Bilkent University

Department of Computer Engineering

Erasmus Registration Management System

Project short-name: ERSMS

Design Report

Kutay Tire - 22001787

Atak Talay Yücel - 21901636

Yiğit Yalın - 22002178

Borga Haktan Bilen - 22002733

Berk Çakar - 22003021

Instructor: Eray Tüzün
Teaching Assistant(s): Muhammad Umair Ahmed, İdil Hanhan, Emre Sülün, Mert Kara

Iteration 1

November 28, 2022

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Object-Oriented Software Engineering course CS319

Contents

Design Report

Project short-name: ERSMS

1 Introduction

1.1 Purpose of the System

ERSMS is a web-based application that aims to facilitate the jobs of students, coordinators and other stakeholders during the Erasmus management process. To this end, ERSMS intends to remove all the paperwork currently done and replace it with digitalized documents. Furthermore, the application automates processes like student placement and creates To-Do lists for both parties in order to save time for both students and exchange coordinators. Finally, necessary information about previously equivalent courses in host schools will be displayed to students in order to ensure a smoother process for them.

1.2 Design Goals

The design goals for the application are determined with the help of non-functional requirements in the analysis report. ERSMS aims to maximize usability and performance to save as much as time and effort for its customers. Also, the system should be reliable and secure as it holds a lot of personal information about students that has to be protected from any attacks or crashes.

1.2.1 Usability

The ERSMS team would like to offer an application that makes the exchange program application process easy, even for those unfamiliar with the procedure. For this reason, we provide features such as to-do lists and previous form decision query panels to achieve an overall intuitive experience for both academicians and students. Moreover, ERSMS has a not-crowded interface with self-explanatory navigation components. Finally, our development team is not in favor of adding new functionality without making sure of the usability of existing features.

1.2.2 Performance

Since ERSMS carries out a sensitive procedure, ERSMS should give the best performance even under a heavy workload. Therefore, we should consider taking some measurements on our side. In this context, we benefited greatly from the async calls in our backend, especially when querying data from the database. Similarly, in the frontend, most of the API calls to the backend are done asynchronously. Also, when we query data from the database in our backend, our aim is to limit the number of transactions as much as possible to prevent bottlenecks. In other words, when implementing the repository design pattern, we have taken care to ensure that methods can retrieve data from the database with only one request most of the time. On the other side, we did some analysis to pick the proper infrastructure setup from AWS in order to ensure the application’s performance.

1.2.3 Reliability

Because of the exchange application process continues for an academic year for a particular student, ERSMS has to have a significant level of reliability to cover at least that time duration. In that sense, we first focused on ensuring the safety of the stored data in case of any system crash. For preventing data loss in such a situation, we set up periodic backups and RAID configurations in our AWS instance. Apart from that, we do not want any downtime caused by our fault. Therefore, we paid extra attention to error handling in our controllers such that users cannot force the application to crash.

1.2.4 Security

ERSMS stores students' relatively private information, such as exchange school preference lists, exchange scores, and CGPA. In order to protect students' privacy in that manner, only authorized actors (such as admins, and exchange coordinators) can display this information in students' profiles. In addition to that, ERSMS only accepts sign-ups with Bilkent University emails to prevent outside access. Lastly, ERSMS utilizes the BCrypt hashing algorithm for storing passwords to improve user account security.

1.3 Top Two Design Goals

For the implementation of ERSMS, usability and reliability were chosen as top two design goals.

Because stakeholders want to eliminate as much paperwork as possible, ERSMS should perform this task in a usable way. For example, if we digitize the form approval process in a complex way, it would be pointless because perhaps it is easier to fill out and sign a simple piece of paper than to learn how to use a complex web application. However, if ERSMS guides and eases the procedures that make up the exchange application process to stakeholders and users in an intuitive and usable way, academics and students will adapt to the system within no time.

For the reliability goal, as stated before, since the exchange application process is a long and sensitive process, ERSMS should be up at any moment and the data it contains should not be lost. For example, in a possible data loss, students will have to fill in the submitted forms again. On the other hand, exchange coordinators will lose the follow-up of the placement process and the CTE forms they have filled, if any. These are just a few examples and data loss is distressing for users anyway.

