CS319

Object Oriented Software Engineering

Design Report

Instructor: Eray Tüzün

Project: Student Internship System

Group Name: Quaso

Members:

Zeynep Doğa Dellal 22002572

Yağız Alkılıç 22003281

Muhammad Ali Waris 22001037

Yağız Berk Uyar 21902318

Mustafa Hamit Dölek 21703136

Table of Contents

1. Introduction 5

1.1 Purpose of the system 5

1.2 Design goals 5

1.2.1 Functionality 5

1.2.2 Information security & privacy 5

2. High-Level Software Architecture 6

2.1 Subsystem Decomposition 6

2.2 Deployment Diagram 7

2.3 Hardware/Software Mapping 7

2.4 Persistent Data Management 8

2.5 Access Control and Security 8

2.6 Boundary Conditions 11

2.6.1 Initialization 11

2.6.2 Termination 11

2.6.3 Failure 12

3. Low Level Design 12

3.1 Object design trade-offs 12

3.1.1 Performance vs. Development Time 12

3.1.3 Code Reusability vs. Customization 12

3.1.4 Querying vs. Data İntegrity 13

3.1.5 Integration vs. Isolation 13

3.1.6 Security vs. Performance 13

3.2 Final Object Design 15

3.2.1 User Interface Layer 16

3.2.2 Web Server Layer 17

3.2.3 Data Management Layer 18

3.3 Packages 19

3.3.1 Internal Packages 19

3.3.2 External Packages 20

3.4 Class Interfaces 21

3.4.1 Entities 21

1. AcademicPersonnel 21

2. Administor 21

3. AdministrativeAction 21

4. AdministrativeActionType 22

5. Company 22

6. Company Report 22

7. CompanyForm 22

8. CompanyOwner 23

9. Coordinator 23

10. Course Description 23

11. CourseType 23
12. DepartmentType 24
13. DescriptionType 24
14. Evaluator 24
15. Feedback 24
16. GradeType 24
17. Grading Item 25
18. Language 25
19. Notification 25
20. Settings 25
21. Student 25
22. Student Report 26
23. TA 26
24. User 26
3.4.2 Web Server 27
1. AdministrativeManagerController 27
2. AdministrativeManagerService 27
3. AssignmentManagerController 27
4. AssignmentManagerService 28
5. AuthenticationManagerController 28
6. AuthenticationManagerService 29
7. CompanyManagerController 29
8. CompanyManagerService 29
9. CourseManagerController 30
10. CourseManagerService 30
11. FormManagerController 30
12. FormManagerService 31
13. NotificationManagerController 32
14. NotificationManagerService 32
15. ReportManagerController 32
16. ReportManagerService 33
3.4.3 User Interface 33
1. DescriptionsPage 33
2. EditUserPage 33
3. FeedbackSubmissionPage 34
4. ForgotPasswordPage 34
5. GradeSubmissionPage 34
6. LoginPage 34
7. NotificationItem 34
8. NotificationsPage 34
9. ProfilePage 35
10. ReportsPage 35

11. ReportSubmissionPage 35
12. SearchUserPage 35
13. SettingsPage 35
14. SideBar 36
15. StaffInfoPage 36
16. StudentsPage 36
17. TopBar 36
18. UserInitializationPage 37
19. ViewCompanyFormPage 37
20. ViewGradePage 37
21. ViewReportPage 37
3.5 Design Patterns 38
3.5.1 Singleton Pattern 38
3.5.2 Factory Method Pattern 38
3.5.3 Façade Pattern 39
3.5.4 Adapter Pattern 39
5. Improvement Summary 40
6. Glossary 40
7. References 41

1. Introduction

The app aims to streamline the internship management process by providing a platform for communication, document sharing, grading, and progress tracking among the different stakeholders involved. This report outlines the purpose, design goals, and high-level software architecture of a software system, as well as delves into the low-level design details and the use of design patterns. It provides a comprehensive understanding of how the software system has been designed to function efficiently and effectively.

1.1 Purpose of the system

The purpose of the system is to provide a platform for managing internship programs between companies, instructors, evaluators, and students. It aims to simplify and streamline the process of submitting, reviewing, and grading reports and to provide all stakeholders with the necessary tools and information to facilitate a successful internship program. The system also includes features such as user management, document management, and progress tracking to enhance communication and collaboration among the different parties involved.

1.2 Design goals

1.2.1 Functionality

The design goals under the functionality header are focused on making the system user-friendly, efficient, flexible, scalable, and integrable. To ensure usability, the system should be easy to use and intuitive for all types of users, regardless of their technical proficiency. The system should also facilitate efficient communication and document management to streamline the internship program process. Additionally, the system should be flexible enough to adapt to different types of internship programs and requirements. It should be scalable and able to handle users and data up to 100GB and accommodate future growth. Furthermore, the system should be compatible with other tools and platforms to ensure seamless integration with existing workflows.

1.2.2 Information security & privacy

The design goals under the information security and privacy header aim to ensure the confidentiality, integrity, availability, compliance, and audibility of user data and documents. The system should ensure that all user data and documents are kept confidential and protected from unauthorized access. It should also ensure that user data and documents are accurate, complete, and unaltered. The system should guarantee that user data and documents are available to authorized users at all times. Additionally, the system should comply with all relevant data protection and privacy laws and regulations. It should maintain an audit trail of all user activities to facilitate accountability and traceability. By adhering to these design goals, the system can provide a secure and private platform for managing internship programs.

