Bilkent University

Department of Computer Engineering

Erasmus Registration Management System

Project short-name: ERSMS (Group Number: 1E)

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 2

December 11, 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

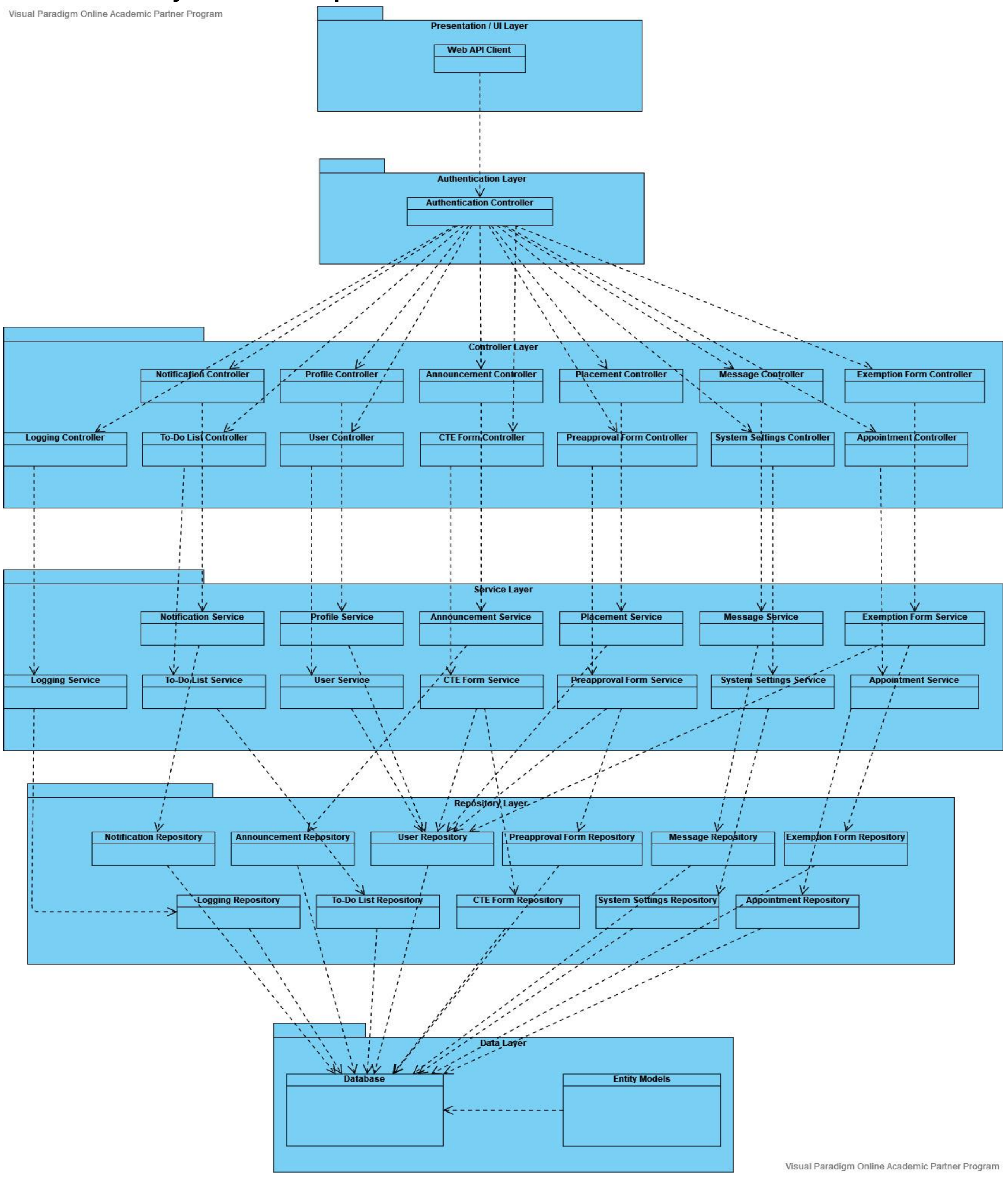
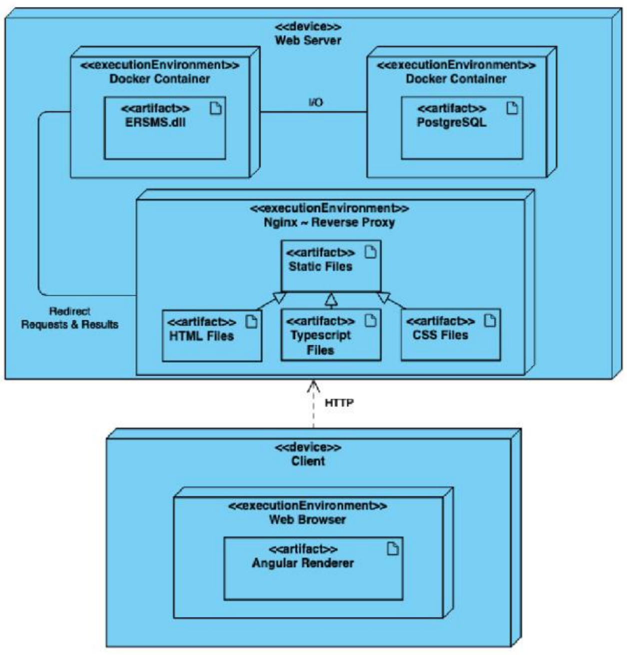


Figure 1: ERSMS Subsystem Decomposition Diagram

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). Authentication layer controls the authorization of the request after that 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



<<device>> Web Server
<<executionEnvironment>> Docker Container
<<artifact>> ERSMS.dll
I/O
<<executionEnvironment>> Docker Container
<<artifact>> PostgreSQL
<<executionEnvironment>> Nginx ~ Reverse Proxy
<<artifact>> Static Files
Redirect Requests & Results
<<artifact>> HTML Files
<<artifact>> Typescript Files
<<artifact>> CSS Files
HTTP
<<device>> Client
<<executionEnvironment>> Web Browser
<<artifact>> Angular Renderer

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 clientside 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.)

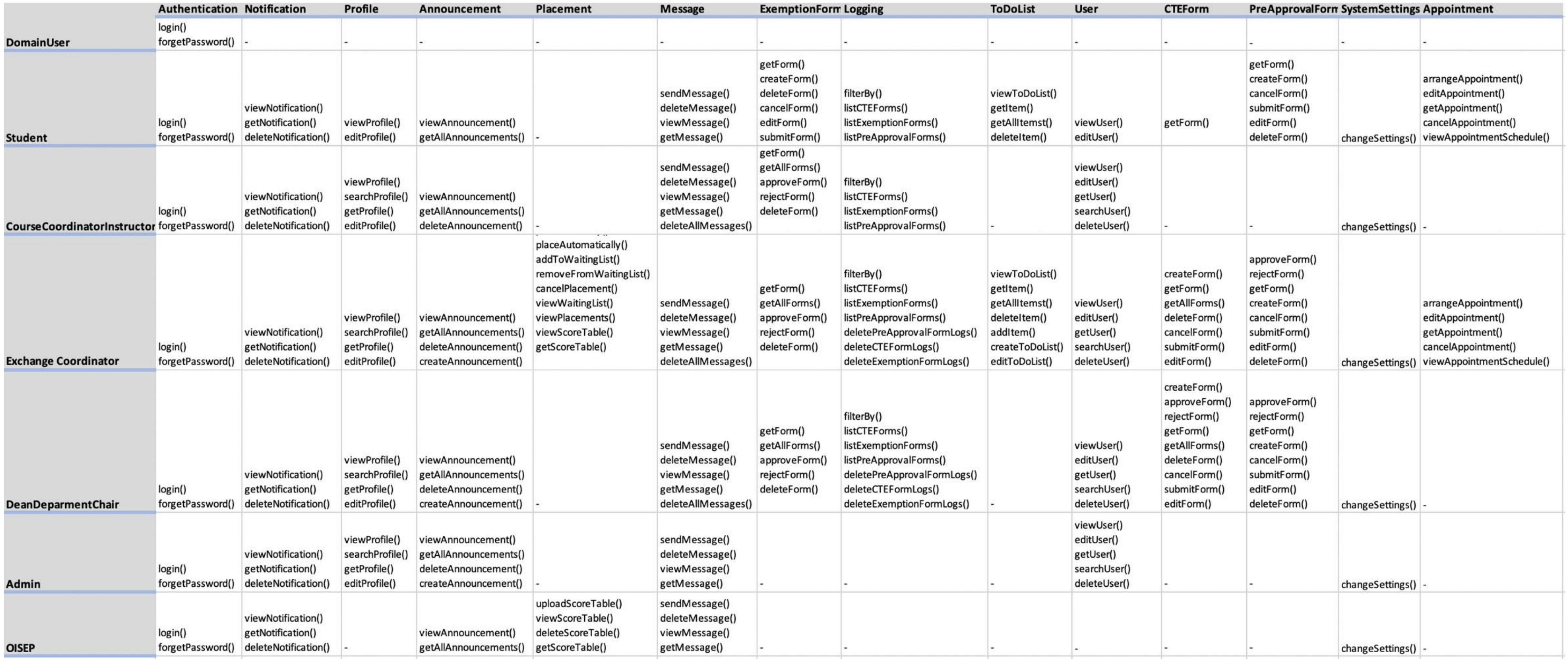
This setup costs approximately $90 a month and is expected to handle up to 100 concurrent users. However, thanks to AWS, depending on the possible new requirements which arise in the production phase, we can scale up or down the infrastructure (i.e., We may need more storage space but less CPU power).

