memory, but with these objects for each datum approach, everything would be more efficient.
- **Security versus Usability:** Since the login procedure consists of ID and password login, this decreases our security, but would increase the usability by being easy to access to the services.
- **Functionality versus Usability:** The main purpose of this project is to make the Erasmus application process more easy to use for all the students and the coordinators. Because of this, we need both functionality and usability. This causes the project to have decreased usability with all the functions needed for the project.

3.2 Final Object Design

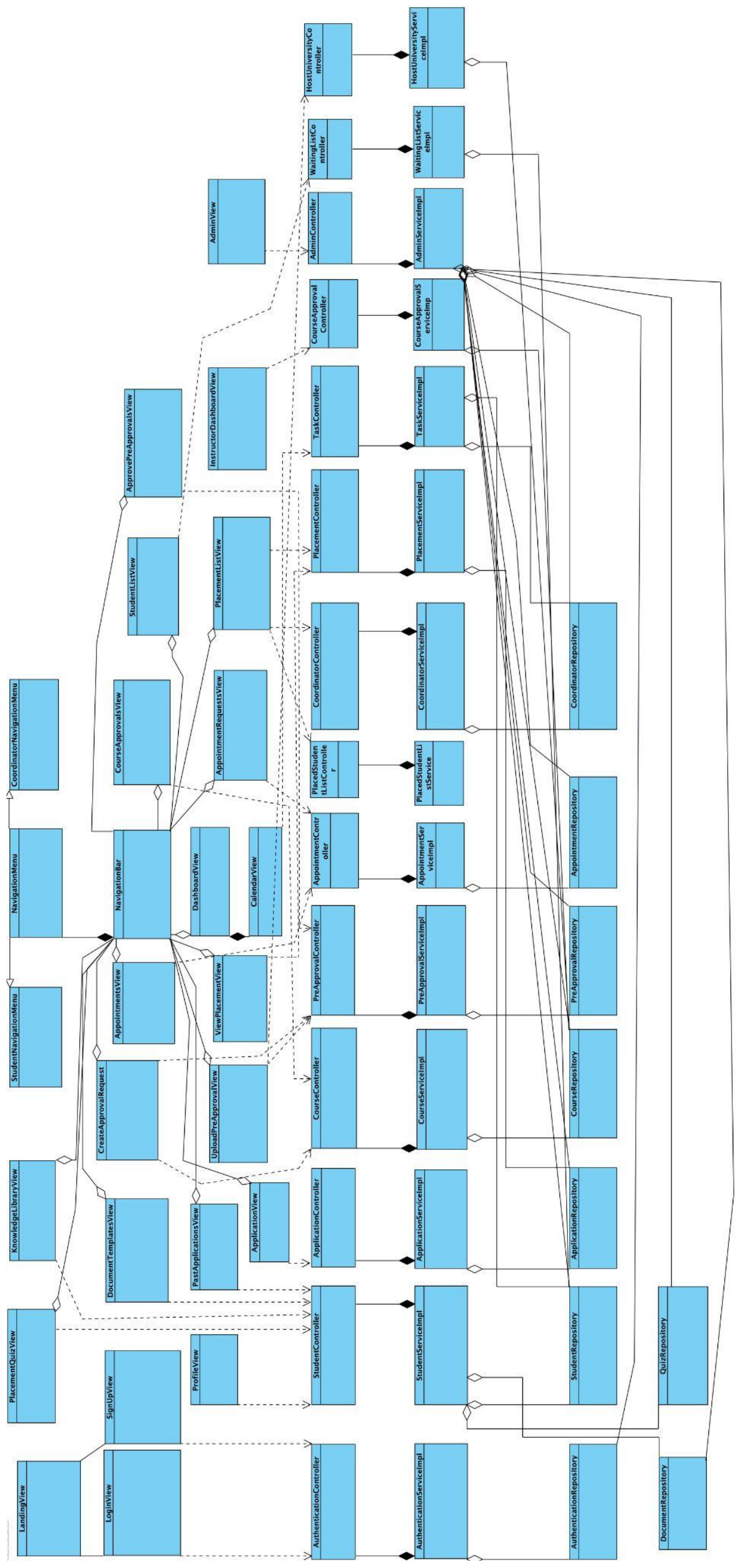

Figure 2: Diagram for Object Design

Left side of the diagram:

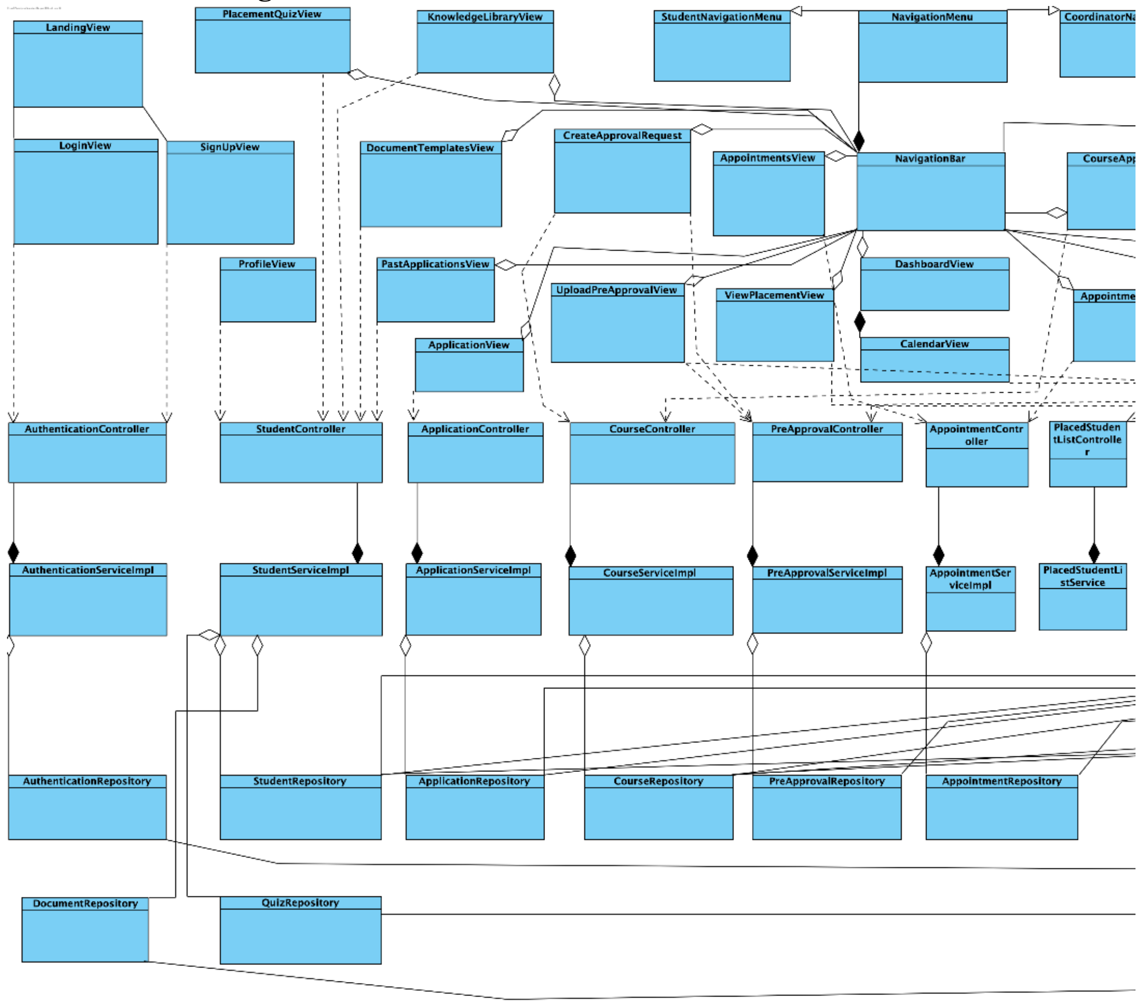

Figure 3: Left Side of the Object Diagram

Right side of the diagram:

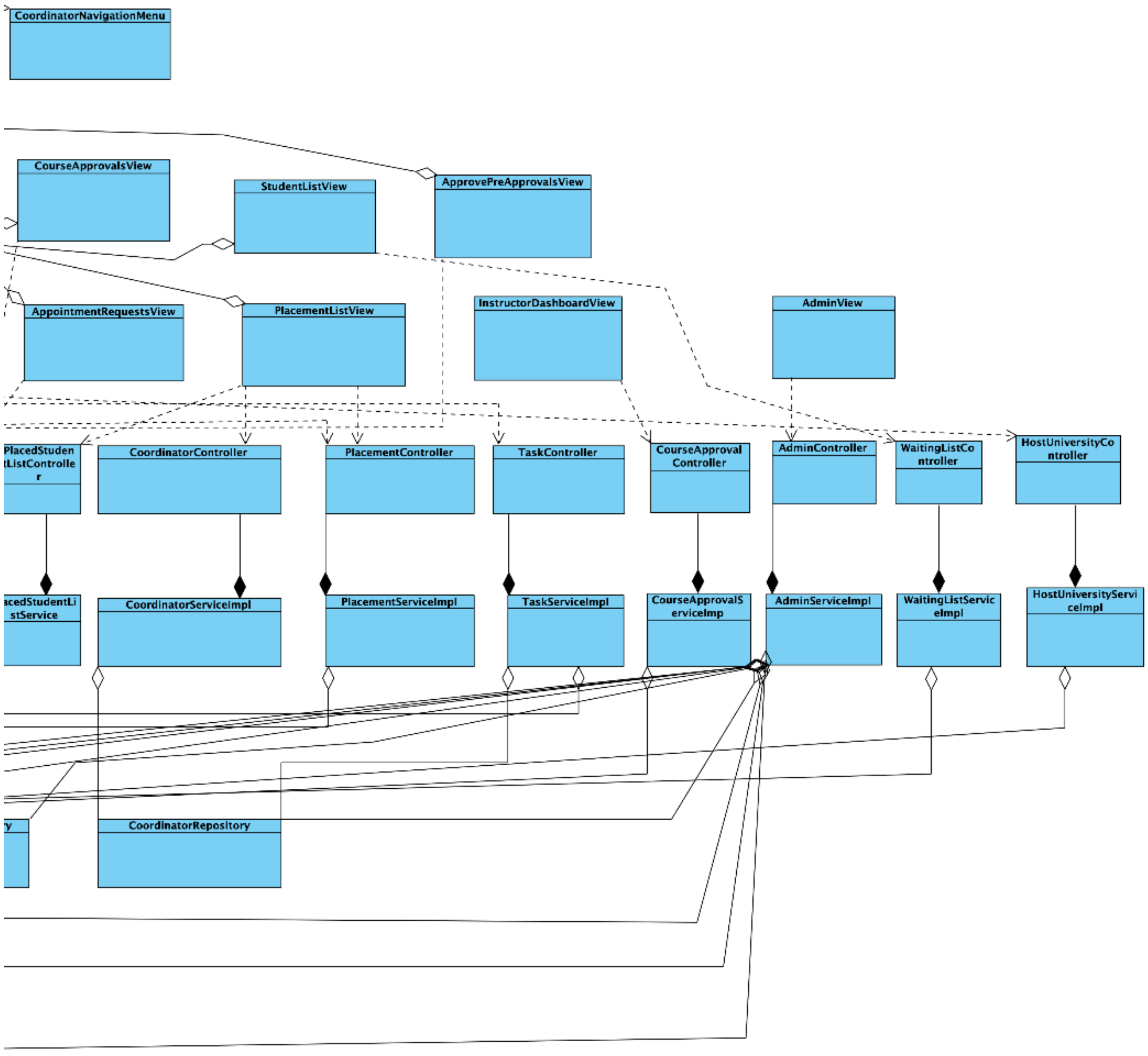

Figure 4: Right Side of the Object Diagram

3.3 Layers

3.3.1 User Interface Management Layer

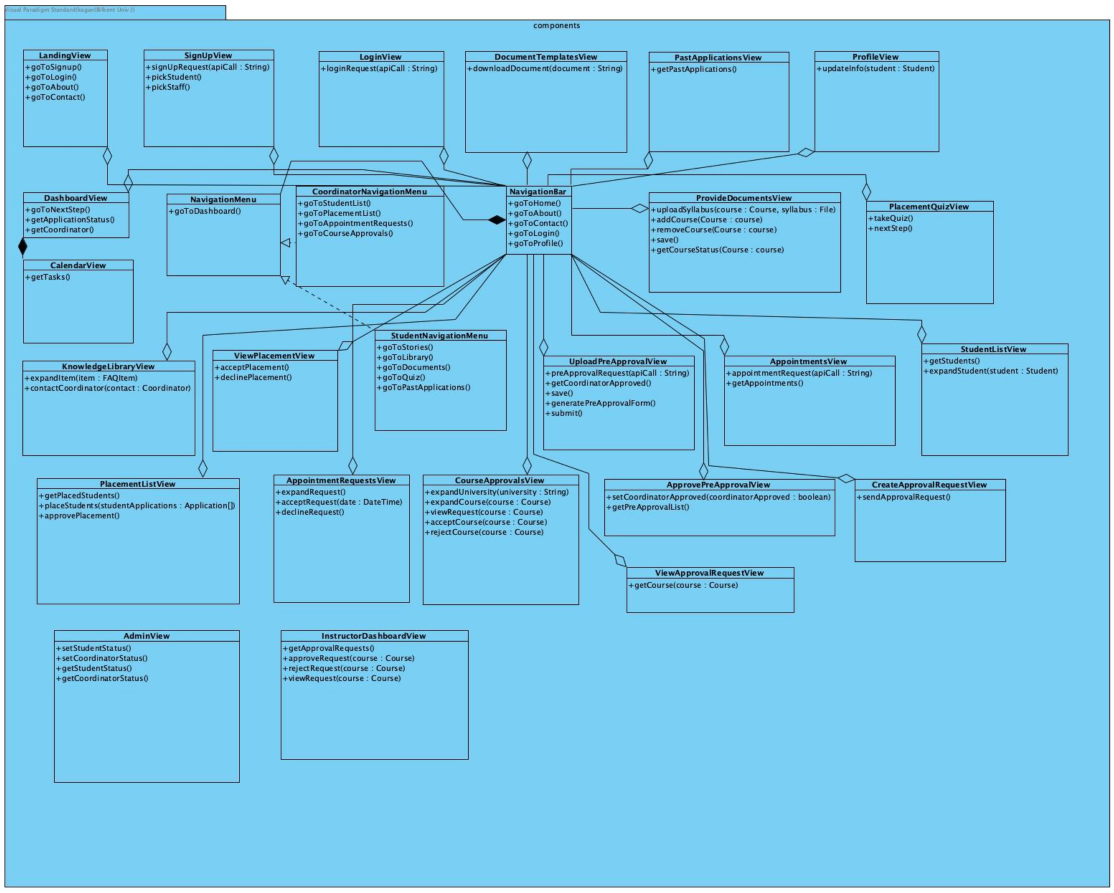

Figure 5: Class Diagram for the User Interface Management Layer

This is the boundary between the user and the controller. Classes in this layer are javascript objects and they communicate with the web server layer.