2 High-level Software Architecture

2.1 Subsystem Decomposition

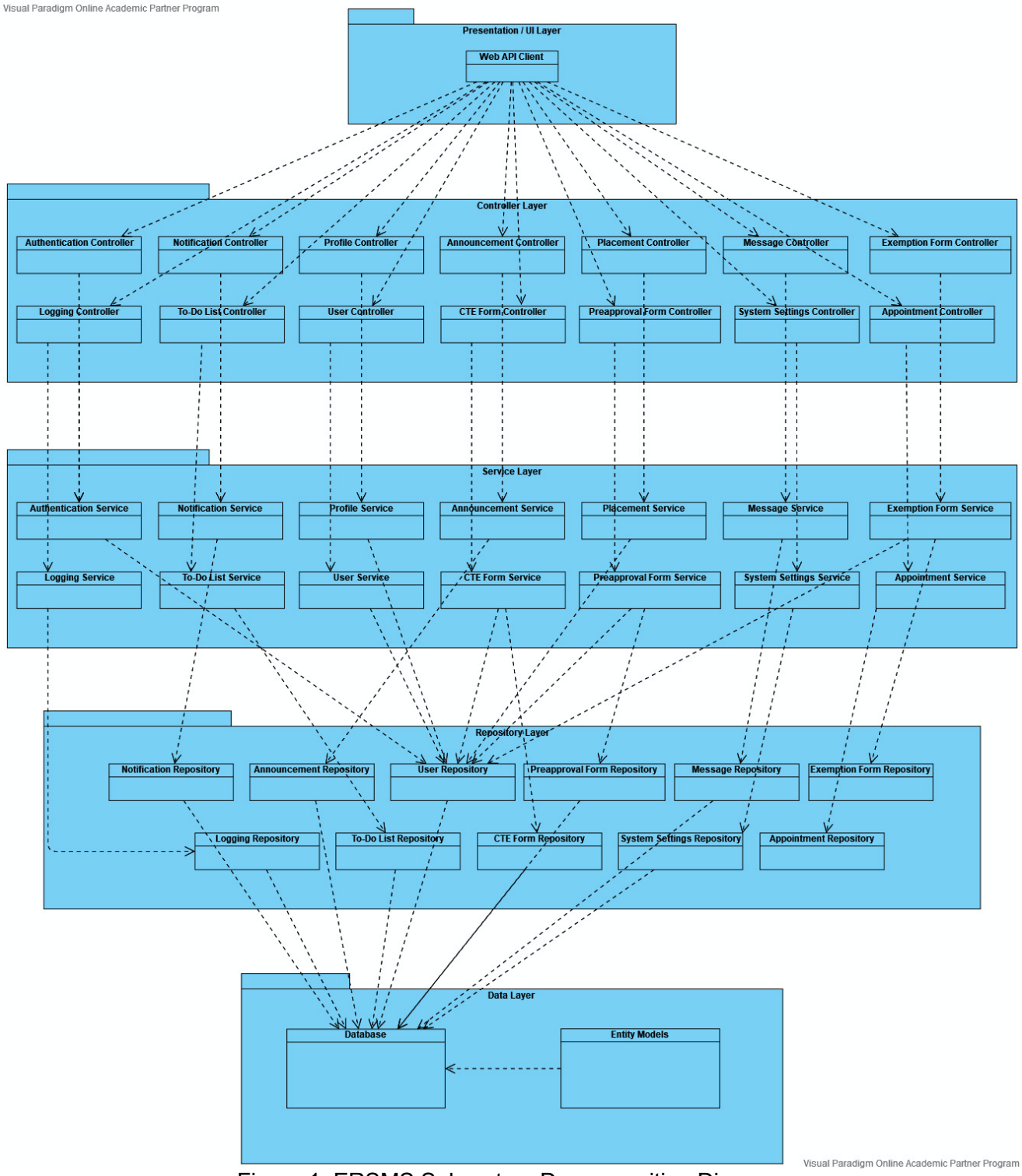

Figure 1: ERSMS Subsystem Decomposition Diagram

Since the project uses Angular for its frontend framework, the presentation layer only contains a web API client, which is responsible for all API calls. The controller layer of the backend is responsible for handling these API requests. These requests are generated by the user's interactions with Angular (i.e., frontend/client components). The controller layer calls the service layer, which is responsible for the business logic (all logical operations that are needed

to execute the specified process). The service layer reaches the repository layer. The repository layer is responsible for the communication between the database and the service layer (which, in essence, is the rest of the code). Finally, the data layer consists of entity models, and these models (entity objects) represent table entries in the database. These models help us to utilize the code first pattern and ORM (Object Relational Mapping) mechanism using the power of the EF library (Entity Framework) included in the .NET framework.

2.2 Deployment Diagram

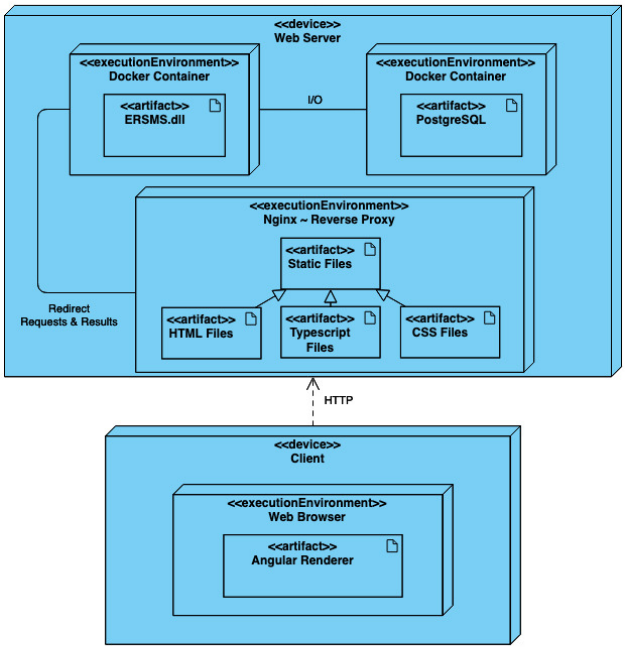

Figure 2: ERSMS Deployment Diagram

ERSMS is hosted on a web server, and the database server of ERSMS is also located on the same server to minimize infrastructure costs. Anyone with a device that can run JavaScript ES5 or newer on its web browser can interact with the ERSMS. These interactions (requests) are forwarded to the ERSMS itself via a reverse proxy solution. Likewise, ERSMS' responses to users' HTTP requests are delivered back to users with the exact reverse proxy mechanism. Moreover, the reverse proxy also serves static files to users, which are required for the client-side Angular Renderer to work.