2. High-Level Software Architecture

2.1 Subsystem Decomposition

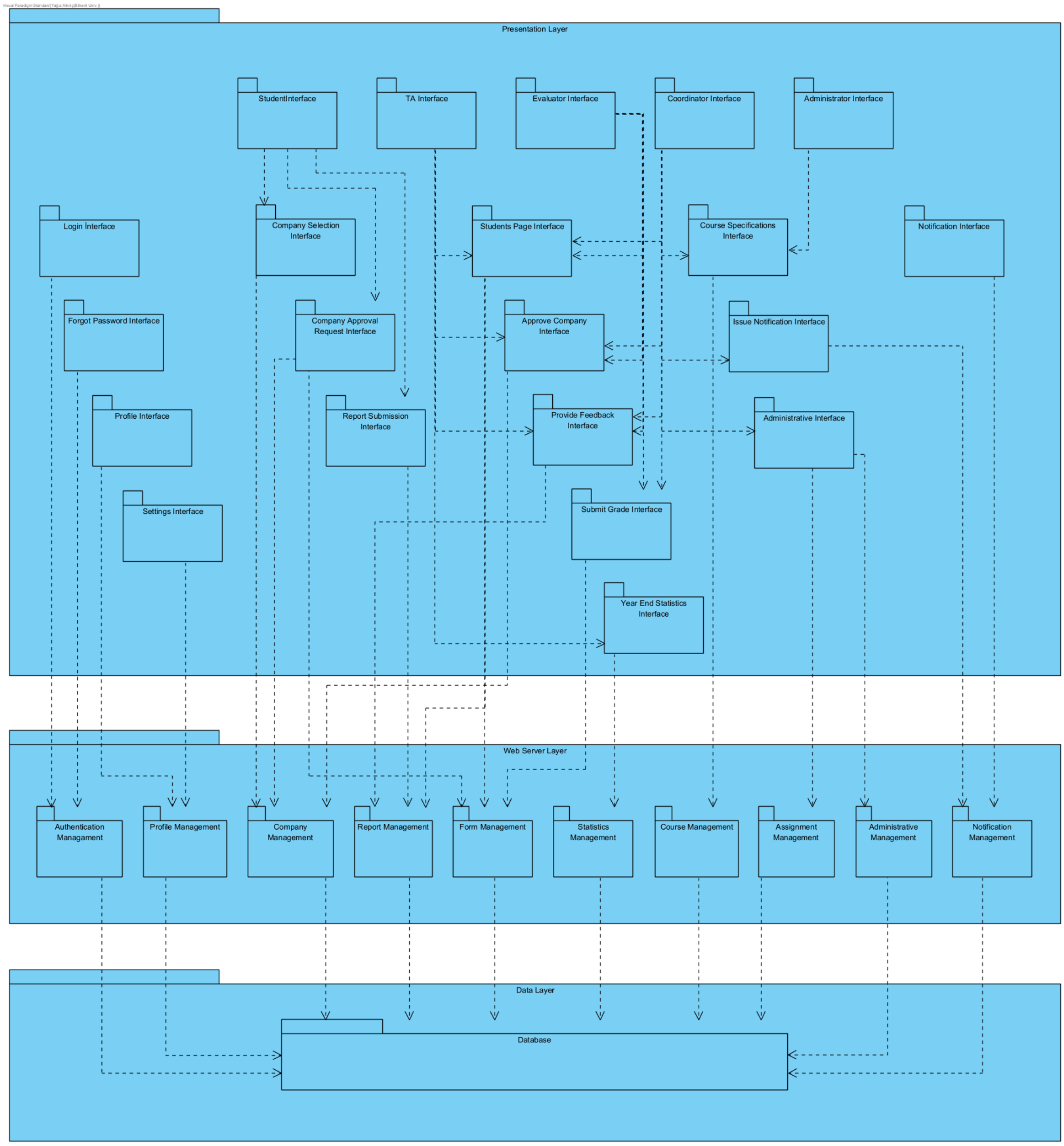


Figure 1. Subsystem Decomposition Diagram (High quality version: https://github.com/zedyjy/CS319-Project/blob/main/Diagrams/Subsytem%20Decomposition.png)

We decided to work with three different layers in terms of codability and efficient use of our system, these are the Presentation layer, Web Server Layer, and Data Layer. The different positioning of the Web Server Layer and the Presentation Layer was useful and sufficient for us to distinguish positions between the front-end and back-end in our team. In this way, we have avoided the possible coding process problems that may occur from the excess layers and preferred a simpler structure.

2.2 Deployment Diagram

Diagram below shows the interaction between the client, web server, and database. The client device represents the user's device, such as a desktop computer, laptop, or mobile device, from which they access the system. The web server component acts as the central processing unit that handles client requests and serves web pages. The database component stores and manages the system's data, providing a reliable and secure storage solution.

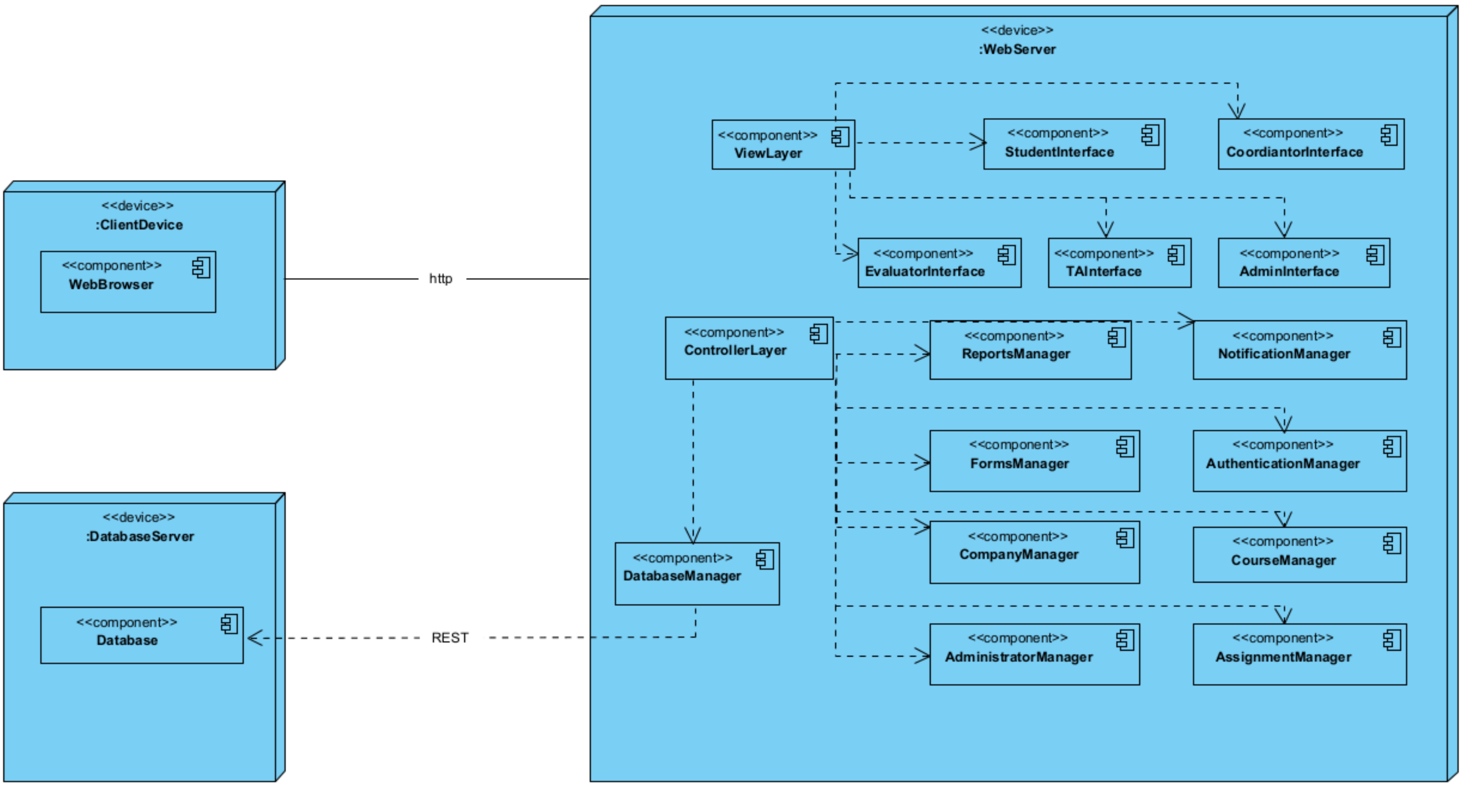


Figure 2. Deployment Diagram

2.3 Hardware/Software Mapping

We preferred to use Node.js in the backend infrastructure of our application, the main reason for this is that it provides us with flexibility in terms of storing files such as pdf files and profile photos to be uploaded to the system. In the frontend, we prefer to use HTML CSS JavaScript, and Bootstrap for forming and styling a responsive, stylish user interface. We prefer them because of their ease of use as well as backward compatibility with browsers. Node.js environments will be used to deploy the website and as a database, MongoDB will be preferred. System requirements to host/deploy the database are at least 10 GB of free disk space plus whatever space is necessary to hold our MongoDB data, at least 4 GB of RAM, and 64-bit architectures

Our system requires only a device capable of running a modern internet browser. While a keyboard and mouse are required to use it on a PC, other modern touchscreen devices have these peripherals built in, so no extra peripherals are required for any user. We will use a node.js based server in our project, the system requirements for the server that will host the website are Processor: Intel Core i5 or equivalent,2.0 GHz or higher clock speed Dual-core or higher. Memory(RAM): 16GB or higher. Storage: 100GB or higher of disk space. Operating System: Linux(eg., Ubuntu, CentOS). Network Bandwidth: Sufficient bandwidth based on anticipated traffic and file transfer requirements.

2.4 Persistent Data Management

The use of a secure and efficient database is crucial for data security and for creating a flawlessly functioning system. We decided that the most suitable database for our system is MongoDB. Some of the reasons we prefer MongoDB are its high scalability, easy data access, ease of data replication, and high-security measures. A document-based data model is used by MongoDB. This indicates that the data has a more adaptable structure and is stored in a JSON-like manner. Data can be stored, maintained, and queried fast using this data model. Some critical information to be used in the system such as User information, uploaded reports, feedback, and grades will be stored in this database.

2.5 Access Control and Security

Security and access control is crucial for our internship system, since our model uses the user passwords, ids, and emails from the SRS system, securing information is crucial. Authentication will be conducted over HTTPS. HTTPS ensures data encryption between the user's browser and web server, the data includes passwords, ids, and emails.

