Bilkent University

Department of Computer Engineering

CS 319 Term Project

Section 3

Group 3C

Bilkent Erasmus-Exchange System

Design Report

Project Group Members

Mehmet Eren Balasar
Mehmet Hasat Serinkan
Tuna Okçu
Yüksel Berkay Erdem
Aftab Fakir

Instructor: Eray Tüzün

TA's:

Muhammed Umair Ahmed

Mert Kara

İdil Hanhan

Emre Sülün

Contents

1 Introduction

1.1 Purpose of the System

The web-based Bilkent Erasmus-Exchange System software is an application that facilitates the work of both students and officials. In addition to uploading and downloading document systems, Bilkent Erasmus-Exchange System creates a chance to get notifications about deadlines and news about the process for both students and officials. The design of this application aims that most of the work about the Erasmus-Exchange process can be done through our software to make the process well-organized and easier. In this line, the important points of the design of the application are being user-friendly, fast and reliable.

1.2 Design Goals

The design goals of the application were determined in the part "Non-Functional Requirements" of the analysis report. Bilkent Erasmus-Exchange System should be user-friendly as well as secure and fast. Sensitive data should be protected from any danger and the performance of the system should be fast as possible. However, our main two design goals are "usability" and "security". Along with them, it is important to mention "performance", "portability and compatibility", "reliability, maintainability and availability".

1.2.1 Usability

Usability is the key standard for the design of an application. It must be considered so that the application can be used by the users. In this line, the Bilkent Erasmus-Exchange System must be easy to use for new users. The designs of the pages must be easy to understand and use so that users can navigate and find what they are looking for quickly. The "Exchange" application must meet this need by designing the pages according to new generation methods. The interfaces must be easy enough for users to find whatever they need without having to search through all of the details. The application must have different interfaces for different levels of authority to avoid any complications. Most useful functions in the application must be located in the main perspective of a user's view on the pages.

1.2.2 Security

The application database should be secure because it will store the data that must be protected. Since sensitive information regarding user's academic and personal life will be stored

on the database, rather advanced encryption and encryption algorithms should be implemented. Every user has their own random algorithm for encryption and decryption so that only the system can reach it, not admins or any dangerous threat from outside.

1.2.3 Performance

The performance criterion is one of the milestones of the design of the application, of course. To create a comfort zone for the user with the features of the application, it is very critical to do this very fast. However, Erasmus-Exchange processes are often long, and time limitations are on the scale of days. So, focusing on fastness is not our number one priority. Also, consider this: The application should work without slowdown when the simultaneous user count is high, but every year, we will have ~400 users registered to the system. That is not a really high user count. That means the system will not be put on stress most of the times. Yet, of course, database implementations should be well-organized to make "less actions – to get more data".

1.2.4 Portability and Compatibility

All kinds of users have many different devices and different browsers on these devices. Therefore, the application must satisfy the user while the same user uses different devices and browsers. In order to accomplish this, the application must use useful libraries and frameworks, which can be compatible with different browsers, as well as new ones.

1.2.5 Reliability, Maintainability and Availability

The Erasmus and Exchange process has sensitive cases in which there should not be any complications. Therefore, the most important use cases must have a very high percentage of success. The crashes should not affect all of the application parts, they should stay local. Also, these crashes must be solved as much as possible. Additionally, the database should be backed-up at short intervals as a precaution against any problem.

1.3 External Packages and Frameworks

1.3.1 React.js

This library is used in the front end to declaratively program UI elements, reducing the amount of code to write while adhering to the principle of modularity, in turn improving readability and writability.

1.3.2 Next.js

This React framework is used to simplify the deployment of React-based web programs.

1.3.3 Django

This is a back-end framework written in Python. The use of Python makes the writability and readability of the code higher. Also, Django includes a built-in REST framework integration, which we will use to create APIs.

1.4 References

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.