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). However, as can be seen, using a relational database is a suitable solution for persisting the data from all domain-related model classes (entities), such as users, forms, messages, and more. Still, it is needed to store BLOB data such as syllabus PDFs for course exemption requests or Excel files for exchange score tables. Since persisting this kind of data in a relational database is not convenient (even if storing files as byte arrays is possible in PostgreSQL, it reduces the database performance significantly), ERSMS stores these files in the local file system of the host server. For this, the application occupies a folder in the home directory of the server (i.e., /home/ERSMS) and stores the files in that folder according to their types (i.e., Excel files for exchange score tables will be stored in /home/ERSMS/ScoreTables). For referring back to these files, for every uploaded file, ERSMS persists the metadata (i.e., the file name) of the file in 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 (For high quality version refer to https://drive.google.com/file/d/1838X7UFb2rz05tcjgy36DjcBMFVFUW3y/view)

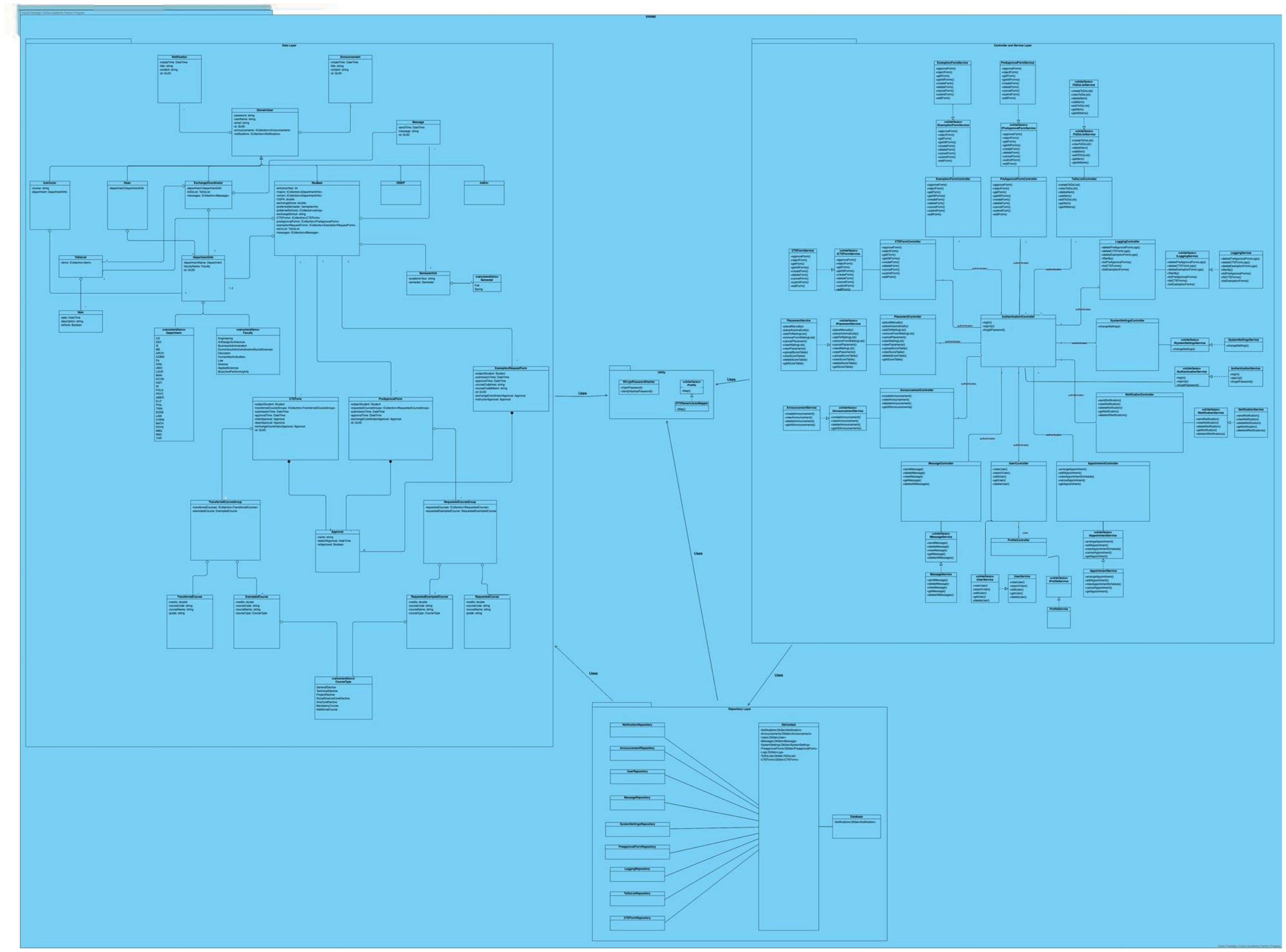


Figure 3: ERSMS Final Object Design Diagram

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 Layer (high resolution reference)