There is also the aspect of personal files and information inside of the system. There are different users such as evaluators, students, admins, and teaching assistants (TA). To give an example, evaluators are only able to see the assigned students' reports. This system is the same as TA's. Evaluators are not allowed to access other evaluators' student reports and feedback. Same with students, a student can only access their report and feedback. Seeing information about other students is not allowed. Shortly, there will be a system of security that also restricts information leaks between individuals and different types of users.

	Student	Evaluator	Coordinator	TA	Admin
Login	x	x	x	x	X
See Announcements	x	x	x	x	X
See Contact	x	x	x	x	X
Change Language	x	x	x	x	X
Change to Dark/Light Mode	x	x	x	x	X
Renew Password	x	x	x	x	X
Upload Report	x	x	x	x	
View Report	x	x	x	X	
Provide Feedback		x		X	
Upload Feedback		x		X	

Grade Student		x		x	
View Grade	x	x	x	x	
Search Student		x	x	x	x
Search Coordinator	x	x	x	x	x
Search TA	x	x	x	x	x
Assign Evaluator			x		x
Assign TA			x		x
View Profile	x	x	x	x	x
View Profile of Another User					x
Randomly Assign Students to Evaluators			x		x
Delete User					x

Add User					**x**

Table 1. User Abilities Table

2.6 Boundary Conditions

We aim to have as few failures as possible. We intend to implement a recovery system that can jump back on its feet to get to a stable state in the fastest way. Below is the boundary conditions state diagram of the system we intend to implement.

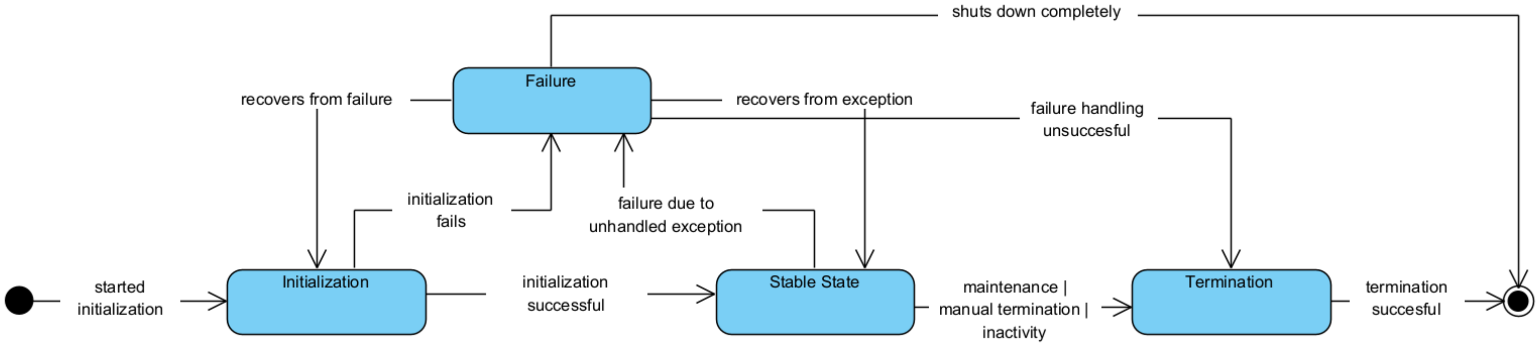


Figure 3. Boundary Conditions State Diagram

2.6.1 Initialization

The initialization process of the Quaso internship application is seamless and straightforward as it is a web-based platform. To start the system, the server is run from Apache, which allows users to access the application through their web browsers. No additional setup is required for users as they can simply log in using their STARS (Student Access to Records System) ID and password. Once logged in, the system retrieves relevant data from the database to initialize the user's personalized experience. The application leverages the Apache and mySQL environments for deployment and loads all necessary external dependencies on the server. This ensures a stable and reliable platform for users to interact with. Furthermore, individuals who do not have STARS credentials can still access certain types of information, such as general course information, documents, announcements, and contact details. The primary focus during initialization is to ensure the system is up and running smoothly, providing users with a seamless and user-friendly experience.

2.6.2 Termination

Within the Quaso internship application, there are scenarios that may lead to the complete shutdown of the system. This can be initiated by the administrator or may occur due to unexpected circumstances, such as security breaches or an overwhelming demand on the system. To ensure the continuity of the system, regular backups of server files and the database are taken. In the event of a shutdown, the system can be restored by reloading it from a recent backup. For server files, a Python script is employed to take regular backups using the 'tar' command, creating compressed backups in relevant formats, and storing them on a separate machine. Similarly, for the database, the 'mongodump' command of MongoDB is utilized to create regular backups, which are also managed through a Python script. These backup procedures guarantee that the most recent versions of files and information can be recovered in the event of a shutdown. When the system is unavailable, users will be greeted with a message stating, "The system is not available for use at the moment. Please try again in 10 minutes." This ensures clear communication and manages user expectations during periods of system termination.

2.6.3 Failure

In possible failures in the connected system, or within the Quaso system, failure will be handled by restarting the environment. If the system is still in a failure state, admins will be notified. The last saved data will be recovered. Internet loss on the client side may cause failure. If the user is trying to upload a big-sized file this may also cause unexpected failures. The Quaso internship system will have a message displayed about the failure that is happening.

3. Low Level Design

3.1 Object design trade-offs

3.1.1 Performance vs. Development Time

PHP is a language that has been around for a while. It has several frameworks and a huge developer community to take help from on different web sources. But here is the catch, PHP is known to create problems when the number of concurrent requests to the server increases by a certain amount. We expect at least 200 logged-in users during internship application times, where each of them could be uploading/modifying their reports or interacting with the server at any given time. Therefore, we went with Node.js as a back-end language so that it can support multiple concurrent requests at the same time, plus it also comes with ‘npm’, a really useful tool.

Trade Off: Using Node.js as a backend language will make the development slow in terms of the available sources to ask for help if the developers working on the site run into a bug. But this will pay off with a higher performing app in the end that can handle multiple users easily.

3.1.3 Code Reusability vs. Customization

Making customizations to views and controllers would require a higher amount of written code, which would possibly mean higher ‘ghost’ errors at project-completion.

Therefore, for the sake of easier maintainable code, there have been decisions to make a reusable component if possible in most cases.

Trade Off: Having more reusable components will make it harder to customize them further for each item specifically, for example having a red navbar for a happy student and a green navbar for a sad student, when having the same navbar for all students by default, although this is exaggerated, it is true. But having reusable components makes it easier for debugging and staying away from 'ghost' errors.

3.1.4 Querying vs. Data İntegrity

mySQL's relational data structure is easier to maintain as it enforces referential integrity and other constraints at the database level. MongoDB does not enforce data integrity in the same way, which can lead to more errors and inconsistencies in the data. However, MongoDB does offer some tools for ensuring data integrity, such as atomic transactions and document validation. Plus, MongoDB's query language and storage pattern are perfectly usable for our case, as all involved parties have a limited amount of data stored on the server, and that limit does not have an effect of any kind on run times or load times.

Trade Off: Using MongoDB will require us to keep it closely maintained to have a proper app with the required amount of data integrity, but querying will be relatively faster.

3.1.5 Integration vs. Isolation

Increased integration between objects and components will certainly make it easier to communicate and share data between them, however, this may also result in a higher degree of coupling between objects, which can make it harder to update or replace individual components in the future. Conversely, we also would probably make it harder to pass data and communicate between objects by using more isolation of objects and components, but we may have a more modular and loosely coupled object model that is easier to update in the future. That is why we went with more widespread isolation.

Trade Off: Using increased isolation between components will make it harder to communicate between components, but it will allow us to have easier updating/debugging of the app.

3.1.6 Security vs. Performance

We could have used basic hashing techniques when users input their passwords, but considering the number of systems in danger due to a password breach from an input field, a better hashing algorithm had to be used. As it is assumed that data will be taken from SRS, and it is assumed that SRS API will provide username/password data, etc., the same password is used for SRS too. Hence, we are going with **bcrypt.js** which uses the BCRYPT

CRYPT Algorithm which takes a 128-bit salt and encrypts a 192-bit magic value, making it secure enough for the project.

Trade Off: Using bcrypt.js will add extra scripts to the code and slow the performance, but the app will be more secure.