3.3.2 Web Server Layer

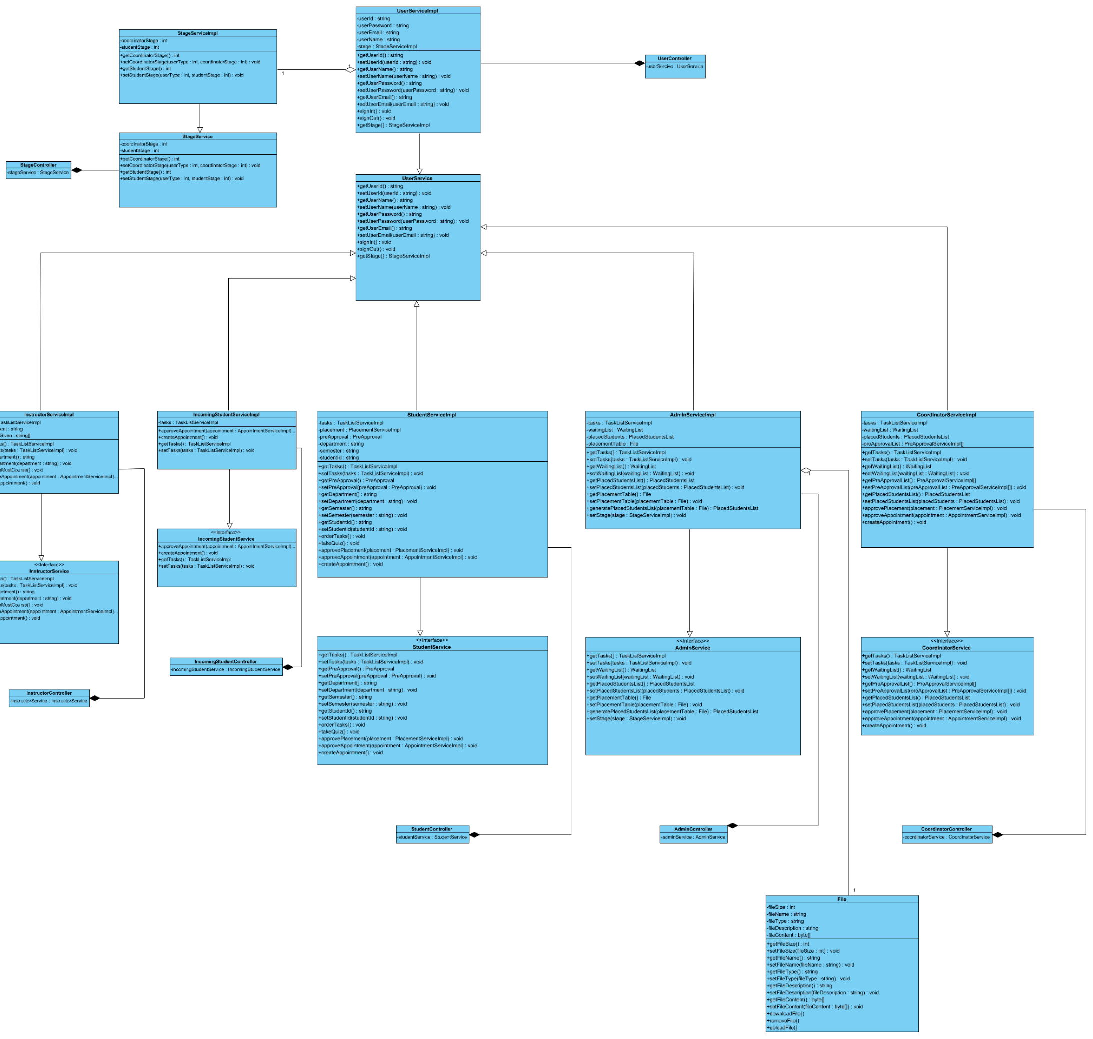

Figure 6: Web Server Layer's Student, Coordinator, Instructor, Admin and IncomingStudent Classes

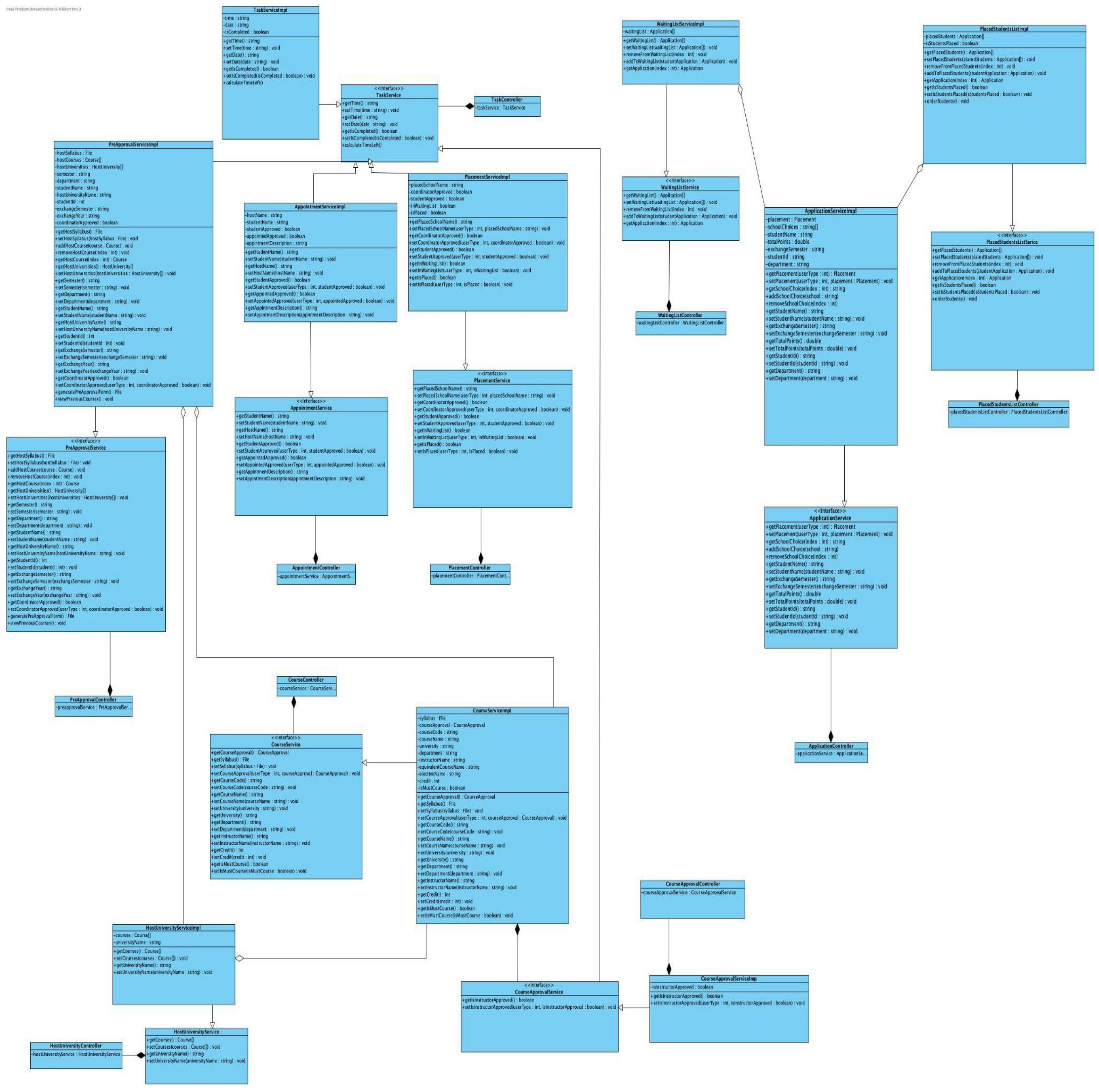

Figure 7: Web Server Layer's Appointment, Application, Course, Placement, PreApproval, Task, HostUniversity, WaitingList, PlacedStudentsList Classes

This layer acts as a controller between UI and data management layers. Primary responsibility of this layer connecting the UI to entities.