2.3 Hardware/Software Mapping

ERSMS is a small to medium-scaled web application; therefore, it does not require excessive hardware infrastructure. However, it still requires at least one server to host the backend of the ERSMS, and users must have a device with browser access/internet connection to use the application. As can be understood, ERSMS adopts the client-server model and is optimized for modern browsers, which support at least HTML5, CSS3, and ES5, on computers and mobile phones.

Since ERSMS depends on the usage of Docker, the host server has to support containerization and virtualization. Hence, the server should be powerful in terms of CPU and RAM. Moreover, ERSMS is only officially supported to run on a GNU/Linux environment, which can be another hardware specification constraint. Lastly, we should consider that the PostgreSQL server runs on the same server as ERSMS. Hence, considering the given requirements, we can pick an AWS EC2 instance with the following specifications:

t4g.xlarge:

- 4 VCPUs (AWS Graviton2 64-Core ARMv8 2.5 GHz)
- 16 GB RAM
- 1 TB HDD (For storing database content, uploaded syllabuses, logs, and more.)

This setup costs approximately $90 a month and is expected to handle up to 100 concurrent users. However, thanks to AWS, depending on the possible new requirements which arise in the production phase, we can scale up or down the infrastructure (i.e., We may need more storage space but less CPU power).

2.4 Persistent Data Management

First of all, regardless in which mode ERSMS is initialized, ERSMS uses a relational database, since this kind reflects the OOP behavior and operations (CRUD) best. In the development environment of ERSMS, SQLite is used as the database solution. The main reason for this decision is to benefit from conventional file system access. Since SQLite databases are held in a single file (i.e., ERSMS.db), it does not require a running database server like MySQL or PostgreSQL. This way, we can track the SQLite database file via Git without needing a remote server. It is crucial because only some team members have access to local database servers, and we need more AWS credits to host a remote database server for development purposes. Also, sharing a single database file ensures that the same data set is available for every team member. However, when started in production mode, ERSMS uses a PostgreSQL server on AWS since SQLite is only good when handling low to medium-traffic HTTP requests. PostgreSQL is superior to other relational database providers in many aspects. First of all, PostgreSQL supports more non-primitive types to store natively, such as JSON, UUID, TIMESTAMP, and more. In other database providers, usually, only primitive types are available, and whenever ERSMS needs to store a non-primitive type value, the database abstraction layer of ERSMS (namely Entity Framework) converts them into a byte array which results in performance loss. Also, PostgreSQL handles multiple simultaneous database access better (in terms of concurrency). Lastly, all domain-related model classes (entities) are persisted to the database.

2.5 Access Control and Security

Security aspect of the application is mostly handled with regular authorization and authentication control flow. These two mechanisms are provided by the .NET's Identity framework, integrated with the Entity Framework. Specific API endpoints are only reachable by those users who have the required role with the adequate authorization level. Session control of the user client is provided and managed by the browser cookies, this allows the application to hold the authorization and authentication status persisted on the user side. Because every endpoint has a related role, the application doesn't contain any functionality for the guest users. Finally, ERSMS utilizes a BCrypt algorithm for hashing a user's password for storing it securely.

	Authentication	Notification	Profile	Announcement	Placement	Message	ExemptionForm	Logging	ToDoList	User	CTEForm	PreApprovalForm	SystemSettings	Appointment
DomainUser	login() forgetPassword()	-	-	-	-	-	-	-	-	-	-	-	-	-
Student	login() forgetPassword()	viewNotification() getNotification() deleteNotification()	viewProfile() editProfile()	viewAnnouncement() getAllAnnouncements()	-	sendMessage() deleteMessage() viewMessage() getMessage()	getForm() createForm() deleteForm() cancelForm() editForm() submitForm()	filterBy() listCTEForms() listExemptionForms() listPreApprovalForms()	viewToDoList() getItem() getAllItemst() deleteItem()	viewUser() editUser()	getForm()	getForm() createForm() cancelForm() submitForm() editForm() deleteForm()	changeSettings()	arrangeAppointment() editAppointment() getAppointment() cancelAppointment() viewAppointmentSchedule()
CourseCoordinatorInstructor	login() forgetPassword()	viewNotification() getNotification() deleteNotification()	viewProfile() searchProfile() getProfile() editProfile()	viewAnnouncement() getAllAnnouncements() deleteAnnouncement()	-	sendMessage() deleteMessage() viewMessage() getMessage() deleteAllMessages()	getForm() getAllForms() approveForm() rejectForm() deleteForm()	filterBy() listCTEForms() listExemptionForms() listPreApprovalForms()	-	viewUser() editUser() getUser() searchUser() deleteUser()	-	-	changeSettings()	-
Exchange Coordinator	login() forgetPassword()	viewNotification() getNotification() deleteNotification()	viewProfile() searchProfile() getProfile() editProfile()	viewAnnouncement() getAllAnnouncements() deleteAnnouncement() createAnnouncement()	placeAutomatically() addToWaitingList() removeFromWaitingList() cancelPlacement() viewWaitingList() viewPlacements() viewScoreTable() getScoreTable()	sendMessage() deleteMessage() viewMessage() getMessage() deleteAllMessages()	getForm() getAllForms() approveForm() rejectForm() deleteForm()	filterBy() listCTEForms() listExemptionForms() listPreApprovalForms() deletePreApprovalFormLogs() deleteCTEFormLogs() deleteExemptionFormLogs()	viewToDoList() getItem() getAllItemst() deleteItem() addItem() createToDoList() editToDoList()	viewUser() editUser() getUser() searchUser() deleteUser()	createForm() getForm() getAllForms() deleteForm() cancelForm() submitForm() editForm()	approveForm() rejectForm() getForm() createForm() cancelForm() submitForm() editForm() deleteForm()	changeSettings()	arrangeAppointment() editAppointment() getAppointment() cancelAppointment() viewAppointmentSchedule()
DeanDeparmentChair	login() forgetPassword()	viewNotification() getNotification() deleteNotification()	viewProfile() searchProfile() getProfile() editProfile()	viewAnnouncement() getAllAnnouncements() deleteAnnouncement() createAnnouncement()	-	sendMessage() deleteMessage() viewMessage() getMessage() deleteAllMessages()	getForm() getAllForms() approveForm() rejectForm() deleteForm()	filterBy() listCTEForms() listExemptionForms() listPreApprovalForms() deletePreApprovalFormLogs() deleteCTEFormLogs() deleteExemptionFormLogs()	-	viewUser() editUser() getUser() searchUser() deleteUser()	createForm() approveForm() rejectForm() getForm() getAllForms() deleteForm() cancelForm() submitForm() editForm()	approveForm() rejectForm() getForm() createForm() cancelForm() submitForm() editForm() deleteForm()	changeSettings()	-
Admin	login() forgetPassword()	viewNotification() getNotification() deleteNotification()	viewProfile() searchProfile() getProfile() editProfile()	viewAnnouncement() getAllAnnouncements() deleteAnnouncement() createAnnouncement()	-	sendMessage() deleteMessage() viewMessage() getMessage()	-	-	-	viewUser() editUser() getUser() searchUser() deleteUser()	-	-	changeSettings()	-
OISEP	login() forgetPassword()	viewNotification() getNotification() deleteNotification()	-	viewAnnouncement() getAllAnnouncements()	uploadScoreTable() viewScoreTable() deleteScoreTable() getScoreTable()	sendMessage() deleteMessage() viewMessage() getMessage()	-	-	-	-	-	-	changeSettings()	-