3.2 Final Object Design

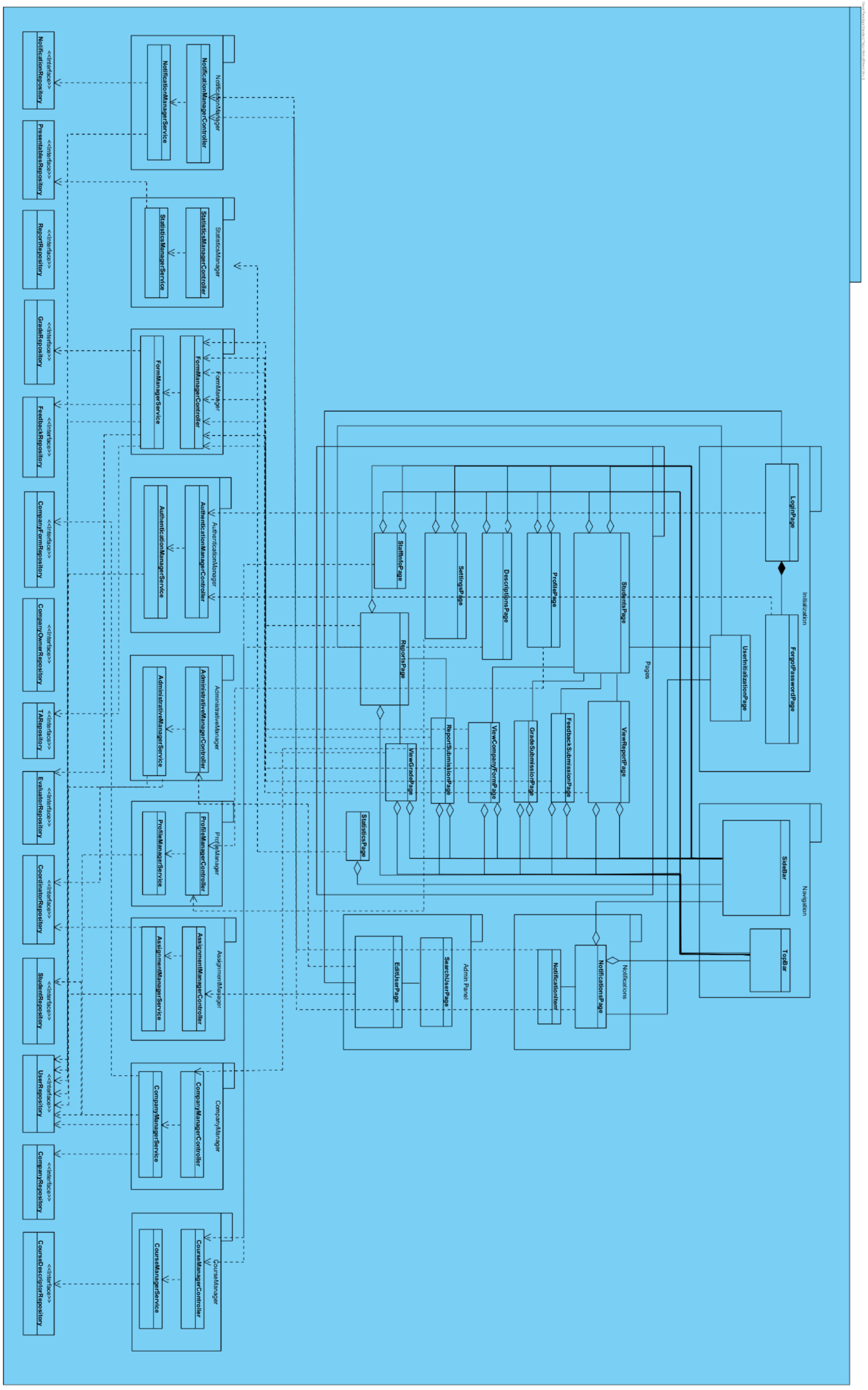


Figure 4. Final Object Design Diagram (High quality version: https://github.com/zedyjy/CS319-Project/blob/main/Diagrams/Final%20Object%20Design.png)

3.2.1 User Interface Layer

Visual Paradigm Standard(Yağız Alkılıç(Bilkent Univ.))

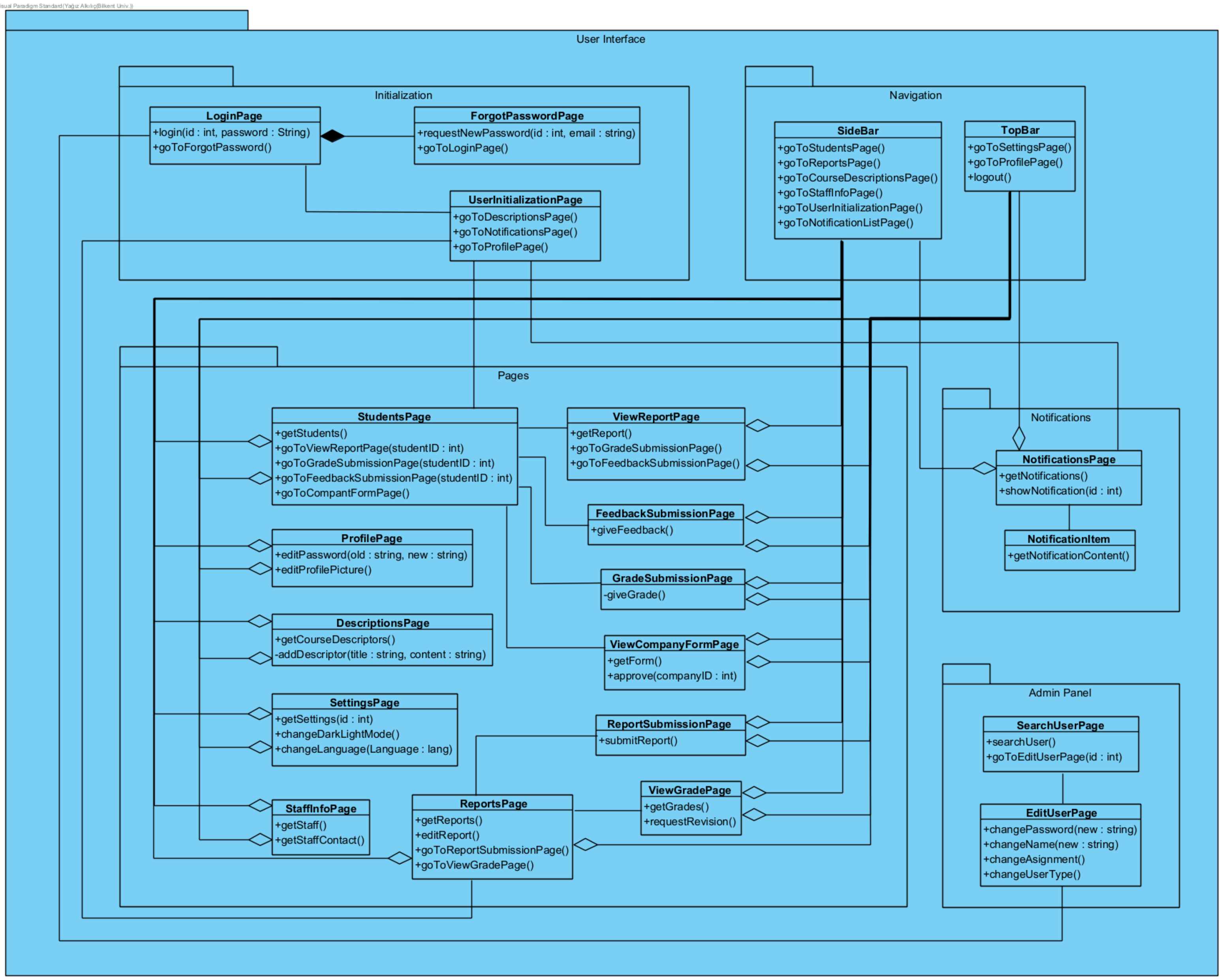


Figure 5. User Interface Management Layer (High quality version: https://github.com/zedyjy/CS319-Project/blob/main/Diagrams/User%20Interface.png)

The user interface layout forms the cornerstone of the presentation layer. The interfaces of the different types of users who will use the application are connected to this user interface layer. In this diagram, it can be observed how the pages in the application communicate with each other and detailed information about the pages themselves. Furthermore, groups such as navigation, and notification initialization, which are independent of the user type, are also included in the user interface layer.