3.3.3 Data Management Layer

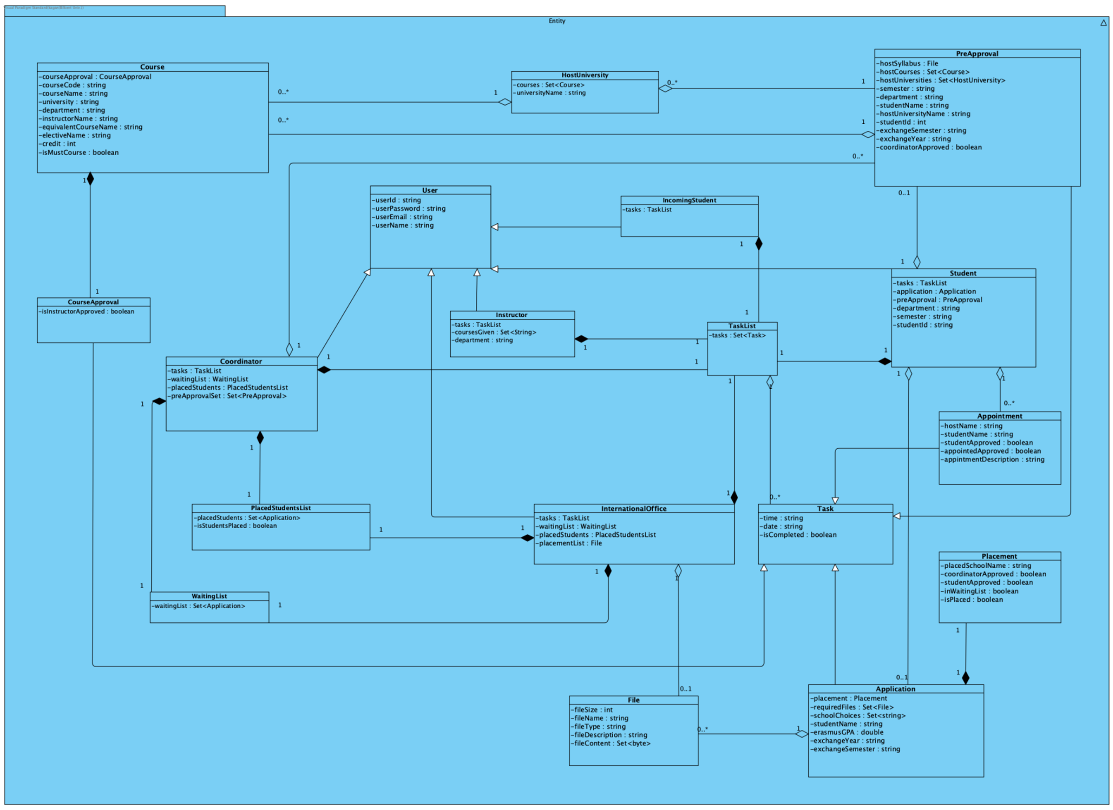

Figure 8: Entity Package for the Data Management Layer

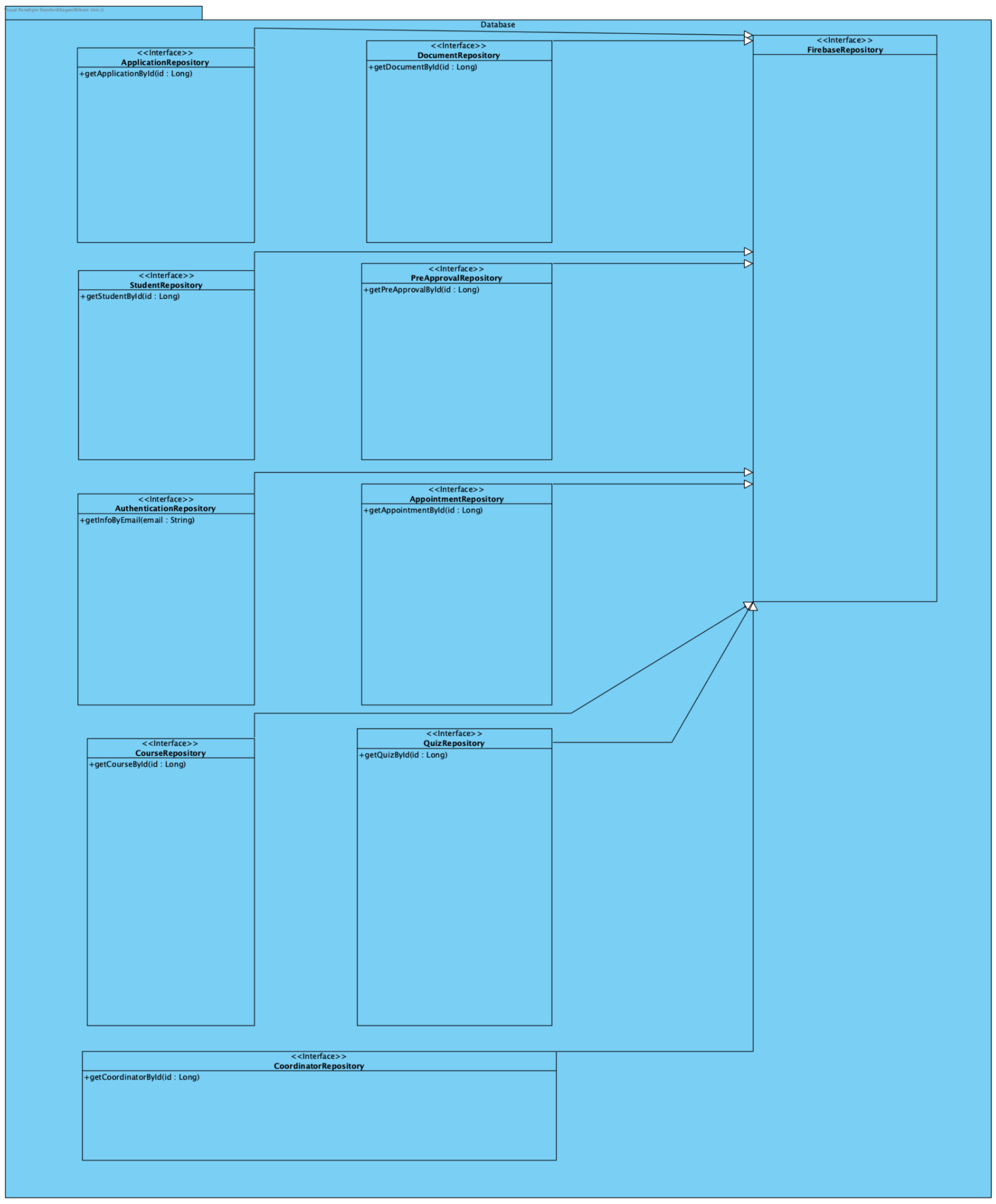

Figure 9: Database Package

This layer consists of Entity and Database subsystems. Primary responsibility of this layer is supporting the web server layer by managing and storing data.