2 Proposed Software Architecture

2.1 Overview

Bilkent Erasmus-Exchange System is an all-in-one web-based application that will help students, coordinators and all other users who are somehow involved in the Erasmus-Exchange procedure in Bilkent have a smooth and spotless Erasmus-Exchange experience. The main input data sources of this system will possibly be Bilkent SRS (Student Registration System), Bilkent Exchange Office and users directly. SRS will be used to authorize and verify users. There will be different tabs for different functionalities. Our system has a 4-layer architecture meaning there is a helper subsystem between application logic layer and database layer. We tried to elaborate on every subsystem component:

2.2 Subsystem Decomposition

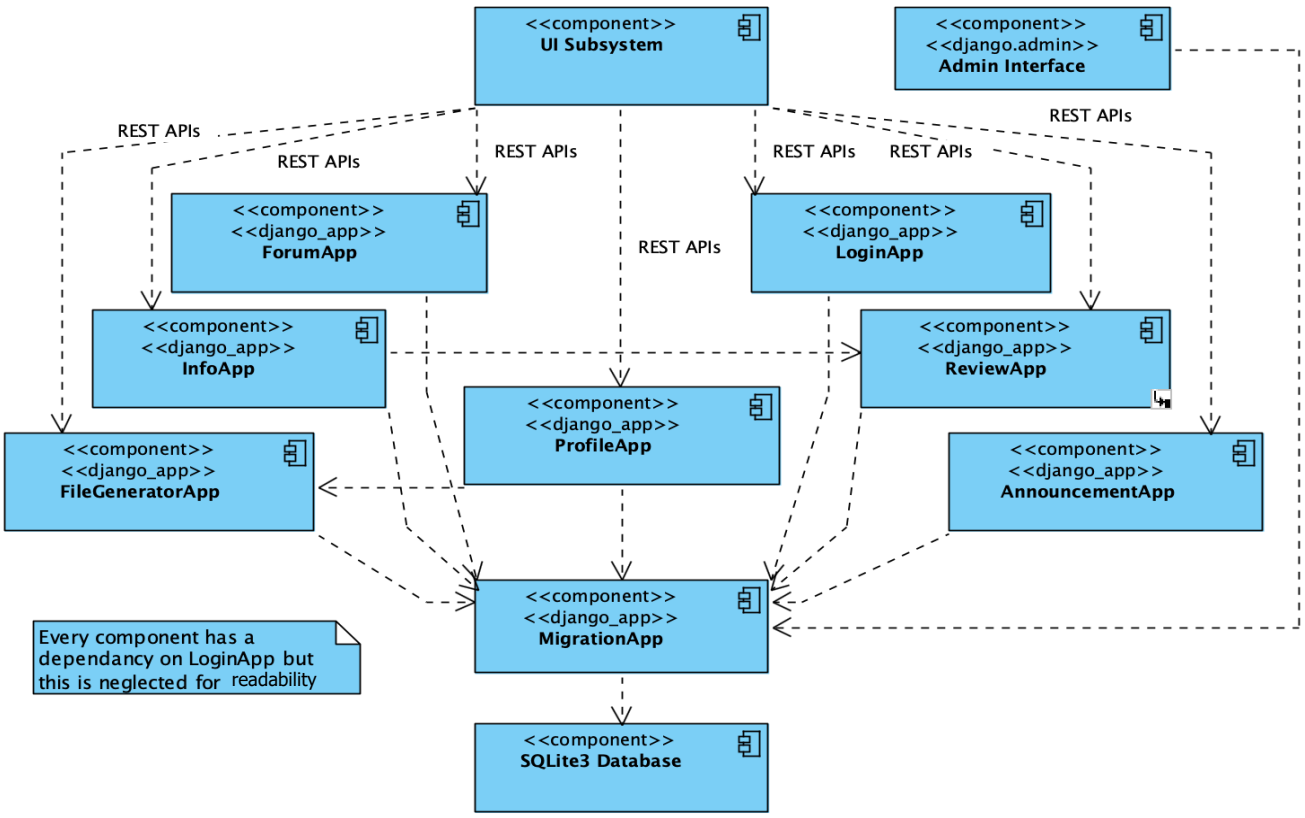

Figure 2.2.1: Subsystem Decomposition

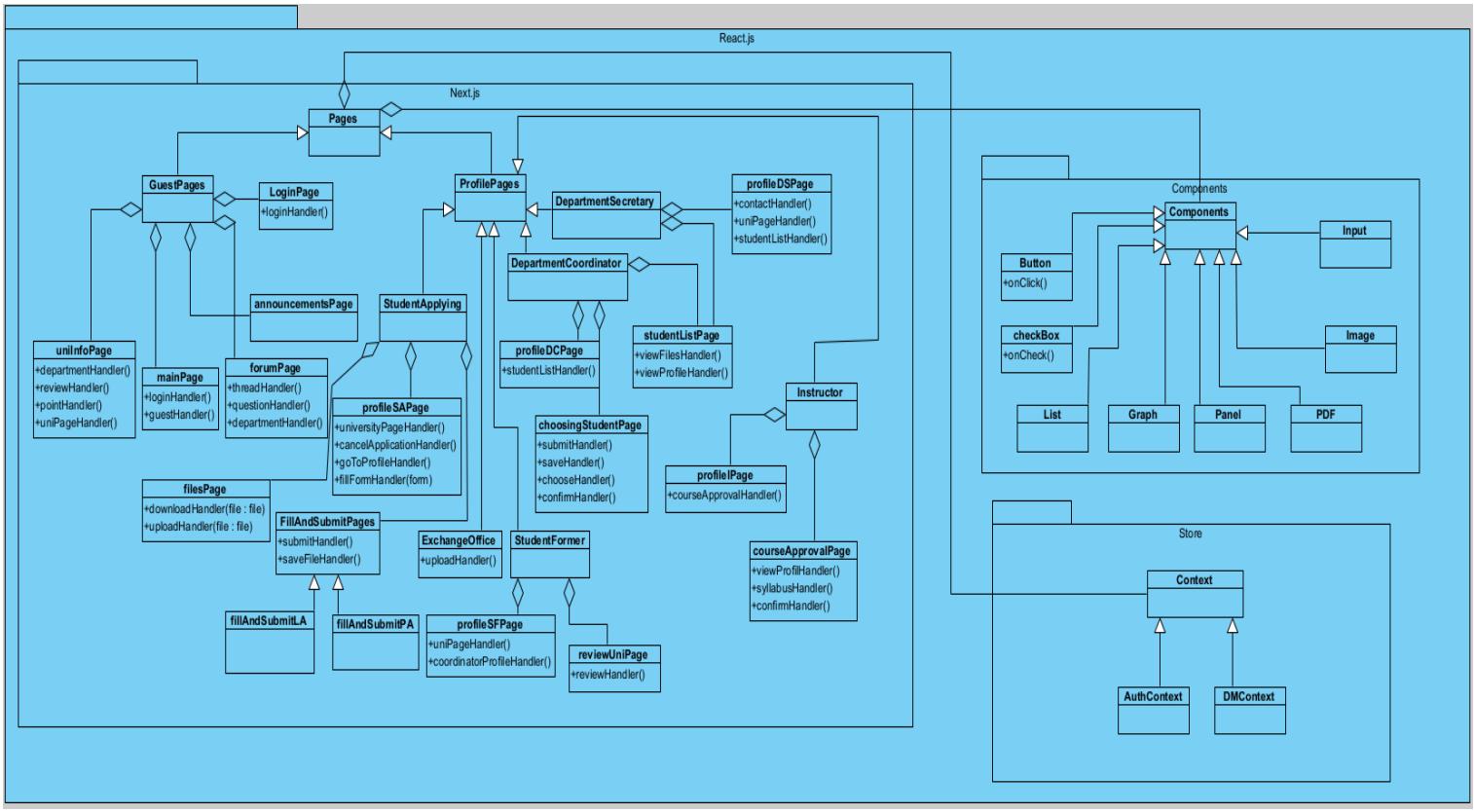

Figure 2.2.2: User Interface Subsystem

This subsystem acts as a boundary object between the user and the application layer. All classes in this subsystem are in React.js format. The page classes are in Next.js format which is a React framework. The store is in React Context format and the components are in React.js format. The User Interface subsystem has dynamic components that work with their methods due to React.js components.