3.2.2 Web Server Layer

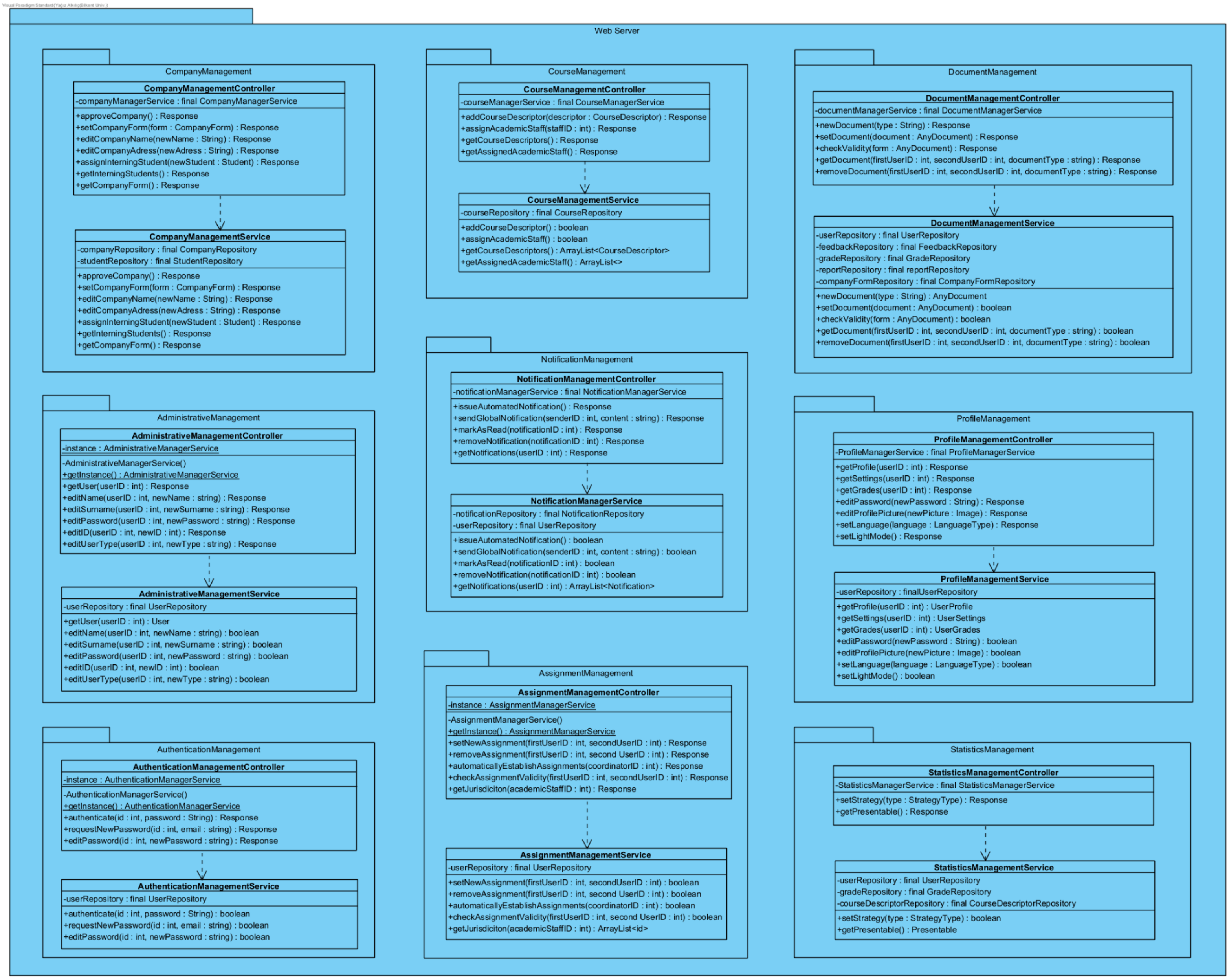
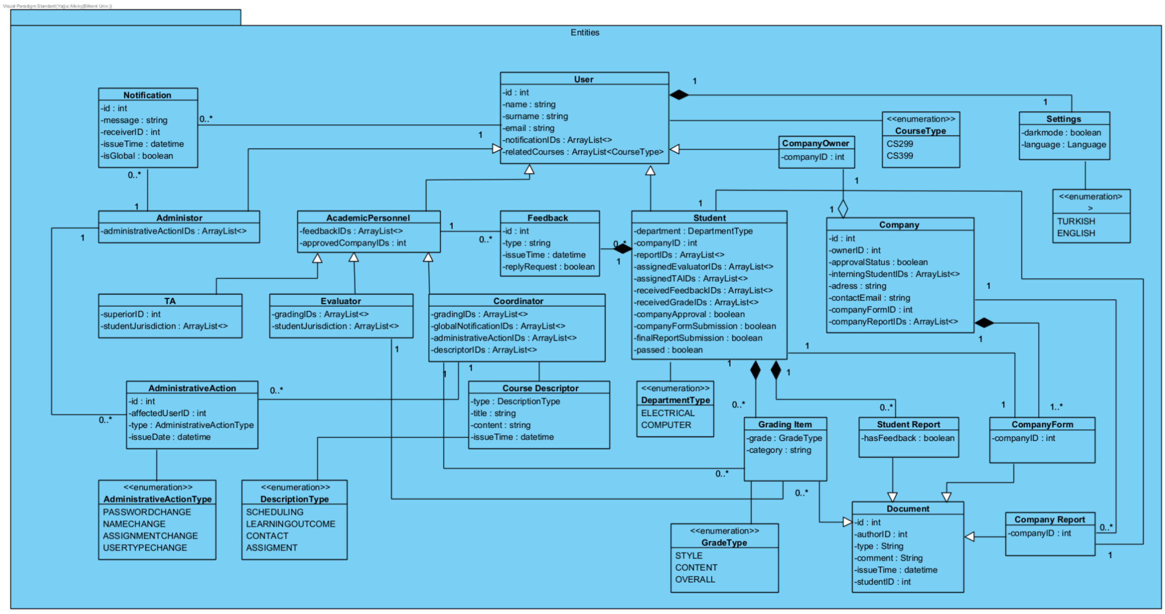