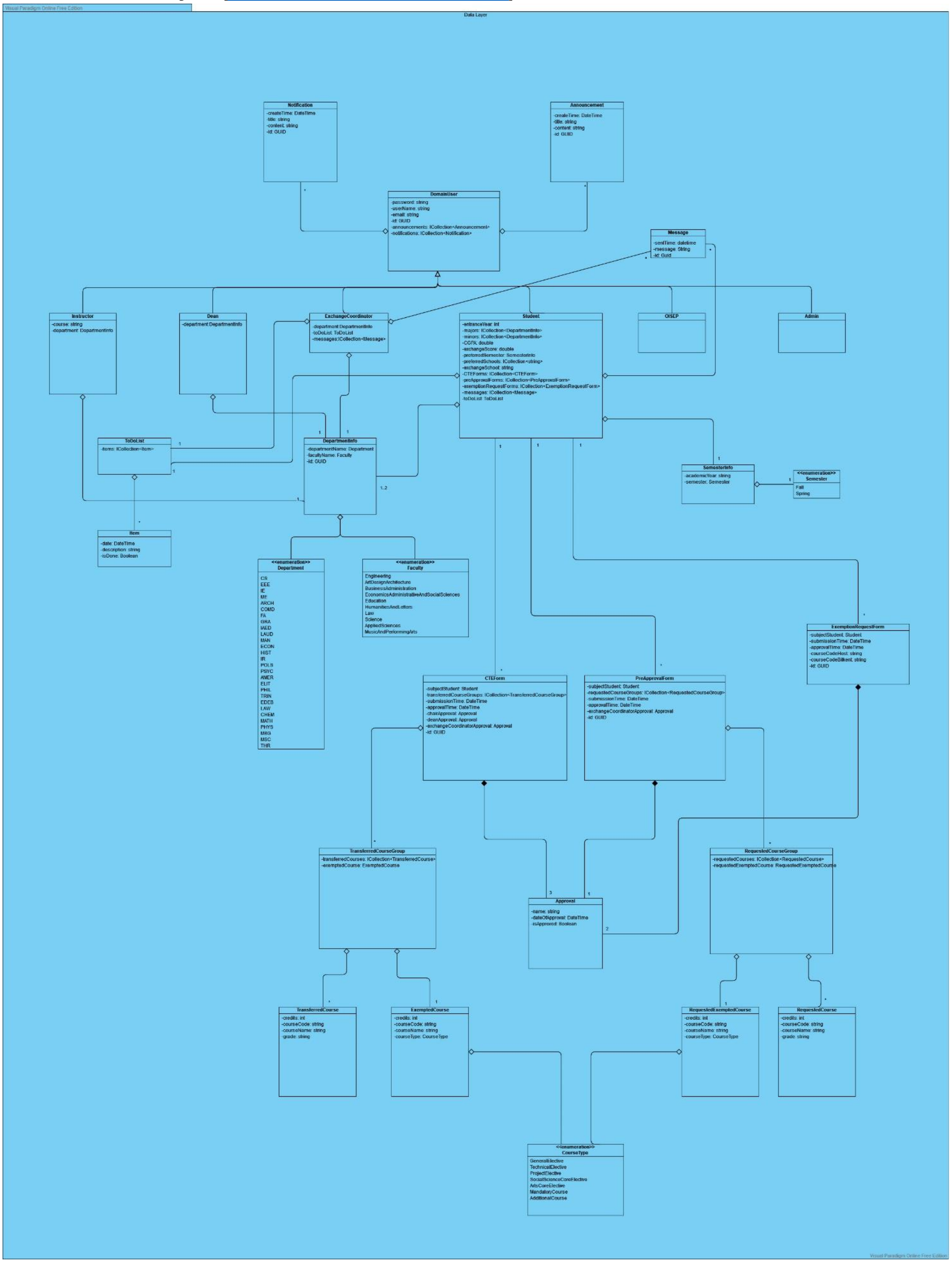


Figure 4: ERSMS Data (Entity) Layer Diagram

Data Layer

Note: In each entity class the following operations exist unless stated otherwise:

- Getter/setter methods
- Default constructors

DomainUser

The DomainUser class is used to represent all the users in the project.

Attributes

- **private String password:** password of the user
- **private String userName:** user name of the user
- **private String email:** email address of the user
- **private GUID id:** user id used as a primary key
- **private ICollection<Announcement> announcements:** list of announcements of a user
- **private ICollection<Notification> notifications**: list of notifications of a user

Notification

The Notification class represents the notifications that will be sent to the users after an event. This event can be sending a form, approving a form, etc.

Attributes

- **private DateTime createTime:** the time when the notification is created
- **private String content:** content of the notification
- **private String title:** title of the notification
- **private GUID id:** notification id used as a primary key

Message

The Message class represents the messages that can be sent between users via the application.

Attributes

- **private DateTime sendTime:** the time when the notification is sent
- **private String message:** message that the notification contains
- **private GUID id:** notification id used as a primary key

Announcement

The Announcement class represents the announcements that can be made by the exchange coordinators in case they want to announce a significant development.

Attributes

- **private DateTime createTime:** the time when the announcement is created
- **private String content:** content of the announcement
- **private String title:** title of the announcement
- **private GUID id:** announcement id used as a primary key

Instructor
The Instructor class is used to represent the instructor users.

Attributes
- **private String course:** course of the instructor
- **private DepartmentInfo department:** department information of the instructor containing which department and faculty s/he is in.

Dean
The Dean class is used to represent the dean users.

Attributes
- **private DepartmentInfo department:** department information of the dean containing which department and faculty s/he is in.

ExchangeCoordinator
The ExchangeCoordinator class is used to represent the exchange coordinator users.

Attributes
- **private DepartmentInfo department:** department information of the exchange coordinator containing which department and faculty he/she is in.
- **private ICollection<Message> messages:** list of messages of an exchange coordinator
- **private ToDoList toDoList:** to-do list of the exchange coordinator

OISEP
The OISEP class is used to represent the OISEP users who can upload the score tablet o the system.

Admin
The Admin class is used to represent the admin user.

Student
The Student class is used to represent the student users in the project.

Attributes
- **private int entranceYear:** the year the student entered the university
- **private ICollection<DepartmentInfo> majors:** list of majors of a student. It can contain at most 2 DepartmentInfo objects as a student can have at most 2 majors.
- **private ICollection<DepartmentInfo> minors:** list of minors of a student. It can contain at most 1 DepartmentInfo object as a student can have at most 1 minor.
- **private double CGPA:** CGPA of the student
- **private double exchangeScore:** exchange score of the student
- **private SemesterInfo preferredSemester:** the semester in which the student goes to the Erasmus program
- **private ICollection<String> preferredSchools:** list of preferred schools of the student. It can contain at most 5 elements
- **private String exchangeSchool:** the school the student is accepted to
- **private ICollection<CTEForm> CTEForms:** list of CTE forms of the student

- **private ICollection<PreApprovalForm> preApprovalForms:** list of pre-approval forms of the student
- **private ICollection<ExemptionRequestForm>: exemptionRequestForms:** list of exemption request forms of the student
- **private ICollection<Message> messages:** list of messages of a student
- **private ToDoList toDoList:** to-do list of the student

DepartmentInfo
The DepartmentInfo class is used to contain the department and faculty information and it is used by Student, Dean, Instructor and ExchangeCoordinator classes.

Attributes
- **private Department departmentName:** name of the department
- **private Faculty facultyName:** name of the faculty
- **private GUID id:** DepartmentInfo id used as a primary key

Department <<enumeration>>
The list of departments in Bilkent University

- **CS**
- **EEE**
- **IE**
- **ME**
- **ARCH**
- **COMD**
- **FA**
- **GRA**
- **IAED**
- **LAUD**
- **MAN**
- **ECON**
- **HIST**
- **IR**
- **POLS**
- **PSYC**
- **AMER**
- **ELIT**
- **PHIL**
- **TRIN**
- **EDEB**
- **LAW**
- **CHEM**
- **MATH**
- **PHYS**
- **MBG**
- **MSC**
- **THR**

Faculty <<enumeration>>

The list of faculties in Bilkent University

- **Engineering**
- **ArtDesignArchitecture**
- **BusinessAdministration**
- **EconomicsAdministrativeAndSocialSciences**
- **Education**
- **HumanitiesAndLetters**
- **Law**
- **Science**
- **AppliedSciences**
- **MusicAndPerformingArts**

ToDoList
The ToDoList class represents the to-do lists of students and Exchange coordinators.

Attributes
- **private ICollection<Item> items:** list of items inside the to-do list

Item
The Item class represents the items inside a to-do list.

Attributes
- **private DateTime date:** date of when the item is added to the list.
- **private String description:** the description of the item
- **private Boolean isDone:** variable indicates whether the activity is completed or not

SemesterInfo
The SemesterInfo class represents the semester information that is used in Erasmus. Student class holds an instance of this class as students have to specify a semester for Erasmus.

Attributes
- **private String academicYear:** represents the year in which the student goes to the Erasmus program
- **private Semester semester:** represents the semester in which the student goes to the Erasmus program

Semester <<enumeration>>
The list of semesters in Bilkent University

- **Fall**
- **Spring**

ExemptionRequestForm

The ExemptionRequestForm class represents the exemption request forms sent by the students to ask for an equivalence for a course in the host university.