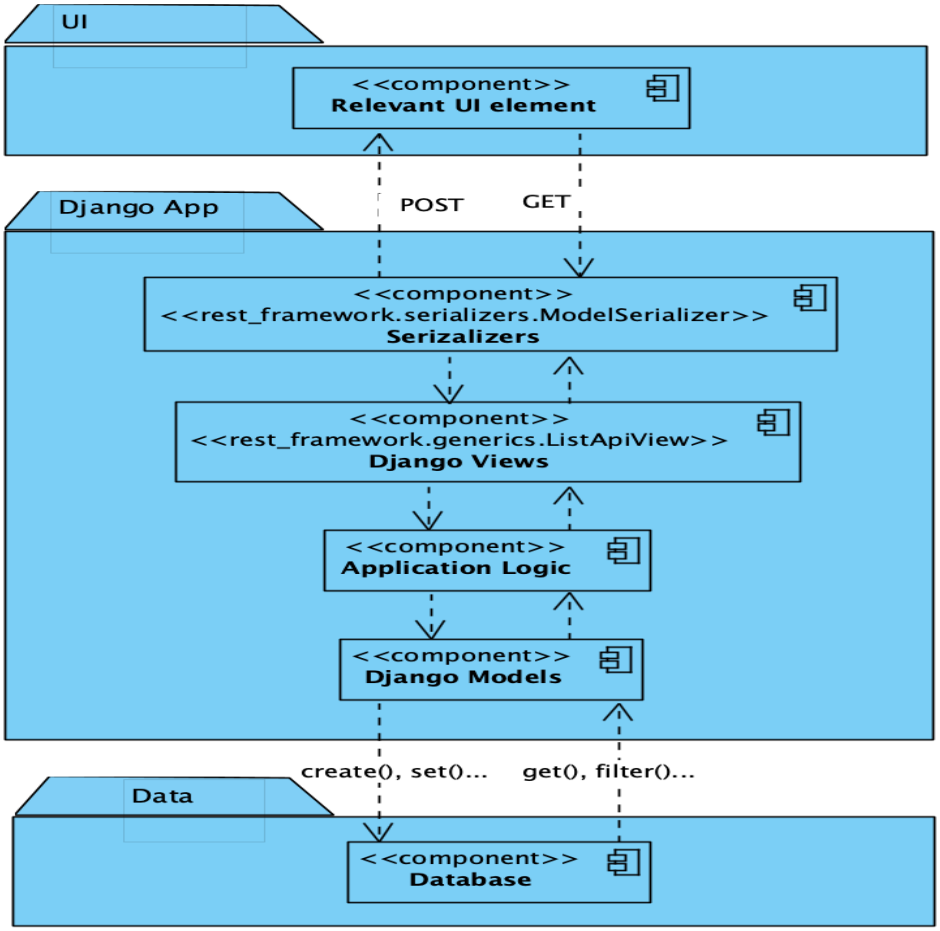

Figure 2.2.3: App Scale Subsystem Decomposition (inside each app component)

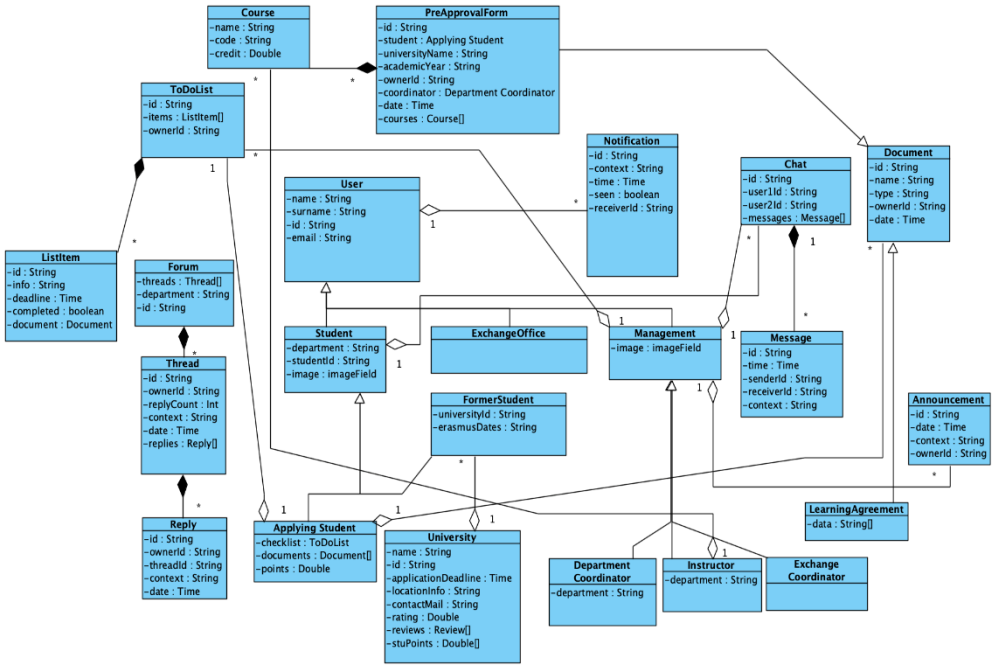

Figure 2.2.4: Data Subsystem

2.3 Hardware/Software Mapping

Bilkent Erasmus-Exchange System does not really require any specialized, high-quality hardware for both client-side and server-side infrastructures. However, our estimations show that:

- Permanent (almost permanent) data will occupy approximately 5 GB of space (~50 management users × 3MB (1MB Image + 2MB other) + system files, partner university data etc.)
- Every year, on average, 400 students are accepted and will be using the system.
  We can say that a student object will have a size of 5MB at most. 1MB image + 4MB of pdf files and other data. That means every year +2GB of data is uploaded to database. For a 5-year plan, we should need 10GB.
- In conclusion, 10GB + 5GB = 15GB. To follow 15/85 principle (85% should be full), 17GB (there is no standard size as 17GB of storage on the market, so 20GB) of disk storage will be enough.