Table 1: ERSMS Actor-Object Access Matrix

2.6 Boundary Conditions

2.6.1 Initialization

The initialization process is triggered by Docker's "docker-compose up" command. As can be seen in the "Deployment Diagram", ERSMS is associated with its database and reverse proxy. Therefore, before ERSMS is initialized, it checks if both subsystems are started and working correctly. Then, ERSMS automatically migrates all database schemas from the entity models according to the code first paradigm if it has not already done so. After that, .NET injects all external packages and internal services (i.e., AuthenticationService, PlacementService, and more) during the initialization. After the procedure mentioned above is completed successfully, ERSMS becomes functional.

2.6.2. Termination

Application is terminated using Docker's "docker-compose down" command, which triggers the .NET's graceful exit. There are no mandatory steps to be taken other than gracefully terminating the Docker containers using docker-compose. The application's subsystems work together; thus, if a subsystem is terminated gracefully, other remaining subsystems are also terminated gracefully without causing any errors or data loss (data saved in the database will be protected during the termination state).

2.6.3. Failure

In an event of failure, the application doesn't terminate, rather it logs the exception (if it is specified by the programmer). Error handling mechanism is designed to catch exceptions in a timely manner and in appropriate places (in order to prevent data loss or data corruption). Additionally, .NET framework is mindful about the exceptions, it can catch and report exceptions that can be easily ignored (mistakenly) by the programmer.

3 Low-level Design

3.1 Object Design Trade-offs

- **Maintainability** versus **Backward Compatibility:**
  ERSMS will be supported by all the browsers that are compatible with ES5. This will restrict the usage of new features that are published in later ECMAScript versions. Thus, the application will become harder to maintain the project to support older browsers.

- **Maintainability** versus **Performance:**
  ERSMS will have an object-oriented design to enhance the readability and writability of the code. Though this will improve the maintainability of the project, it might also lead to a decrease in the performance.

- **Reliability** versus **Cost:**
  Since ERSMS stores sensitive data, it will hold multiple copies of the data to prevent data loss. This will increase the storage needs and the cost of the project.

- **Reliability** versus **Performance:**
  Since ERSMS processes sensitive information, it will be implemented in a way that minimizes the risk of crashes. This will increase the number of conditional statements and exception handling statements, and thus, reduce the performance.

- **Usability** versus **Functionality:**
  ERSMS will provide only the necessary functionality to facilitate the jobs of students, coordinators and other stakeholders during the Erasmus management process. There will not be any redundant functionality that might confuse the user to increase the usability.

- **Usability** versus **Security:**
  Since ERSMS stores personal information of the users, it is necessary to ensure the security. This necessity will add extra steps such as email authentication, and thus, reduce the usability.