Attributes

- **private Student subjectStudent:** represents the student who uploaded the form
- **private DateTime submissionTime:** represents the time exemption request form is sent
- **private DateTime approvalTime:** represents the time exemption request form is approved
- **private String courseCodeHost:** represents the course code in the host university
- **private String courseCodeBilkent:** represents the course code in Bilkent university
- **private Approval exchangeCoordinatorApproval:** represents the approval for the form by the exchange coordinator.
- **private Approval instructorApproval:** represents the approval for the form by the instructor.
- **private GUID id:** Exemption Request Form id used as a primary key

CTEForm

The CTEForm class represents the CTE Form sent by the students to ask for their completed courses at the host university to be transferred to their Bilkent curriculum.

Attributes

- **private Student subjectStudent:** represents the student who uploaded the form.
- **private ICollection<TransferredCourseGroup>:** represents the group of transferred courses that contain courses taken at the host university and the courses that will be exempted at Bilkent University.
- **private DateTime submissionTime:** represents the time exemption request form is sent.
- **private DateTime approvalTime:** represents the time exemption request form is approved.
- **private Approval chairApproval:** represents the approval for the form by the chair.
- **private Approval deanApproval:** represents the approval for the form by the dean.
- **private Approval exchangeCoordinatorApproval:** represents the approval for the form by the exchange coordinator.
- **private GUID id:** CTE Form id used as a primary key

PreApprovalForm
The PreApprovalForm class represents the pre-approval form sent by the students to ask for necessary approvals of the courses that they want to take at the host university.

Attributes
- **private Student subjectStudent:** represents the student who uploaded the form.
- **private ICollection<RequestedCourseGroup>:** represents the group of requested courses that contain courses taken at the host university and the courses that will be exempted at Bilkent University.
- **private DateTime submissionTime:** represents the time exemption request form is sent.
- **private DateTime approvalTime:** represents the time exemption request form is approved.
- **private Approval exchangeCoordinatorApproval:** represents the approval for the form by the exchange coordinator.
- **private GUID id:** Pre-approval form id used as a primary key

TransferredCourseGroup
The TransfferedCourseGroup class represents the group of courses that contain courses taken at the host university and the courses that will be exempted at Bilkent University.

Attributes
- **private ICollection<TransferredCourse> transferredCourses:** represents the courses taken at the host university. In some instances, two or more courses can be counted as one course at Bilkent, hence it has a type of ICollection<TransferredCourses>.
- **private ExemptedCourse exemptedCourse:** represents the course at Bilkent University that courses taken at the host university will be counted as.

TransferredCourse
The course that is taken at the host university which will be transferred to the student's curriculum at Bilkent.

Attributes
- **private double credits:** represents the credits of the transferred course.
- **private String courseCode:** represents the code of the transferred course.
- **private String courseName:** represents the name of the transferred course.
- **private String grade:** represents the letter grade of the transferred course.

ExemptedCourse

The course that is exempted at Bilkent University.

Attributes

- **private double credits:** represents the credits of the exempted course.
- **private String courseCode:** represents the code of the exempted course.
- **private String courseName:** represents the name of the exempted course.
- **private CourseType courseType:** represents the type of the exempted course.

CourseType<<enumeration>>

The list of course types that are in Bilkent University

- **GeneralElective**
- **TechnicalElective**
- **ProjectElective**
- **SocialScienceCoreElective**
- **ArtsCoreElective**
- **MandatoryCourse**
- **AdditionalCourse**

Approval

The approval of the forms or requests.

Attributes

- **private String name:** represents the name of the person that makes the decision.
- **private DateTime dateOfApproval:** represents the time the decision.
- **private Boolean isApproved:** represents the decision made.

RequestedCourseGroup

The RequestedCourseGroup class represents the group of courses that contain requested courses at the host university and the courses that are asked to be exempted at Bilkent University.

Attributes

- **private ICollection<RequestedCourse> requestedCourses:** represents the courses that will be taken at the host university. In some instances, two or more courses can be counted as one course at Bilkent, hence it has a type of ICollection<TransferredCourses>.
- **private RequestedExemptedCourse requestedExemptedCourse:** represents the course at Bilkent University that courses taken at the host university are requested to be counted as.

RequestedCourse

The course that will be taken at the host university which is requested to be transferred to the student's curriculum at Bilkent.

Attributes

- **private double credits:** represents the credits of the requested course.
- **private String courseCode:** represents the code of the requested course.
- **private String courseName:** represents the name of the requested course.
- **private String grade:** represents the letter grade of the requested course.

RequestedExemptedCourse

The course that is requested to be exempted at Bilkent University.

Attributes

- **private double credits:** represents the credits of the exempted course.
- **private String courseCode:** represents the code of the exempted course.
- **private String courseName:** represents the name of the exempted course.
- **private CourseType courseType:** represents the type of the exempted course.

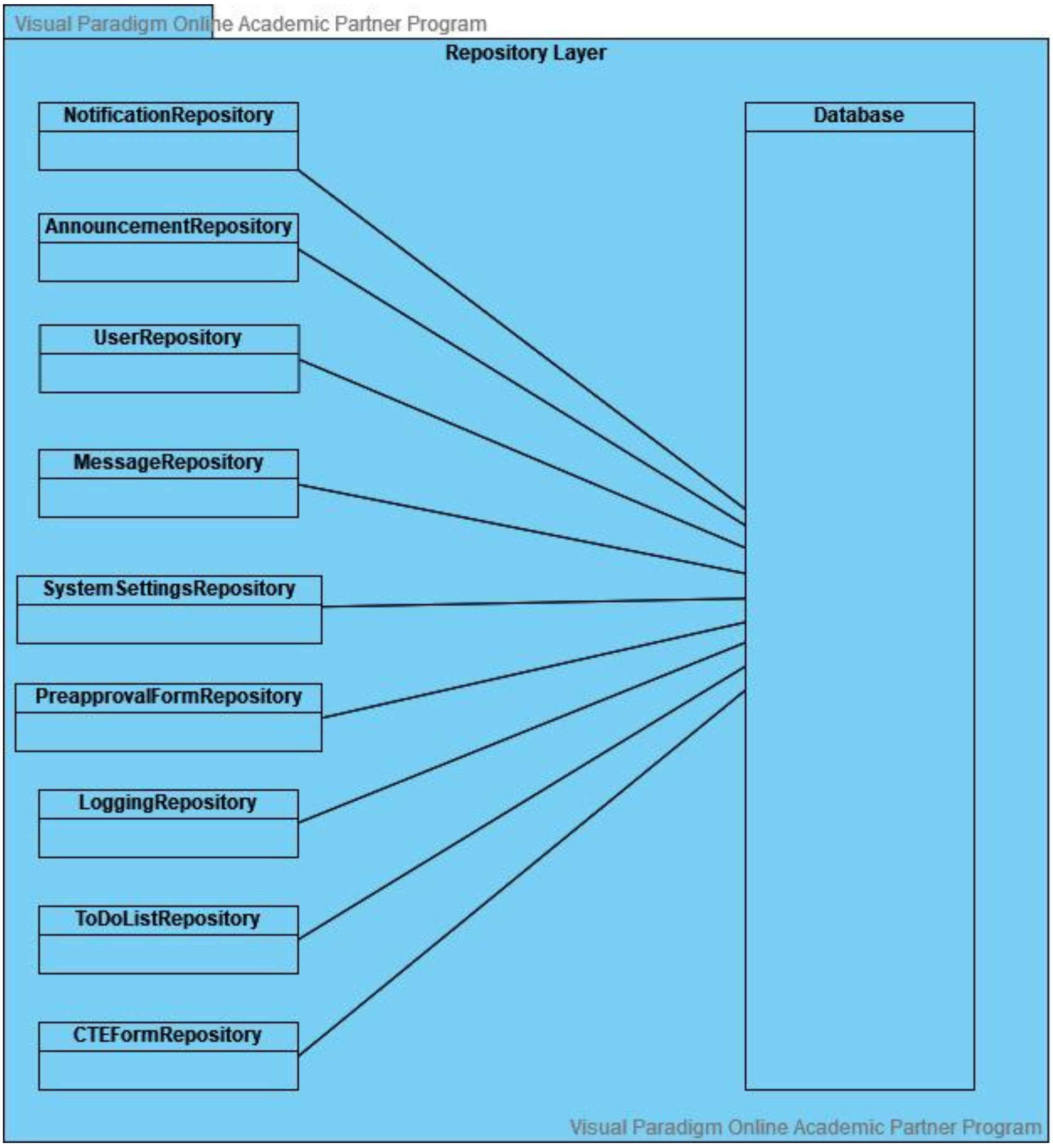


Figure 5: ERSMS Repository Layer Diagram

Data 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. Repository classes are used to execute CRUD operations (Create, Read, Update and Delete) on the database.