In terms of processor and RAM:

- 20% of disk size = ~4GB of RAM, quad-core 2.5GHz of CPU will be enough to handle ~30 concurrent connections.

Since our system runs on web, devices that will be used to connect to the system should have web browsing features. The system is optimized to run on computer web browsers, but it is also possible to access the website via mobile browsers, though we do not guarantee that it will be fully functional on mobile.

Our system uses Django Web Framework 4.1.3, REST API, and Python 3.9.9 for its backend, SQLite3 for database systems and Next.js, React.js, CSS, HTML5 for its frontend. The details of our website's deployment are not yet finalized.

All popular web browsers including Chrome, Firefox, Safari, Opera, Edge, and Internet Explorer 9 and above will support website. However, some polyfills may be required for older browsers since React is not fully compatible (older versions than Chrome 49, Firefox 50, Safari 10, IE 9).

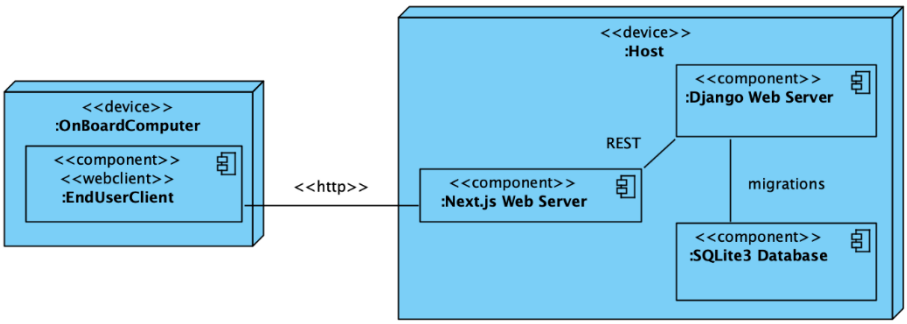

Figure 2.3.1: Deployment diagram

2.4 Persistent Data Management

SQLite3 will be used as the database engine for our system. Since Django makes it fairly easy to deal with databases, finding a proper database engine was not our team's main focus. The default is SQLite3 in Django, which fulfill all our needs. Our two primary requirements for a database were it being a relational database (since we want to integrate OOP principles smoothly) and simple to use to with Django. SQLite3 has these features. All entity objects (see Figure 2.2.1) in our system will be stored in the database including historical data. We are communicating with the database not directly but using Django models, which enables us to focus more on the business logic for our system than on quarrelsome database operations.

2.5 Access Control and Security

Security is one of the main concerns for Bilkent Erasmus-Exchange System since there will be sensitive information and academic processes going on in the system. We have many actors who will be using the system and there are specific pages that only some actors can access. Every user has different permissions. In production, an external login system, which is Bilkent SRS Login System, will be used. Details of this is not yet decided. However, for development purposes, we implemented a token-based authentication system. With token-based authentication, a user receives a special token in exchange for their username and password. The token will be produced, saved in the database for future usage, and given to the user each time they log in. The user may use this token to approve each API call they make because it is specific to them. It is no longer necessary to supply the username and password with each request thanks to token-based authentication. It is considerably safer and works best for client-server communication, such when a web client powered by JavaScript communicates with a backend application using REST APIs.

2.6 Global Software Control

Our system's control flow mechanism is mainly event-driven control since the user will input the data generally and will select the wanted data among the pages. However, there will be also procedure-driven control. They will be done for holding the store.

2.7 Boundary Conditions

2.7.1 Configuration

For each UI component, they will be created when user renders the page, and when user left the page, they will be destroyed. This will be done thanks to React. Moreover, some components can be archived in order to use them later.

2.7.2 Start-up and shutdown

The Erasmus Application will not require any installation on any of the user's side since the application in run on Bilkent Servers; this brings ease of use to the application; with the opening of application dates, all the students registered in the SRS can access the website. When the application portal opens, the students can apply to various universities according to their courses; otherwise, only can see login, details, about pages.

The Erasmus application works as a complete system, all changes are made step-by-step. If a conflict arises regarding any of the steps filled out by the student, then the whole process in put on hold until the issue has been resolved. Until the finalization of the whole application process, it can put into hold multiple times and at times can also be declined.