3.2 Final Object Design

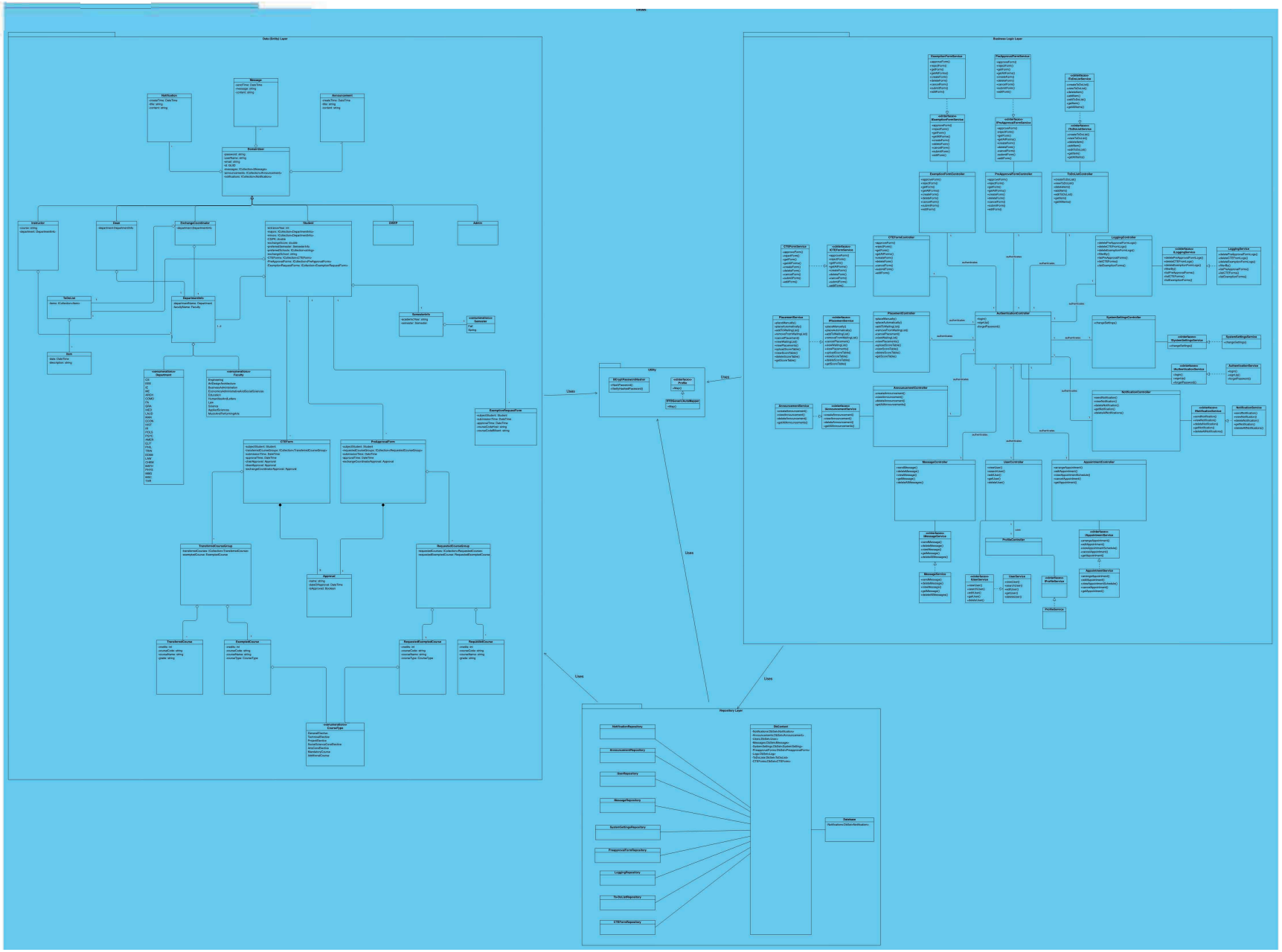

Figure 3: ERSMS Final Object Design Diagram

3.3 Packages

Following list is the namespaces (i.e., packages) in the project with small descriptions:

- ERSMS.Entities: Contains the entity models that are mapped to the tables of the database. Thus, these entities are the main units of data in the application.
- ERSMS.Entities.Enums: Contains the enum definitions that are used in the entity models.
- ERSMS.Controllers: Contains the API endpoints, which are the exposed part of the backend to the internet. External communication with the application handled here.
- ERSMS.Data: This namespace has the database context file which contains the configurations for the database. This file initializes the Entity Framework library of the .NET framework.
- ERSMS.DataAnnotations: Contains the special annotations and their rules. These special annotations are, specifically, used in the controller layer to enforce a set of rules on an object.
- ERSMS.Services: This namespace contains the layer which is responsible for fundamental business logic. This layer also acts like a bridge between the controller (API endpoints) layer and the repository layer.
- ERSMS.Interfaces: Contains the interfaces (especially service layer interfaces and repository layer interfaces) that are used in service, controller and repository layers.
- ERSMS.Utility: Contains the classes that consist of utility functions. These functions are used among many different namespaces.
- ERSMS.Utility.Enums: Contains the enum definitions that are used in many different namespaces through the code.
- ERSMS.DTOs: Contains objects that are used to exchange data during external communications (between front-end and controller layer). These objects are the representations of entity models, which are their modified versions (modified with the consideration of data security).

3.3.1 External Packages

External packages (libraries) used in the application.

- Microsoft.AspNetCore: AspNet Core library contains the framework (.NET) that is used throughout the application.
- Microsoft.EntityFrameworkCore: This library is used for implementing the ORM (object relational mapping) pattern/mechanism and code first paradigm for the persistent data. This library enables us to use the database tables as a C# object inside the code.
- Microsoft.AspNetCore.Identity.EntityFrameworkCore: This library is used for the authentication and authorization of users using user roles. This authentication and authorization mechanism is also integrated with the Entity Framework.
- BCrypt.Net-Next: This library contains an encryption algorithm that is used for hashing the user passwords.
- Microsoft.EntityFrameworkCore.Sqlite: This package contains a driver that facilitates the connection between SQLite database and Entity Framework.