3.4.2 Controller and Service Layer

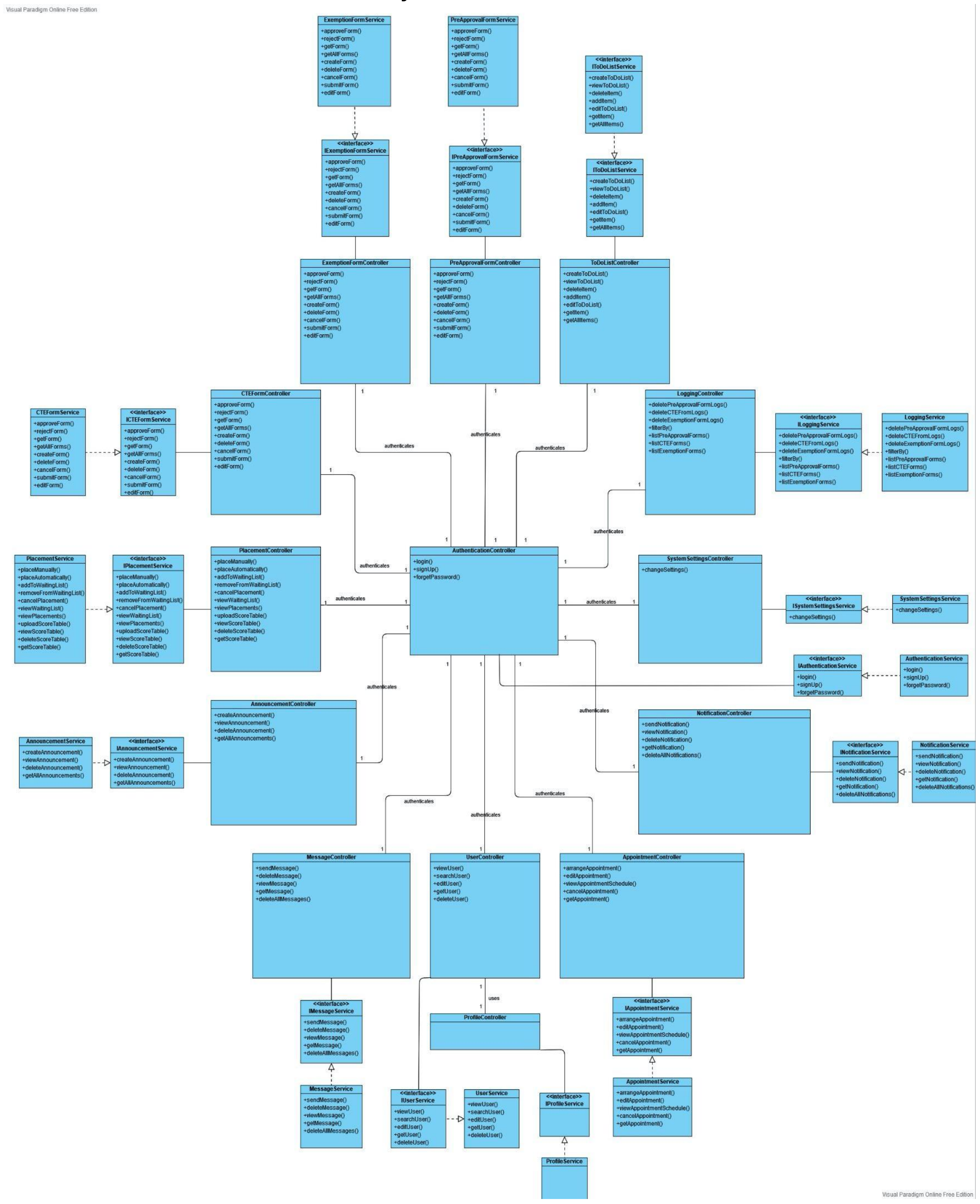


Figure 6: ERSMS Controller and Service Layer Diagram

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.

Controller and Service Layer

Controllers in this layer are handling the HTTP requests sent by the client, by calling its corresponding service in the service layer (for instance, PreApprovalFormController calls PreApprovalFormService) for executing the action specified by the request. All of the controllers mentioned in this section returns an ActionResult type which corresponds to a HTTP code (like 200, 404, 500). Request details are taken from the HTTP request body.

Note: In each class the following operations exist unless stated otherwise:

- Default constructors (for dependency injection)

Controllers:

AuthenticationController

AuthenticationController is used to handle the authentication and authorization requests.

- **public login():** This operation is used to handle authentication requests, in essence, logging in requests.
- **public signUp():** This operation is used to handle requests that are responsible from signing up.
- **public forgetPassword():** This operation is used to handle requests for resetting a registered user’s password.

SystemSettingsController

This controller is used to handle the requests regarding the settings of the system.

- **public changeSettings():** This operation is used for handling requests that are responsible from changing a system setting, specified by the user in the request.

LoggingController

This controller is used to handle the requests regarding the form logs.

- **public deletePreApprovalFormLogs():** This operation is used for handling the requests regarding the deletion of pre-approval form log.
- **public deleteCTEFormLogs():** This operation is used for handling the requests regarding the deletion of CTE form log.
- **public deleteExemptionFormLogs():** This operation is used for handling the requests regarding the deletion of course exemption form log.
- **public filterBy():** This operation is used for handling the requests regarding the filtering the form logs with a desired set of criterions.
- **public listPreApprovalForms():** This operation is used for handling the requests about fetching the pre-approval form logs.
- **public listCTEForms():** This operation is used for handling the requests about fetching the CTE form logs.
- **public listExemptionForms():** This operation is used for handling the requests about fetching the course exemption form logs.

ToDoListController

This controller is used to handle the requests regarding to the to-do lists and to-do list items.

- **public createToDoList():** This operation is used for handling the requests that are responsible from creating a new to-do list for the user.
- **public viewToDoList():** This operation is used for handling the requests regarding the fetching of the to-do list of a user.
- **public deleteItem():** This operation is used for handling the requests that are used for the deletion of a to-do list item.
- **public addItem():** This operation is used for handling the requests that are responsible from adding (creating) a new to-do list item to a user.
- **public editToDoList():** This operation is used for handling the requests that are responsible from editing a user’s to-do list.
- **public getItem():** This operation is used for handling the requests regarding the fetching of a to-do list item (singular).
- **public getAllItems():** This operation is used for handling the requests that are responsible from the fetching of all the to-do list items in the database.

PreApprovalFormController

This controller is used to handle the requests regarding the pre-approval forms.

- **public approveForm():** This operation is used for handling the requests that are responsible from the approval of a pre-approval form. Requests sent by exchange coordinators.
- **public rejectForm():** This operation is used for handling the requests that are responsible from the rejection of a pre-approval form. Requests sent by exchange coordinators.
- **public getForm():** This operation is used for handling the requests regarding the fetching of a pre-approval form.
- **public getAllForms():** This operation is used for handling the requests regarding the fetching of all pre-approval forms.
- **public createForm():** This operation is used for handling the requests that are responsible from creating a new pre-approval form.
- **public deleteForm():** This operation is used for handling the requests that are responsible from deleting a pre-approval form.
- **public cancelForm():** This operation is used for handling the requests that are responsible from the cancelation of a pre-approval form (by the student).
- **public submitForm():** This operation is used for handling the requests that are responsible from the submission of the pre-approval form (for the approval).
- **public editForm():** This operation is used for handling the requests that are responsible from the editing of the existing pre-approval form.