2.7.3 Exception Handling

In case of user error, the system will display an error message to user so that the user can correct the input.

In case of a network error, the system will save the state so that it can recover when the network comes back. Also, it will show a meaningful error message to the user.

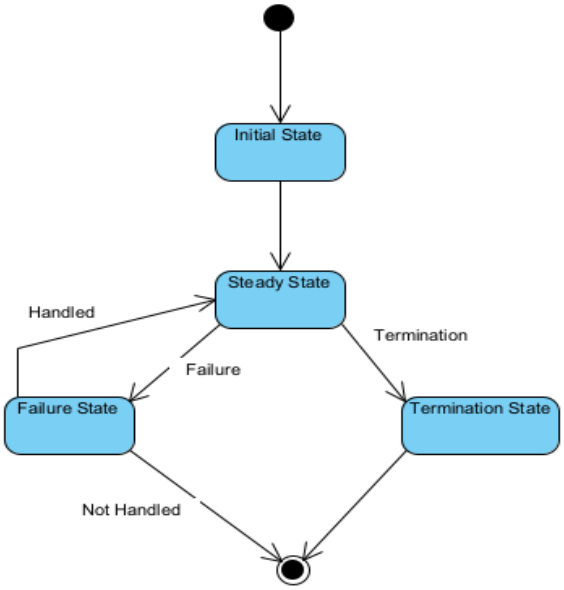

Figure 2.7.3.1: State diagram of the Sysem

2.8 Object Design Trade-offs

Functionality versus Usability: Bilkent Erasmus-Exchange System is designed to meet the needs of the students who consider about Erasmus-Exchange process and the needs of the officials. Therefore, the application should be functional and well organized as possible even if this trade-off will lead to a decrease in usability.

Security versus Performance: The response time and other topics related to the performance criterion are very important points of the application. However, the security part of the application can cause a decrease in performance because of encryption and decryption algorithms. Because security issues are more important than performance issues, the application selects higher security and as a result of this selection, the application will work with lower performance.

2.9 Final Object Design

The figure below is our final object design diagram.

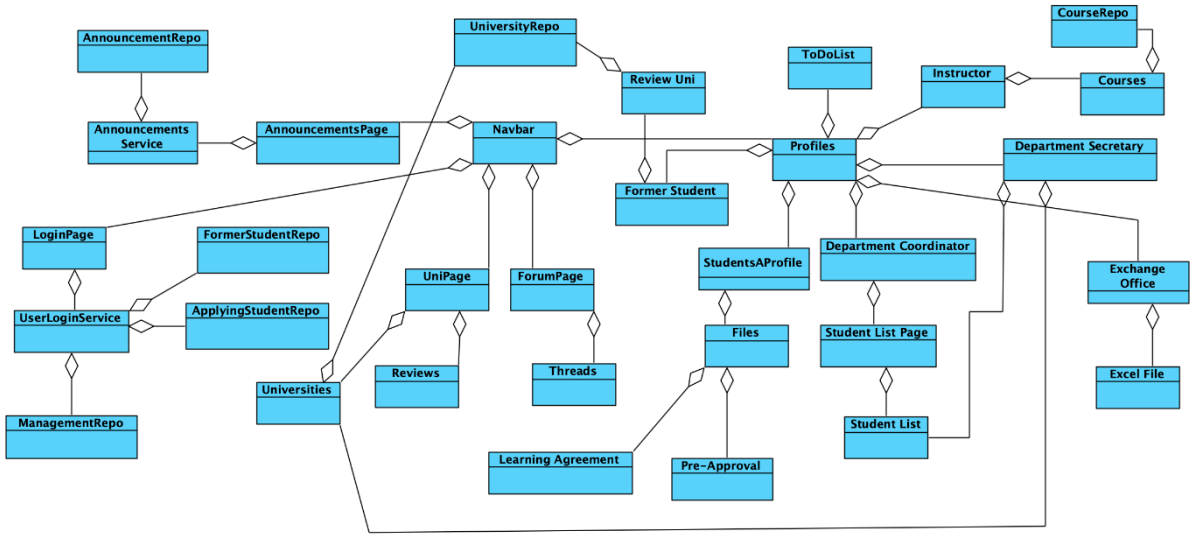

Figure 2.9.1: Final Object Design

3 Glossary

Polyfill: A polyfill is a piece of code (usually JavaScript on the Web) used to provide modern functionality on older browsers that do not natively support it.