Visual Paradigm Standard(Yağız Alkılıç(Bilkent Univ.))
Web Server
CompanyManagement
CompanyManagementController
-companyManagerService : final CompanyManagerService
+approveCompany() : Response
+setCompanyForm(form : CompanyForm) : Response
+editCompanyName(newName : String) : Response
+editCompanyAdress(newAdress : String) : Response
+assignInterningStudent(newStudent : Student) : Response
+getInterningStudents() : Response
+getCompanyForm() : Response
CompanyManagementService
-companyRepository : final CompanyRepository
-studentRepository : final StudentRepository
+approveCompany() : Response
+setCompanyForm(form : CompanyForm) : Response
+editCompanyName(newName : String) : Response
+editCompanyAdress(newAdress : String) : Response
+assignInterningStudent(newStudent : Student) : Response
+getInterningStudents() : Response
+getCompanyForm() : Response
AdministrativeManagement
AdministrativeManagementController
-instance : AdministrativeManagerService
-AdministrativeManagerService()
+getInstance() : AdministrativeManagerService
+getUser(userID : int) : Response
+editName(userID : int, newName : string) : Response
+editSurname(userID : int, newSurname : string) : Response
+editPassword(userID : int, newPassword : string) : Response
+editID(userID : int, newID : int) : Response
+editUserType(userID : int, newType : string) : Response
AdministrativeManagementService
-userRepository : final UserRepository
+getUser(userID : int) : User
+editName(userID : int, newName : string) : boolean
+editSurname(userID : int, newSurname : string) : boolean
+editPassword(userID : int, newPassword : string) : boolean
+editID(userID : int, newID : int) : boolean
+editUserType(userID : int, newType : string) : boolean
AuthenticationManagement
AuthenticationManagementController
-instance : AuthenticationManagerService
-AuthenticationManagerService()
+getInstance() : AuthenticationManagerService
+authenticate(id : int, password : String) : Response
+requestNewPassword(id : int, email : string) : Response
+editPassword(id : int, newPassword : string) : Response
AuthenticationManagementService
-userRepository : final UserRepository
+authenticate(id : int, password : String) : boolean
+requestNewPassword(id : int, email : string) : boolean
+editPassword(id : int, newPassword : string) : boolean
CourseManagement
CourseManagementController
-courseManagerService : final CourseManagerService
+addCourseDescriptor(descriptor : CourseDescriptor) : Response
+assignAcademicStaff(staffID : int) : Response
+getCourseDescriptors() : Response
+getAssignedAcademicStaff() : Response
CourseManagementService
-courseRepository : final CourseRepository
+addCourseDescriptor() : boolean
+assignAcademicStaff() : boolean
+getCourseDescriptors() : ArrayList<CourseDescriptor>
+getAssignedAcademicStaff() : ArrayList<>
NotificationManagement
NotificationManagementController
-notificationManagerService : final NotificationManagerService
+issueAutomatedNotification() : Response
+sendGlobalNotification(senderID : int, content : string) : Response
+markAsRead(notificationID : int) : Response
+removeNotification(notificationID : int) : Response
+getNotifications(userID : int) : Response
NotificationManagerService
-notificationRepository : final NotificationRepository
-userRepository : final UserRepository
+issueAutomatedNotification() : boolean
+sendGlobalNotification(senderID : int, content : string) : boolean
+markAsRead(notificationID : int) : boolean
+removeNotification(notificationID : int) : boolean
+getNotifications(userID : int) : ArrayList<Notification>
AssignmentManagement
AssignmentManagementController
-instance : AssignmentManagerService
-AssignmentManagerService()
+getInstance() : AssignmentManagerService
+setNewAssignment(firstUserID : int, secondUserID : int) : Response
+removeAssignment(firstUserID : int, second UserID : int) : Response
+automaticallyEstablishAssignments(coordinatorID : int) : Response
+checkAssignmentValidity(firstUserID : int, secondUserID : int) : Response
+getJurisdiciton(academicStaffID : int) : Response
AssignmentManagementService
-userRepository : final UserRepository
+setNewAssignment(firstUserID : int, secondUserID : int) : boolean
+removeAssignment(firstUserID : int, second UserID : int) : boolean
+automaticallyEstablishAssignments(coordinatorID : int) : boolean
+checkAssignmentValidity(firstUserID : int, second UserID : int) : boolean
+getJurisdiciton(academicStaffID : int) : ArrayList<id>
DocumentManagement
DocumentManagementController
-documentManagerService : final DocumentManagerService
+newDocument(type : String) : Response
+setDocument(document : AnyDocument) : Response
+checkValidity(form : AnyDocument) : Response
+getDocument(firstUserID : int, secondUserID : int, documentType : string) : Response
+removeDocument(firstUserID : int, secondUserID : int, documentType : string) : Response
DocumentManagementService
-userRepository : final UserRepository
-feedbackRepository : final FeedbackRepository
-gradeRepository : final GradeRepository
-reportRepository : final reportRepository
-companyFormRepository : final CompanyFormRepository
+newDocument(type : String) : AnyDocument
+setDocument(document : AnyDocument) : boolean
+checkValidity(form : AnyDocument) : boolean
+getDocument(firstUserID : int, secondUserID : int, documentType : string) : boolean
+removeDocument(firstUserID : int, secondUserID : int, documentType : string) : boolean
ProfileManagement
ProfileManagementController
-ProfileManagerService : final ProfileManagerService
+getProfile(userID : int) : Response
+getSettings(userID : int) : Response
+getGrades(userID : int) : Response
+editPassword(newPassword : String) : Response
+editProfilePicture(newPicture : Image) : Response
+setLanguage(language : LanguageType) : Response
+setLightMode() : Response
ProfileManagementService
-userRepository : finalUserRepository
+getProfile(userID : int) : UserProfile
+getSettings(userID : int) : UserSettings
+getGrades(userID : int) : UserGrades
+editPassword(newPassword : String) : boolean
+editProfilePicture(newPicture : Image) : boolean
+setLanguage(language : LanguageType) : boolean
+setLightMode() : boolean
StatisticsManagement
StatisticsManagementController
-StatisticsManagerService : final StatisticsManagerService
+setStrategy(type : StrategyType) : Response
+getPresentable() : Response
StatisticsManagementService
-userRepository : final UserRepository
-gradeRepository : final GradeRepository
-courseDescriptorRepository : final CourseDescriptorRepository
+setStrategy(type : StrategyType) : boolean
+getPresentable() : Presentable

Figure 6. Web Server Management Layer (High quality version: https://github.com/zedyjy/CS319-Project/blob/main/Diagrams/Web%20Server.png)

The web server layer consists of control objects that manage the basic functions of the internship system. These objects contain the functions that will ensure that the main functioning of the system continues in a complete manner.