ExemptionFormController

This controller is used to handle the requests regarding the course exemption forms.

- **public approveForm():** This operation is used for handling the requests that are responsible from the approval of a course exemption form. Requests sent by exchange coordinators and course instructors.
- **public rejectForm():** This operation is used for handling the requests that are responsible from the rejection of a course exemption form. Requests sent by exchange coordinators and course instructors.

- **public getForm():** This operation is used for handling the requests regarding the fetching of course exemption form.
- **public getAllForms():** This operation is used for handling the requests regarding the fetching of all course exemption forms.
- **public createForm():** This operation is used for handling the requests that are responsible from creating a new course exemption form.
- **public deleteForm():** This operation is used for handling the requests that are responsible from deleting a course exemption form.
- **public cancelForm():** This operation is used for handling the requests that are responsible from the cancelation of a course exemption form (by the student).
- **public submitForm():** This operation is used for handling the requests that are responsible from the submission of the course exemption form (for the approval).
- **public editForm():** This operation is used for handling the requests that are responsible from the editing of the existing course exemption form.

CTEFormController
This controller is used to handle the requests regarding the CTE forms.

- **public approveForm():** This operation is used for handling the requests that are responsible from the approval of a CTE form. Requests sent by exchange coordinators, department chair and dean.
- **public rejectForm():** This operation is used for handling the requests that are responsible from the rejection of a CTE form. Requests sent by exchange coordinators, department chair and dean
- **public getForm():** This operation is used for handling the requests regarding the fetching of CTE form.
- **public getAllForms():** This operation is used for handling the requests regarding the fetching of all CTE forms.
- **public createForm():** This operation is used for handling the requests that are responsible from creating a new CTE form (requests sent by the coordinators).
- **public deleteForm():** This operation is used for handling the requests that are responsible from deleting a CTE form.
- **public cancelForm():** This operation is used for handling the requests that are responsible from the cancelation of a CTE form (by the exchange coordiantor).
- **public submitForm():** This operation is used for handling the requests that are responsible from the submission of the CTE form (for the approval).
- **public editForm():** This operation is used for handling the requests that are responsible from the editing of the existing CTE form.

PlacementController
This controller is used to handle the requests regarding the placement operations of students.

- **public placeManually():** This operation is used for handling the requests that are responsible from setting a student’s host university manually.
- **public placeAutomatically():** This operation is used for handling the requests regarding the auto placement mechanism for students.
- **public addToWaitingList():** This operation is used for handling the requests that are responsible from adding a student to the placement waiting list.
- **public removeFromWaitingList():** This operation is used for handling the requests that are responsible from removing a student from the placement waiting list.
- **public cancelPlacement():** This operation is used for handling the requests that are responsible from canceling a student’s placement.

- **public viewWaitingList():** This operation is used for handling the requests that are responsible from fetching the waiting list.
- **public viewPlacements():** This operation is used for handling the requests that are responsible from fetching the placement list of students.
- **public uploadScoreTable():** This operation is used for handling the requests regarding the upload of the score table (by the officer of OISEP) of the students.
- **public viewScoreTable():** This operation is used for handling the requests that are responsible from fetching the score table of the students.
- **public deleteScoreTable():** This operation is used for handling the requests that are responsible from deleting the score table of the students.
- **public getScoreTable():** This operation is used for handling the requests that are responsible from downloading the score table (different than viewing, which returns the table itself rather than the document as downloadable) of the students.

AnnouncementController

AnnouncementController is used to handle the requests regarding the announcements in the system.

- **public createAnnouncement():** This operation is used for handling the requests regarding the creation of a new announcement (used by the exchange coordinators).
- **public viewAnnouncement():** This operation is used for handling the requests regarding the fetching of the announcements for a user.
- **public deleteAnnouncement():** This operation is used for handling the requests regarding the deletion of an announcement.
- **public getAllAnnouncements():** This operation is used for handling the requests that are responsible from fetching all the announcements.

MessageController

MessageController is used for handling the requests regarding the messaging mechanism. This controller is used by the students and exchange coordinators (by sending requests).

- **public sendMessage():** This operation is used for handling the requests regarding the message sending mechanism.
- **public deleteMessage():** This operation is used for handling the requests regarding the deletion of a message.
- **public viewMessage():** This operation is used for handling the requests that are responsible from the fetching messages of a user.
- **public getMessage():** This operation is used for handling the requests that are responsible from fetching of a message (singular, can be any message) if the user is authorized.
- **public deleteAllMessages():** This operation is used for handling the requests that are responsible from deleting all the messages of a user.

UserController

This controller is used for handling the requests regarding the operations on users.

- **public viewUser():** This operation is used for handling the requests that are responsible from fetching the current users' user object.
- **public searchUser():** This operation is used for handling the requests that are responsible for checking whether a user is existing or not.
- **public editUser():** This operation is used for handling the requests that are responsible from editing the user's information.
- **public getUser():** This operation is used for handling the requests that are responsible from the fetching of a specified user.

- **public deleteUser():** This operation is used for handling the requests regarding the deletion of a user from the system.

AppointmentController
This controller is used for handling the requests regarding the appointments.

- **public arrangeAppointment():** This operation is used for handling the requests that are responsible from creating new appointments in the system.
- **public editAppointment():** This operation is used for handling the requests that are responsible from editing an existing appointment.
- **public viewAppointmentSchedule():** This operation is used for handling the requests that are responsible from fetching a user's appointment schedule (waiting appointments of a user).
- **public cancelAppointment():** This operation is used for handling the requests that are responsible from canceling an existing appointment from a user.
- **public getAppointment():** This operation is used for handling the requests that are responsible from fetching a specified appointment of a user.

NotificationController
This controller is used for handling the requests regarding the notifications.

- **public sendNotification():** This operation is used for handling the requests that are responsible from sending a notification to a list of users.
- **public viewNotification():** This operation is used for handling the requests that are responsible from fetching the notifications of a specified user.
- **public deleteNotification():** This operation is used for handling the requests that are responsible from deleting a specified notification.
- **public getNotification():** This operation is used for handling the requests that are responsible from fetching a notification (singular).
- **public deleteAllNotifications():** This operation is used for handling the requests that are responsible from deleting all the notifications of a user.

Services:

Every service class implements its corresponding interface (can be seen in the figure). Thus, there is no need for explanation of interfaces, every function declared in the interface has the same functionality as its implementation class, which is described in this section.

SystemSettingsService
This service is used for changing the settings of the system.

- **public changeSettings():** This operation is used for changing the settings of the application.

LoggingService
This service is used for operating on form logs.

- **public deletePreApprovalFormLogs():** This operation is used for deleting pre-approval forms.
- **public deleteCTEFormLogs():** This operation is used for deleting CTE forms.
- **public deleteExemptionFormLogs():**This operation is used for deleting course exemption forms
- **public filterBy():** This operation is used for filtering the form logs with a desired set of criteria.

- **public listPreApprovalForms():** This operation is used for listing the pre-approval form logs.
- **public listCTEForms():** This operation is used listing the CTE form logs.
- **public listExemptionForms():** This operation is used for listing the course exemption form logs.

ToDoListService
This service is used for operating on to-do lists and to-do list items.

- **public createToDoList():** This operation is used for creating a new to-do list for the user.
- **public viewToDoList():** This operation is used for viewing the to-do list of a user.
- **public deleteItem():** This operation is used for deleting an item of a to-do list.
- **public addItem():** This operation is used for adding (creating) a new item to the to-do list of a user.
- **public editToDoList():** This operation is used for editing a user’s to-do list.
- **public getItem():** This operation is used for fetching a to-do list item.
- **public getAllItems():** This operation is used for fetching all the to-do list items of a user in the database.

PreApprovalFormService
This service is used for operating on the pre-approval forms.