3.4 Class Diagrams

3.4.1 Data Access Layer

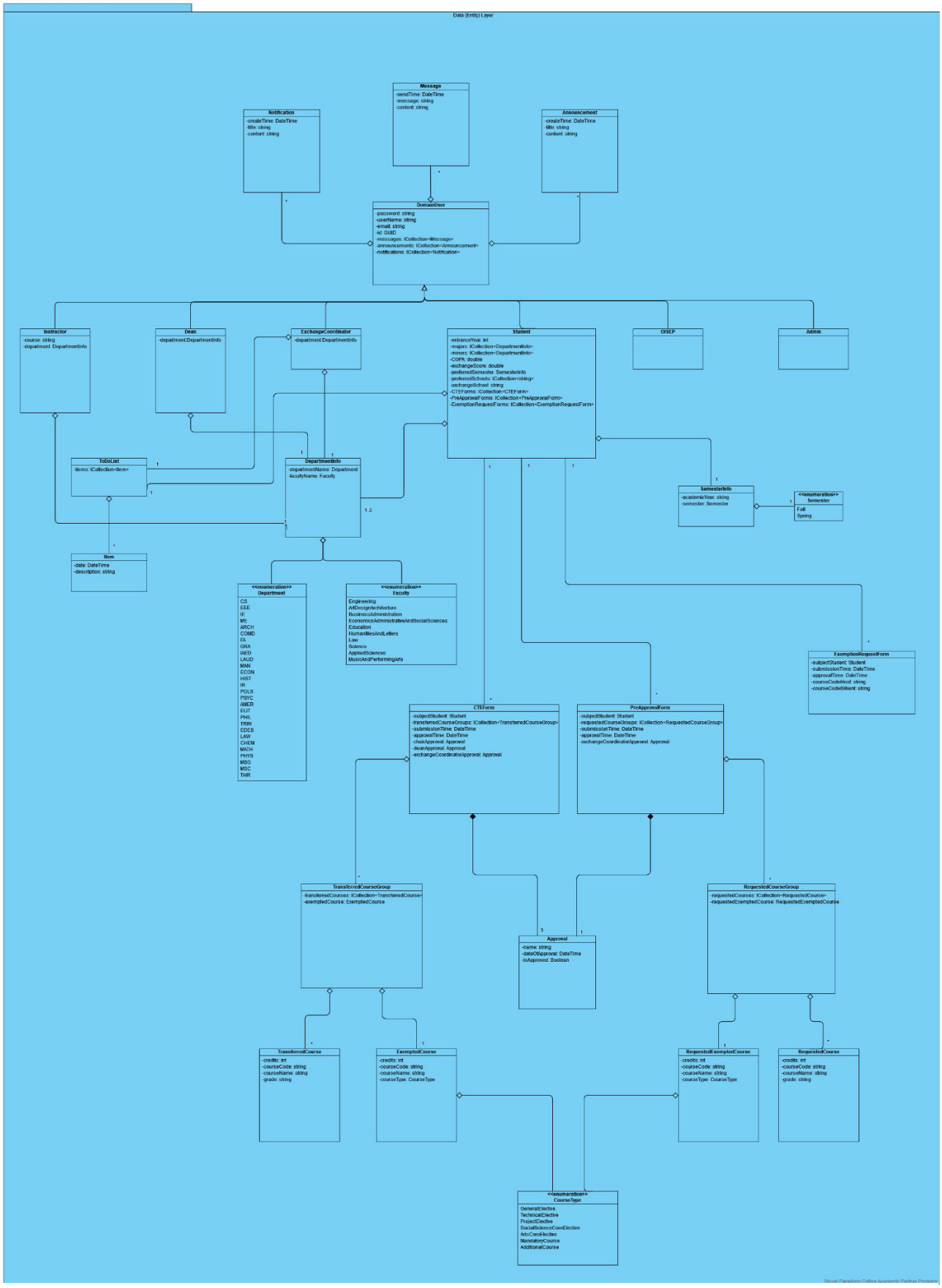

Figure 4: ERSMS Data (Entity) Layer Diagram

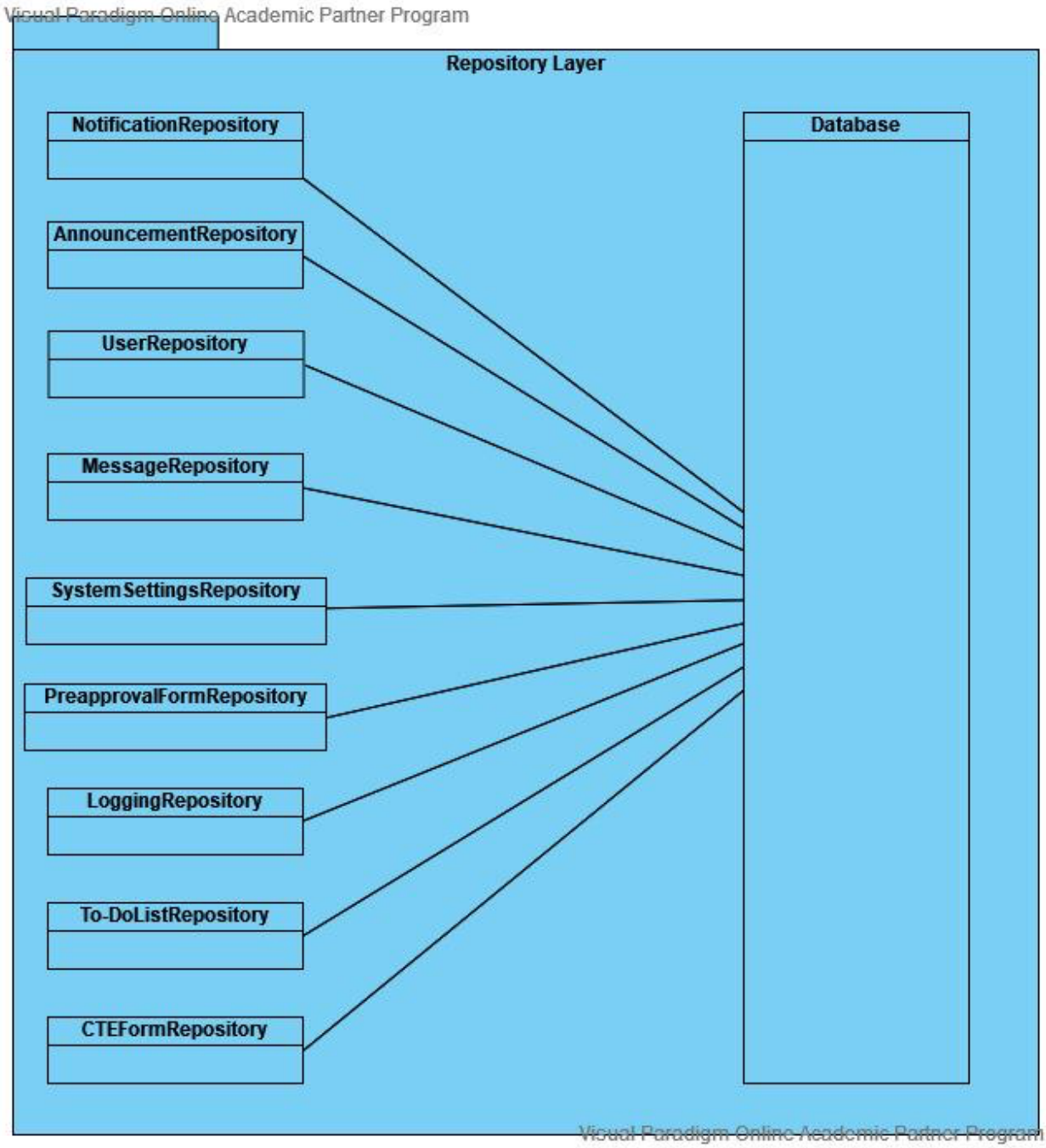

Figure 5: ERSMS Repository Layer Diagram

Data access layer consists of two sub-layers which are the repository layer and entity (model) layer. Entity layer consists of model classes, which are the persistent data. The database migration schematics are derived from these models.