3.2.3 Data Management Layer



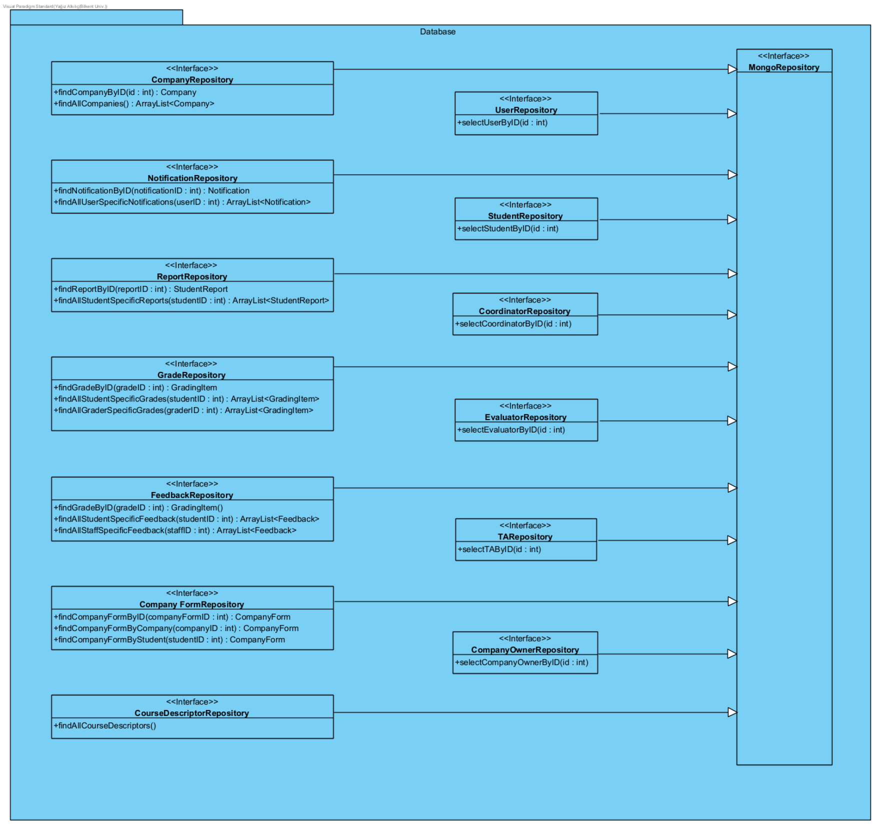

Visual Paradigm Standard(Yağız Alkılıç(Bilkent Univ.))
Database
<<Interface>>
CompanyRepository
+findCompanyByID(id : int) : Company
+findAllCompanies() : ArrayList<Company>
<<Interface>>
UserRepository
+selectUserByID(id : int)
<<Interface>>
MongoRepository
<<Interface>>
NotificationRepository
+findNotificationByID(notificationID : int) : Notification
+findAllUserSpecificNotifications(userID : int) : ArrayList<Notification>
<<Interface>>
StudentRepository
+selectStudentByID(id : int)
<<Interface>>
ReportRepository
+findReportByID(reportID : int) : StudentReport
+findAllStudentSpecificReports(studentID : int) : ArrayList<StudentReport>
<<Interface>>
CoordinatorRepository
+selectCoordinatorByID(id : int)
<<Interface>>
GradeRepository
+findGradeByID(gradeID : int) : GradingItem
+findAllStudentSpecificGrades(studentID : int) : ArrayList<GradingItem>
+findAllGraderSpecificGrades(graderID : int) : ArrayList<GradingItem>
<<Interface>>
EvaluatorRepository
+selectEvaluatorByID(id : int)
<<Interface>>
FeedbackRepository
+findGradeByID(gradeID : int) : GradingItem()
+findAllStudentSpecificFeedback(studentID : int) : ArrayList<Feedback>
+findAllStaffSpecificFeedback(staffID : int) : ArrayList<Feedback>
<<Interface>>
TARepository
+selectTAByID(id : int)
<<Interface>>
Company FormRepository
+findCompanyFormByID(companyFormID : int) : CompanyForm
+findCompanyFormByCompany(companyID : int) : CompanyForm
+findCompanyFormByStudent(studentID : int) : CompanyForm
<<Interface>>
CompanyOwnerRepository
+selectCompanyOwnerByID(id : int)
<<Interface>>
CourseDescriptorRepository
+findAllCourseDescriptors()

Figure 7 and 8. Data Management Layer (High quality version of entities: https://github.com/zedyjy/CS319-Project/blob/main/Diagrams/Entities.png, high quality version of database: https://github.com/zedyjy/CS319-Project/blob/main/Diagrams/Database.png)

The data layer consists only of the database, the database is our main information storage required for the sustainability and usability of the system. Since it contains the necessary information for the system, in some cases, data needs to be retrieved from the database, and in some cases, data needs to be uploaded therefore, it communicates with web service entities with the help of the database manager. In the database layer, we can observe the interfaces of the data that we intend to store in the database during the use of the application and modify and delete it when necessary. The basic attributes of the entities required for the use of the web application and the communication of these entity types with each other are shown in detail in the entities package.

3.3 Packages

3.3.1 Internal Packages

Model:

- **Report Manager Package:** For handling student reports, grading/revisions, and requests from instructors and TAs.
    - **Feedback Package:** Handles all feedback-related tasks. (Uploading PDF, Text inputs, etc.)
    - **Student Report Package:** Handles all Student Report tasks.
- **User Package:** For managing user accounts, authentication, and authorization.
- **Internship Package:** For handling intern applications and company documents.
- **Coordinator Package:** For assigning TAs to instructors or students to instructors.
- **Forms Manager Package:** For managing submissions, grading or reports, etc.

View:

- **HTML/CSS/JS:** For creating the user interface of the application.
- **User Screens Package:** Handles all user screens.
    - **Student Screen Package:** Handles all user screens.
    - **Instructor Screen Package:** Handles all Instructor Screens.
    - **TA Screen Package:** Handles all TA Screens.
    - **Coordinator Screen Package:** Handles all Coordinator Screens.
    - **Admin Screen Package:** Handles all Admin Screens.
- **Notifications Screen Package:** Handles all notification screens. (Including email notification templates etc.)

Controller:

- **Router package:** For handling incoming HTTP requests and directing them to the appropriate controller method.

- **Internal Middleware package:** For performing common tasks like authentication and authorization before executing controller methods. And other relevant repetitive low stake actions.
- **STARS Package:** For getting all stakeholder data.
- **Notification Manager Package:** For managing all types of notifications (email, pop-ups, etc.)
- **Authentication Manager Package:** For handling all authentication processes like file type authentication, login, and register data input.
- **Database Package**: For deploying the database and starting the database server.
    - **MongoDB Package:** For Starting a MongoDB server and deploying a database.
- **Database Manager Package**: For interacting with the database and managing model relationships.
    - **MongoDB Driver Package:** For connecting to the MongoDB database and querying data.
- **Email Sender Package:** Handles all email-sending processes. (For Notifications, Report Submissions, Revision Requests, etc.)

3.3.2 External Packages

- **Routing package:** For defining application routes and handling incoming HTTP requests.
- **Authentication package**: For handling user authentication and authorization.
- **External Middleware package**: For performing common tasks like authentication and authorization before executing controller methods. And other relevant repetitive low stake actions.
- **STARS API**: For getting/setting grades, having communication with Moodle, or for other functionalities that we might add in the future.
- **TCPDF:** For generating PDF reports for student submissions/ or for any other PDF-related task.
- **Bootstrap**: For styling the user interface of the application and making it responsive.
- **jQuery:** For handling user interactions and making asynchronous requests to the server.
    - **Fetch API:** A native JavaScript API for making HTTP requests that are supported in modern browsers.
- **SweetAlert2:** A JS library for pop-up notification handling.
- **bcrypt.js:** For hashing password inputs and making other relevant data secure. Uses the BCRYPT crypt algorithm.
- **npm:** For handling external packages.
- **cors:** Cross-Origin Resource Sharing (CORS) package that enables secure communication between different domains in a web application.
- **ejs:** Embedded JavaScript templates package that allows you to embed JavaScript code within HTML templates to generate dynamic web content.
- **express:** A fast and minimalist web application framework for Node.js that simplifies the process of building robust and scalable web applications.

- **fs:** File system package that provides methods to interact with the file system, allowing reading, writing, and manipulating files.
- **http**: A package that provides a set of HTTP utility methods for creating HTTP servers, making HTTP requests, and handling HTTP responses.
- **mongodb:** The official MongoDB driver for Node.js that allows interaction with MongoDB databases.
- **mongoose:** An object data modeling (ODM) library for MongoDB and Node.js that provides a higher-level abstraction for interacting with MongoDB.
- **path:** A package that provides utilities for working with file and directory paths, resolving and manipulating path strings, and handling file extensions.

3.4 Class Interfaces

3.4.1 Entities

1. AcademicPersonnel

1.1. Attributes Summary

Name	Description
approvedCompanyIDs	
feedbackIDs	

2. Administor

2.1. Attributes Summary

Name	Description
administrativeActionIDs	

3. AdministrativeAction

3.1. Attributes Summary

Name	Description
affectedUserID	
id	
issueDate	
type	

4. AdministrativeActionType

4.1. Attributes Summary

Name	Description
ASSIGNMENTCHANGE	Removing an appointment from a student or appointing an academic personnel to a student.
NAMECHANGE	
PASSWORDCHANGE	
USERTYPECHANGE	

5. Company

5.1. Attributes Summary

Name	Description
adress	
approvalStatus	
companyFormID	Company form is associated with the company only after it is approved. Therefore, there is a need for only one company form. If more than one student applies for the company after one form is approved all other forms are discarded.
companyReportIDs	
contactEmail	
id	
interningStudentIDs	
ownerID	

6. Company Report

6.1. Attributes Summary

Name	Description
authorID	
grade	For the company's evaluation of the student. The report will still need to be evaluated by academic personnel so that the grade is approved.
id	
issueTime	
studentID	

7. CompanyForm

7.1. Attributes Summary

Name	Description
authorID	
companyID	
id	
issueTime	
submittorID	

8. CompanyOwner

8.1. Attributes Summary

Name	Description
companyID	

9. Coordinator

9.1. Attributes Summary

Name	Description
administrativeActionIDs	
descriptorIDs	
globalNotificationIDs	
gradingIDs	

10. Course Description

10.1. Attributes Summary

Name	Description
content	Handles course-wide changes and announcements. These could be deadlines, report specifications, academic personnel changes, and so on.
issueTime	
title	
type	

11. CourseType

11.1. Attributes Summary

Name	Description
CS299	
CS399	

12. DepartmentType

12.1. Attributes Summary

Name	Description
COMPUTER	
ELECTRICAL	

13. DescriptionType

13.1. Attributes Summary

Name	Description
ASSIGMENT	
CONTACT	
LEARNINGOUTCOME	
SCHEDULING	

14. Evaluator

14.1. Attributes Summary

Name	Description
gradingIDs	
studentJurisdiction	Students that the evaluator is responsible for.