- **public approveForm():** This operation is used for approving a pre-approval form.
- **public rejectForm():** This operation is used for rejecting a pre-approval form.
- **public getForm():** This operation is used for fetching a pre-approval form.
- **public getAllForms():** This operation is used for fetching all pre-approval forms.
- **public createForm():** This operation is used for creating a new pre-approval form.
- **public deleteForm():** This operation is used for deleting a pre-approval form.
- **public cancelForm():** This operation is used for cancelling a pre-approval form by the user.
- **public submitForm():** This operation is used for submitting a pre-approval form (for the approval).
- **public editForm():** This operation is used for editing an existing pre-approval form.

ExemptionFormService
This service is used for operating on the course exemption forms.

- **public approveForm():** This operation is used for approving a course exemption form. This operation can only be done by exchange coordinators and course instructors.
- **public rejectForm():** This operation is used for rejecting a course exemption form. This operation can only be done by exchange coordinators and course instructors.
- **public getForm():** This operation is used for fetching a course exemption form.
- **public getAllForms():** This operation is used for fetching all course exemption forms.
- **public createForm():** This operation is used for creating a new course exemption form.
- **public deleteForm():** This operation is used for deleting a course exemption form.
- **public cancelForm():** This operation is used for cancelling a course exemption form by the student.
- **public submitForm():** This operation is used for submitting a course exemption form (for the approval).
- **public editForm():** This operation is used for editing of an existing course exemption form.

CTEFormService

This service is used for operating on the CTE forms.

- **public approveForm():** This operation is used for approving a CTE form. This operation can only be done by exchange coordinator, department chair and dean.
- **public rejectForm():** This operation is used for rejecting a CTE form. This operation can only be done by exchange coordinator, department chair and dean.
- **public getForm():** This operation is used for fetching a CTE form.
- **public getAllForms():** This operation is used for fetching all CTE forms.
- **public createForm():** This operation is used for creating a new CTE form.
- **public deleteForm():** This operation is used for deleting a CTE form.
- **public cancelForm():** This operation is used for cancelling a CTE form.
- **public submitForm():** This operation is used for submitting a CTE form (for the approval).
- **public editForm():** This operation is used for editing of an existing CTE form.

PlacementService

This service is used conducting placement operations of students.

- **public placeManually():** This operation is used for setting a student’s host university manually.
- **public placeAutomatically():** This operation is used for conducting auto placement for the students.
- **public addToWaitingList():** This operation is used for adding a student to the placement waiting list.
- **public removeFromWaitingList():** This operation is used for removing a student from the placement waiting list.
- **public cancelPlacement():** This operation is used for canceling a student’s placement.
- **public viewWaitingList():** This operation is used for fetching the waiting list.
- **public viewPlacements():** This operation is used for fetching the placement list of students.
- **public uploadScoreTable():** This operation is used for uploading the score table (by the officer of OISEP) of the students.
- **public viewScoreTable():** This operation is used for fetching the score table of the students.
- **public deleteScoreTable():** This operation is used for deleting the score table of the students.
- **public getScoreTable():** This operation is used for downloading the score table (different than viewing, which returns the table itself rather than the document as downloadable) of the students.

AnnouncementService

AnnouncementService is used for conducting the announcement methods of the system.

- **public createAnnouncement():** This operation is used for creating a new announcement (used by the exchange coordinators).
- **public viewAnnouncement():** This operation is used for fetching an announcement.
- **public deleteAnnouncement():** This operation is used for deleting an announcement.
- **public getAllAnnouncements():** This operation is used for fetching all the announcements.

MessageService

MessageService is used for using the messaging mechanism. It is used by the students and exchange coordinators.

- **public sendMessage():** This operation is used for sending a message.

- **public deleteMessage():** This operation is used for deleting a message.
- **public viewMessage():** This operation is used for fetching messages of a user.
- **public getMessage():** This operation is used for fetching a message (singular, can be any message) if the user is authorized.
- **public deleteAllMessages():** This operation is used for deleting all the messages of a user.

UserService
This service is used for operating on users.

- **public viewUser():** This operation is used for fetching the current users' user object.
- **public searchUser():** This operation is used for checking whether a user exists or not.
- **public editUser():** This operation is used for editing the user's information.
- **public getUser():** This operation is used for fetching a specified user.
- **public deleteUser():** This operation is used for deleting a user from the system.

AppointmentService
This service is used for conducting and managing appointments.

- **public arrangeAppointment():** This operation is used for creating new appointments in the system.
- **public editAppointment():** This operation is used for editing an existing appointment.
- **public viewAppointmentSchedule():** This operation is used for fetching a user's appointment schedule (waiting appointments of a user).
- **public cancelAppointment():** This operation is used for canceling an existing appointment of a user.
- **public getAppointment():** This operation is used for fetching a specified appointment of a user.

NotificationService
This service is used for sending and managing notifications.

- **public sendNotification():** This operation is used for sending a notification to a list of users.
- **public viewNotification():** This operation is used for fetching the notifications of a specified user.
- **public deleteNotification():** This operation is used for deleting a specified notification.
- **public getNotification():** This operation is used for fetching a notification (singular).
- **public deleteAllNotifications():** This operation is used for deleting all the notifications of a user.

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

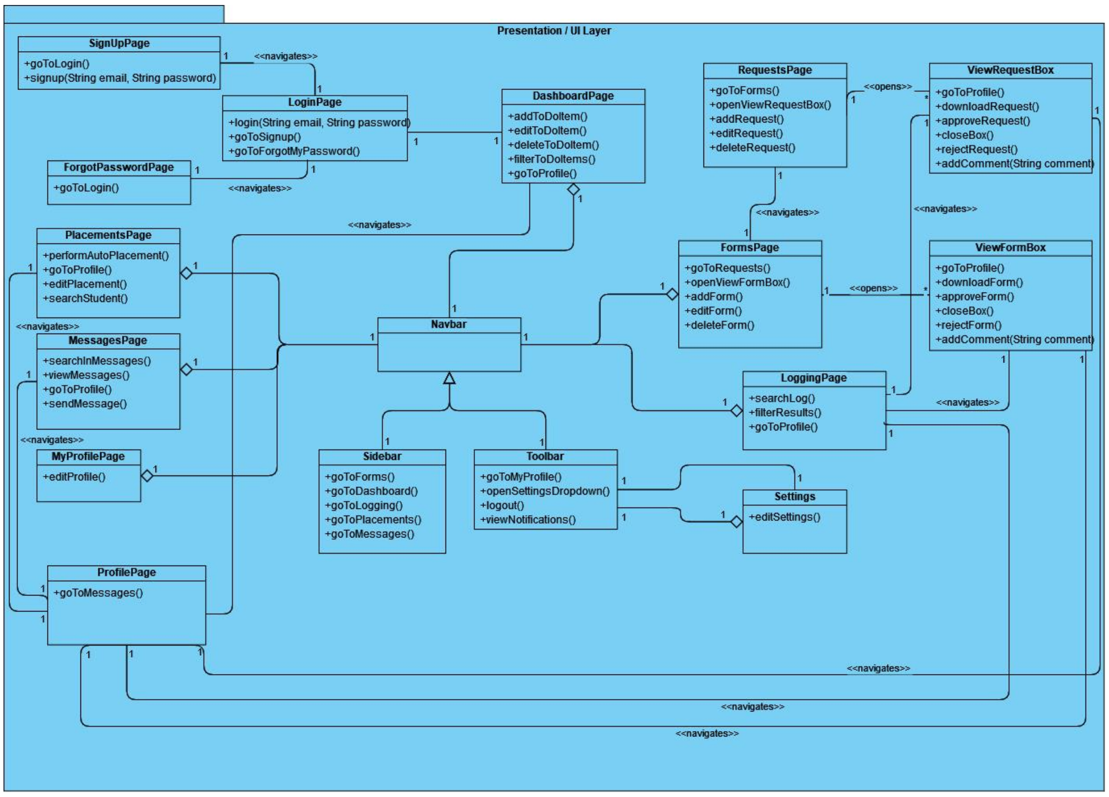


Figure 7: ERSMS Presentation / UI Layer Diagram

Presentation / UI Interface Layer

SignupPage
The SignupPage class constructs the view for the signup page, which allows the users to create a new account in ERSMS.