3.4 Packages

3.4.1 Internal Packages

3.4.1.1 Entities

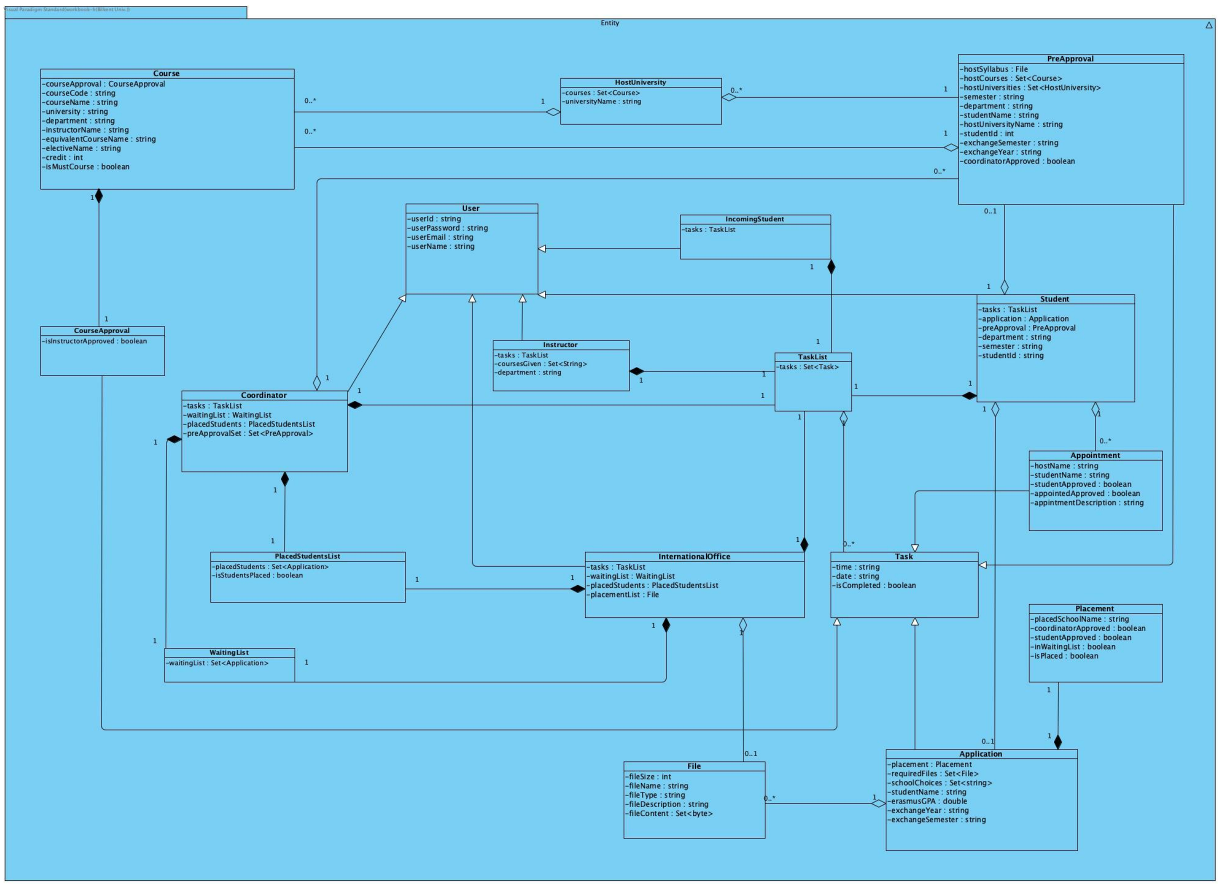

Figure 10: Entity Package

This package contains entity classes that contain methods and objects for managing the interactions and storing information.

For simplicity purposes we haven't included the getter and setter methods for each attribute.

3.4.1.2 assets

This package contains assets for the frontend of the website such as images.

3.4.1.3 src.components

This package contains the components of the website. Each component represents a small reusable UI element.

3.4.1.4 public
This package contains the basic HTML and needed icons for the frontend of the website.

3.4.1.5 src.components.pages
This package contains the individual views that are represented in the UI Subsystem diagram. These views themselves are components that are made out of other components.

3.4.1.6 src.components.common
This package contains the individual components that are common in almost all views. This package allows us to increase code reuse.

3.4.1.7 erasmusnet-api
This package contains the web server of the website.

3.4.2 External Packages

3.4.2.1 firebase
This package is provided by npm and contains all of the Firebase API including auth and firestore.

3.4.2.2 react
This package is provided by npm and contains the functionality of UI. It allows us to define views that interact with other views and backend.

3.4.2.3 react-dom
This package is provided by npm and contains libraries for creating HTML Document Object Modeling (DOM).

3.4.2.4 react-router-dom
This package is provided by npm and contains libraries for handling the routing inside DOM.

3.4.2.5 react-scripts
This package is provided by npm and contains scripts for building the react webapp.

3.4.2.6 web-vitals
This package is provided by npm and contains library functions for providing great user experience.

3.4.2.7 tailwindcss
This package is provided by npm and contains a library for defining css for individual DOM tags.

3.4.2.8 framer-motion
This package is provided by npm and contains library functions for animating DOM elements.

3.4.2.9 autoprefixer
This package is provided by npm and is a dependency of tailwindcss.

3.4.2.10 postcss
This package is provided by npm and is a dependency of tailwindcss.

3.4.2.11 nestjs
This package is provided by npm and is a framework for writing server-side applications in Javascript.

4. Improvement Summary

4.1. General

- The application process have been removed from the program as the old system will be used for the applications.

4.2. Design Goals

- The design goals were limited to the two that are the most important for the project.

4.3. Subsystem Decomposition

- The subsystem decomposition was updated to better serve the purposes of the project.
- The packages regarding the application process were removed.

4.4. Hardware/Software Mapping

- More detail was added on the development side of the software.

4.5. Persistent Data Management

- Detailed description of the data that will be stored in the database has been included.

4.6. Access Control and Security

- An access matrix was added to demonstrate the authentications of each user.

4.7. Boundary Conditions

- Boundary conditions were updated to give more information on how the system works.

4.8. Final Object Design

- Refactored and added more objects since related subsystems were modified.

4.9. Layers

- Refactored and added more views to UI Subsystem.
- Business layer got updated with new class diagram

4.10. Packages

- Extended the external packages since we added more dependencies to our project.

5. References

[1] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0