3.4.2 Business Logic Layer

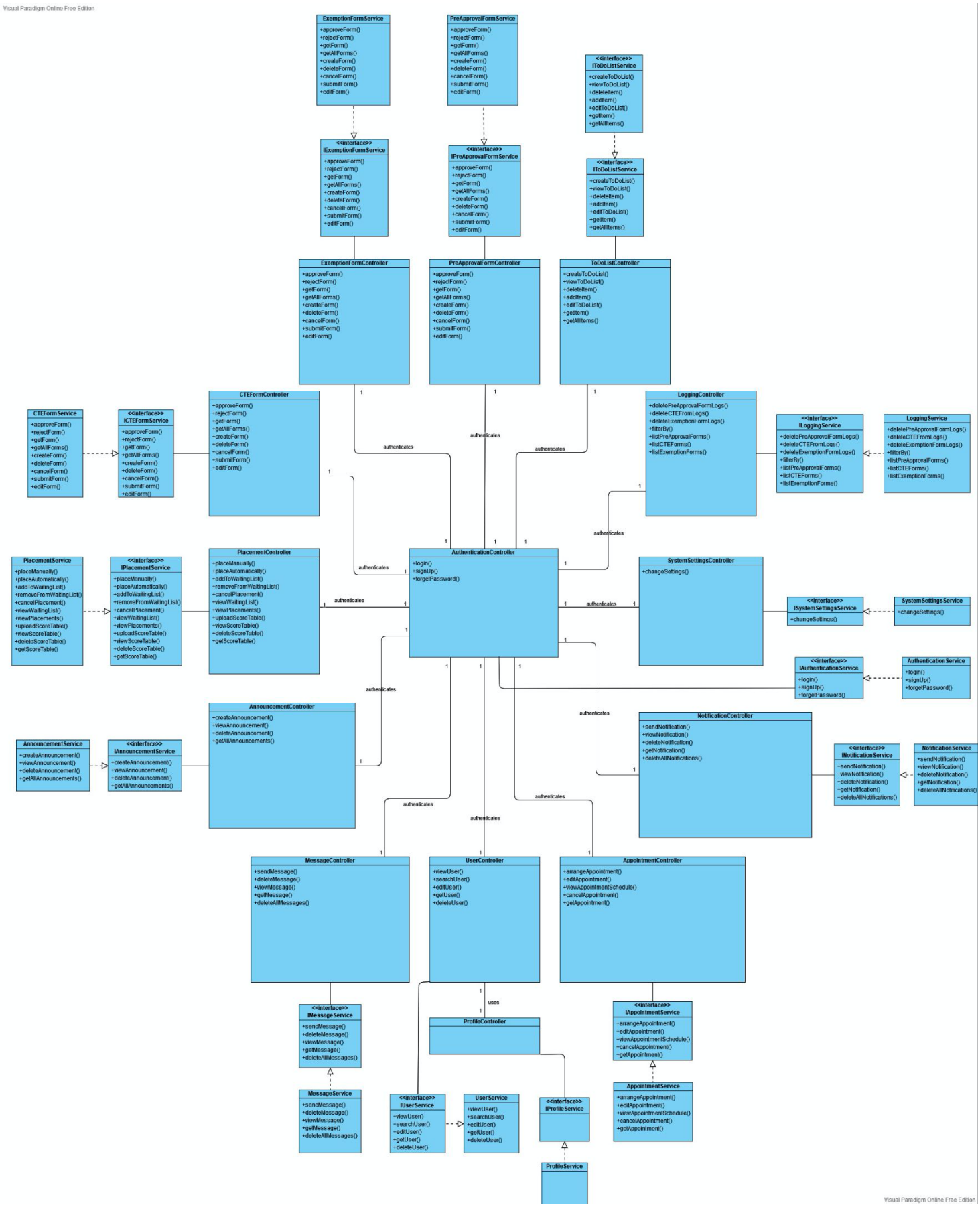

Figure 6: ERSMS Business Logic Layer Diagram

Business logic layer diagram shows the controller layer and service layer in a merged manner. The web API of the application consists of controllers (i.e., controller layer), which controller layer uses service layer to perform business logic, mostly related with application domain.

3.4.3 User Interface Layer

The user interface layer is a boundary object that manages the interaction between the system and user.

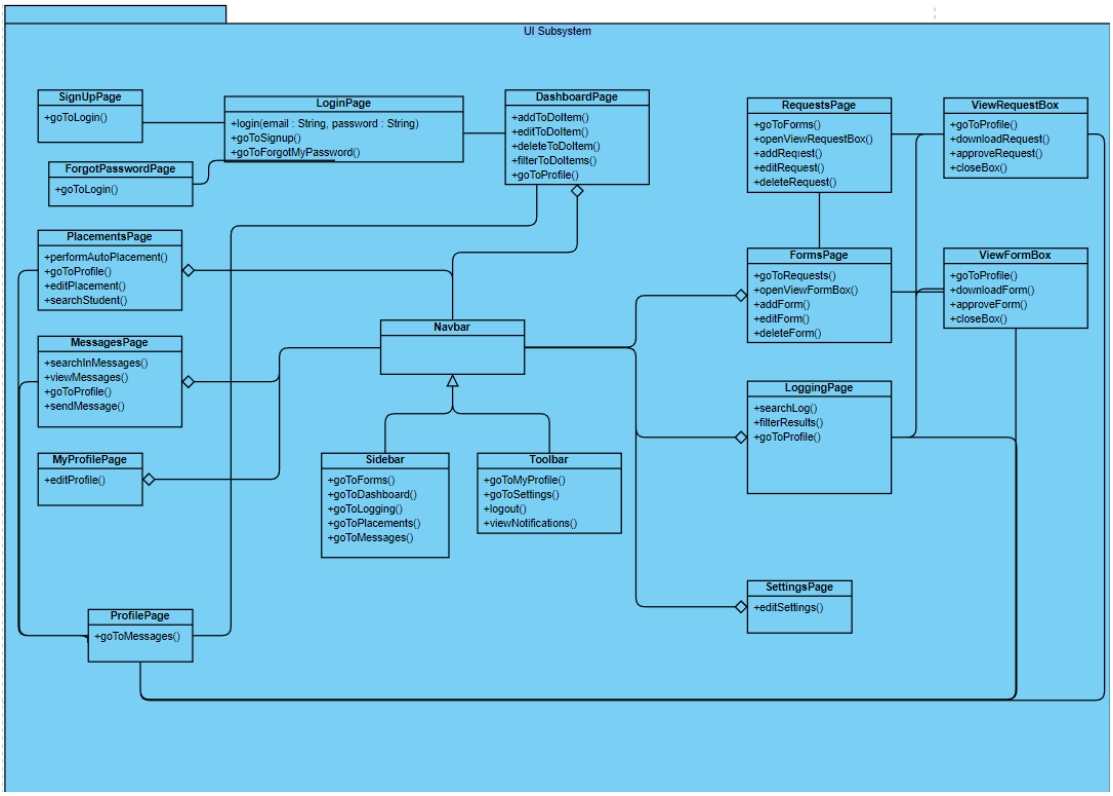

Figure 7: ERSMS User Interface Layer Diagram

4 Glossary

- **API:** Application Programming Interface
- **AWS:** Amazon Web Services
- **RAID:** Redundant Array of Inexpensive Disks. It is a data storage technology that combines physical disks into several logical units for the purposes of data safety.
- **BCrypt:** A popular hashing algorithm used for hashing and storing passwords.
- **Angular:** A popular JavaScript frontend development framework.
- **ORM:** Object Relational Mapping
- **EF (or EF Core):** Entity Framework. A widely used .NET extension for automated object relational mapping.
- **.NET:** Microsoft's application development framework.
- **ES5:** ECMAScript 5. It is the JavaScript standard implemented in modern web browsers.
- **HTTP:** Hypertext Transfer Protocol
- **HTML:** Hyper Text Markup Language. HTML files form web pages.
- **CSS:** Cascading Style Sheet. CSS is used when customizing the look of the HTML web pages.
- **CRUD:** Create, Retrieve, Update, and Delete
- **Docker:** A virtualization solution to run software in "containers" which are isolated spaces from the host machine.
- **Nginx:** Nginx is a web server which can also be used as a reverse proxy solution.