Methods:

- **goToLogin():** Navigates the user to the login page.
- **signup(String email, String password):** Allows the user to sign up to the ERSMS system by checking the credentials.

ForgotPasswordPage
The ForgotPasswordPage class constructs the view for the forgot password page, which allows the users to change their password.

Methods:

- **goToLogin():** Navigates the user to the login page.

LoginPage
The LoginPage class constructs the view for the login page, which allows the users to login their accounts.

Methods:

- **login(String email, String password):** Allows the user to login by checking their credentials.
- **goToSignup():** Navigates the user to the signup page.
- **goToForgotMyPassword():** Navigates the user to forgot my password page.

DashboardPage
The DashboardPage class constructs the view for the dashboard page, which allows the users to see an overview of the form submissions and their to-do lists.

Methods:

- **addToDoItem():** Allows the user to add an item to the to-do list.
- **editToDoItem():** Allows the user to edit an item in the to-do list.
- **deleteToDoItem():** Allows the user to delete an item from the to-do list.
- **filterToDoItem():** Allows the user to search an item in the to-do list by filtering.
- **goToProfile():** Navigates the user to the profile page.

Sidebar
The Sidebar class constructs the view for the sidebar of the application, which allows the users to navigate to the different pages in the ERSMS application. These pages are Forms and Requests, Dashboard, Logging, Placements and Messages

Methods:

- **goToForms():** Navigates the user to the Forms and Requests page.
- **goToDashboard():** Navigates the user to the Dashboard page.
- **goToLogging():** Navigates the user to the Logging page.
- **goToPlacements():** Navigates the user to the Placements page.
- **goToMessages():** Navigates the user to the Messages page.

Toolbar
The Toolbar class constructs the view for the toolbar of the application, which allows the users to see their notifications, go to the settings and their profiles, and log out.

Methods:

- **goToMyProfile():** Navigates the user to the their profile page.
- **openSettingsDropdown():** Opens the settings dropdown.
- **logout():** Allows the user to logout from their account.
- **viewNotifications():** Opens the notifications dropdown.

ProfilePage
The ProfilePage class constructs the view for the profile pages of the other users, which allows the users to view the other people's profiles.

Methods:

- **goToMessages():** Navigates the user to the messages page.

MyProfilePage
The MyProfilePage class constructs the view for the profile pages of their own, which allows the users to see their own profile pages and makes changes in their profiles.

Methods:

- **editProfile():** Allows the users to edit their profiles.

MessagesPage
The MessagesPage class constructs the view for the messages page, which allows the users to see their old messages and send new messages to the other users.

Methods:

- **searchInMessages():** Allows the users to search in their old messages.
- **viewMessages():** Loads the messages to the page when the message box of a user is clicked.
- **goToProfile():** Allows the user to navigate to the other users' profiles from the messages page.
- **sendMessage():** Allows the users to send a message to a user.

PlacementsPage
The PlacementsPage class constructs the view for the placements page, which allows the coordinators to see and control the placements of the exchange students, see the score tables and perform autoplacement.

Methods:

- **performAutoPlacement():** Allows the coordinators to make ERSMS make placements of the students according to the score table.
- **goToProfile():** Allows the coordinators to navigate to the other users' profiles from the placements page by on a table row associated to a student.
- **searchStudent():** Allows the coordinators to search students in the placements.

LoggingPage
The LoggingPage class constructs the view for the logging page, which allows the coordinators to see logs of the previous forms and requests.

Methods:

- **goToProfile():** Allows the coordinators to navigate to the other users' profiles from the logging page by on a table row associated to a student.
- **filterResults():** Allows the coordinators to search the logs by filtering the results.
- **searchLog():** Allows the coordinators to search forms in the logs.

SettingsDropdown
The SettingsDropdown class constructs the view for the settings dropdown, which allows the users to change the application settings.

Methods:

- **editSettings():** Allows the users to edit their application settings.

RequestsPage
The RequestsPage class constructs the view for the requests, which allows the students to create new requests and view their previous requests, and allows the coordinators to view the requests and respond to them.

Methods:

- **goToForms():** Allows the users to navigate to the forms page.
- **openViewRequestBox():** Allows the users to see the details of a request by opening a new dialog.
- **addRequest():** Allows the students to add new requests.
- **editRequest():** Allows the students to edit their requests.
- **deleteRequest():** Allows the students to delete their requests.

FormsPage
The FormsPage class constructs the view for the forms, which allows the students to create new forms and view their previous forms, and allows the coordinators to view the forms and respond to them.

Methods:

- **goToRequests**(): Allows the users to navigate to the requests page.
- **openViewFormBox():** Allows the users to see the details of a form by opening a new dialog.
- **addForm():** Allows the students to add new forms.
- **editForm():** Allows the students to edit their forms.
- **deleteForm():** Allows the students to delete their forms.

ViewFormBox
The ViewFormBox class constructs the view for the form dialogs, which allows the users to see the details of the forms.

Methods:

- **goToProfile():** Allows the users to navigate to the profile of the form’s creator.
- **downloadForm():** Allows the users to download the form to their local storage.
- **approveForm():** Allows the coordinators to approve the form.
- **closeBox():** Allows the users to close the form dialog.
- **rejectForm():** Allows the coordinators to reject the form.
- **addComment():** Allows the coordinators to add comment to the form.

ViewRequestBox
The ViewRequestBox class constructs the view for the request dialogs, which allows the users to see the details of the requests.

Methods:

- **goToProfile():** Allows the users to navigate to the profile of the request’s creator.
- **downloadRequest():** Allows the users to download the request to their local storage.
- **approveRequest():** Allows the coordinators to approve the request.
- **closeBox():** Allows the users to close the request dialog.
- **rejectRequest():** Allows the coordinators to reject the request.
- **addComment():** Allows the coordinators to add comment to the request.

3.5 Design Patterns

- **Facade:** Facade is a structural design pattern designed to hide the complexities by abstracting the classes that make up the parts of a subsystem from the client. In that sense, ERSMS's layered architecture employs the Façade design pattern such that higher layers are unaware of the on-going complexities in the lower layers. For example, UserController asks the list of computer engineering students from the UserService, and UserService asks the list of all students from the UserRepository. Hence, filtering process in the service layer and database interaction in the repository layer become abstracted from the controller layer. Furthermore, ERSMS uses Entity Framework that converts entity models to database migration schemas automatically which can also be considered as an example usage of facade design pattern since it abstracts the complexity of dealing with different relational database mapping conventions from the ERSMS.
- **Dependency Injection:** Dependency Injection is a design pattern that injects the dependencies into concrete classes. In ERSMS, we use dependency injection by injecting Interfaces to Services through .NET provided Startup.cs interface.
- **Singleton:** Singleton is a creational pattern to ensure that a specific class has only one instance. In ERSMS, BCryptPasswordHasher is an example of this design pattern. This class is a static class with only one instance which is used for sign up password encryption process.
- **Adapter:** Adapter is a structural design pattern that allows incompatible objects to collaborate. In ERSMS, we link DomainUser's Id field with the Microsoft's Identity Framework such that DomainUser can benefit from the properties of the IdentityFramework's AppUser to provide easiness in the authentication processes (i.e., DomainUser adapts the login and sign up capabilities from the AppUser).

4 Improvement Summary

In the second iteration of the design report, necessary adjustments were made to correct some mistakes in the diagrams and explanations. For example, the deployment diagram was transferred to the hardware/software mapping part as asked in the feedback. However, the key points were the class explanations. All of the methods and the attributes of data layer and service/controller layers were explained in detail.

Furthermore, the user interface layer was also made more detailed to resemble the code written. Previously, the multiplicities of the classes were missing in the first iteration of the design report. These were added with the relations between the classes. Finally, the used design patterns like facade design pattern were explained in detail.

5 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.