15. Feedback

15.1. Attributes Summary

Name	Description
id	
issueTime	
replyRequest	
type	

16. GradeType

16.1. Attributes Summary

Name	Description
COMPANYEVALUATION	
CONTENT	
OVERALL	

Name	Description
STYLE	

17. Grading Item

17.1. Attributes Summary

Name	Description
category	
grade	
id	

18. Language

18.1. Attributes Summary

Name	Description
ENGLISH	
TURKISH	

19. Notification

19.1. Attributes Summary

Name	Description
id	
isGlobal	Describes if the notification is only for one user or if it is for every user.
issueTime	
message	
receiverID	

20. Settings

20.1. Attributes Summary

Name	Description
darkmode	
language	

21. Student

21.1. Attributes Summary

Name	Description
assignedEvaluatorIDs	

Name	Description
assignedTAIDs	
companyApproval	
companyFormSubmission	
companyID	
department	
finalReportSubmission	
passed	
receivedFeedbackIDs	
receivedGradeIDs	
reportIDs	

22. Student Report

22.1. Attributes Summary

Name	Description
authorID	
id	
issueTime	

23. TA

23.1. Attributes Summary

Name	Description
studentJurisdiction	Students that the TA is responsible for.
superiorID	

24. User

24.1. Attributes Summary

Name	Description
email	
id	
name	
notificationIDs	
relatedCourses	Is not applicable to coordinators, they are initialized as related to all courses.
surname	

3.4.2 Web Server

1. AdministrativeManagerController

1.1. Attributes Summary

Name	Description
administrativeManagerService	

1.2. Operations Summary

Name	Description
editID	
editName	
editPassword	
editSurname	
editUserType	
getUser	

2. AdministrativeManagerService

2.1. Attributes Summary

Name	Description
userRepository	

2.2. Operations Summary

Name	Description
editID	
editName	
editPassword	
editSurname	
editUserType	
getUser	

3. AssignmentManagerController

3.1. Attributes Summary

Name	Description
assignmentManagerService	

3.2. Operations Summary

Name	Description
automaticallyEstablishAssignments	
checkAssignmentValidity	Checks if such assignment can be made. Students cannot be assigned to students or a user cannot be assigned to another user multiple times.
getJurisdiciton	
removeAssignment	
setNewAssignment	

4. AssignmentManagerService

4.1. Attributes Summary

Name	Description
userRepository	

4.2. Operations Summary

Name	Description
automaticallyEstablishAssignments	
checkAssignmentValidity	
getJurisdiciton	
removeAssignment	
setNewAssignment	

5. AuthenticationManagerController

5.1. Attributes Summary

Name	Description
authenticationManagerService	

5.2. Operations Summary

Name	Description
authenticate	
editPassword	
requestNewPassword	

6. AuthenticationManagerService

6.1. Attributes Summary

Name	Description
userRepository	

6.2. Operations Summary

Name	Description
authenticate	
editPassword	
requestNewPassword	

7. CompanyManagerController

7.1. Attributes Summary

Name	Description
companyManagerService	

7.2. Operations Summary

Name	Description
approveCompany	
assignInterningStudent	
editCompanyAdress	
editCompanyName	
getCompanyForm	
getInterningStudents	
setCompanyForm	

8. CompanyManagerService

8.1. Attributes Summary

Name	Description
companyRepository	
studentRepository	

8.2. Operations Summary

Name	Description
approveCompany	
assignInterningStudent	
editCompanyAdress	
editCompanyName	
getCompanyForm	
getInterningStudents	
setCompanyForm	

9. CourseManagerController

9.1. Attributes Summary

Name	Description
courseManagerService	

9.2. Operations Summary

Name	Description
addCourseDescriptor	
assignAcademicStaff	
getAssignedAcademicStaff	
getCourseDescriptors	

10. CourseManagerService

10.1. Attributes Summary

Name	Description
courseRepository	

10.2. Operations Summary

Name	Description
addCourseDescriptor	
assignAcademicStaff	
getAssignedAcademicStaff	
getCourseDescriptors	

11. FormManagerController

11.1. Attributes Summary

Name	Description
formManagerService	

11.2. Operations Summary

Name	Description
checkValidity	
getForm	
receiveFeedback	
receiveGrade	
removeForm	
sendReevaluationRequest	
setFeedback	
setGrade	
setStudentReport	

12. FormManagerService

12.1. Attributes Summary

Name	Description
feedbackRepository	
gradeRepository	
userRepository	

12.2. Operations Summary

Name	Description
checkValidity	Checks if the forms are in required standards. Checks if the sections are empty or missing information, and so on.
getForm	
receiveFeedback	
receiveGrade	
removeForm	
sendReevaluationRequest	
setFeedback	
setGrade	

13. NotificationManagerController

13.1. Attributes Summary

Name	Description
notificationManagerService	

13.2. Operations Summary

Name	Description
getNotifications	
issueAutomatedNotification	Automated notifications are generated by the system in response to events. These could be close deadlines, received revision requests, or received grading requests.
markAsRead	
removeNotification	
sendGlobalNotification	

14. NotificationManagerService

14.1. Attributes Summary

Name	Description
notificationRepository	
userRepository	

14.2. Operations Summary

Name	Description
getNotifications	
issueAutomatedNotification	
markAsRead	
removeNotification	
sendGlobalNotification	

15. ReportManagerController

15.1. Attributes Summary

Name	Description
reportManagerService	

15.2. Operations Summary

Name	Description
checkValidity	
receiveStudentReport	
removeForm	

Name	Description
setStudentReport	

16. ReportManagerService

16.1. Attributes Summary

Name	Description
studentReportRepository	
userRepository	

16.2. Operations Summary

Name	Description
checkValidity	Checks if the report is within provided standards.
receiveStudentReport	
removeForm	
setStudentReport	

3.4.3 User Interface

1. DescriptionsPage

1.1. Operations Summary

Name	Description
addDescriptor	
getCourseDescriptors	

2. EditUserPage

2.1. Operations Summary

Name	Description
changeAsignment	
changeName	
changePassword	
changeUserType	

3. FeedbackSubmissionPage

3.1. Operations Summary

Name	Description
giveFeedback	

4. ForgotPasswordPage

4.1. Operations Summary

Name	Description
goToLoginPage	
requestNewPassword	

5. GradeSubmissionPage

5.1. Operations Summary

Name	Description
giveGrade	

6. LoginPage

6.1. Operations Summary

Name	Description
goToForgotPassword	
login	

7. NotificationItem

7.1. Operations Summary

Name	Description
getNotificationContent	

8. NotificationsPage

8.1. Operations Summary

Name	Description
getNotifications	
showNotification	

9. ProfilePage

9.1. Operations Summary

Name	Description
editPassword	

Name	Description
editProfilePicture	

10. ReportsPage

10.1. Operations Summary

Name	Description
editReport	
getReports	
goToReportSubmissionPage	
goToViewGradePage	

11. ReportSubmissionPage

11.1. Operations Summary

Name	Description
submitReport	

12. SearchUserPage

12.1. Operations Summary

Name	Description
goToEditUserPage	
searchUser	

13. SettingsPage

13.1. Operations Summary

Name	Description
changeDarkLightMode	
changeLanguage	
getSettings	

14. SideBar

14.1. Operations Summary

Name	Description
goToCourseDescriptionsPage	
goToNotificationListPage	
goToReportsPage	
goToStaffInfoPage	
goToStudentsPage	
goToUserInitializationPage	

15. StaffInfoPage

15.1. Operations Summary

Name	Description
getStaff	
getStaffContact	

16. StudentsPage

16.1. Operations Summary

Name	Description
getStudents	
goToCompantFormPage	
goToFeedbackSubmissionPage	
goToGradeSubmissionPage	
goToViewReportPage	

17. TopBar

17.1. Operations Summary

Name	Description
goToProfilePage	

Name	Description
goToSettingsPage	
logout	

18. UserInitializationPage

18.1. Operations Summary

Name	Description
goToDescriptionsPage	
goToNotificationsPage	
goToProfilePage	

19. ViewCompanyFormPage

19.1. Operations Summary

Name	Description
approve	
getForm	

20. ViewGradePage

20.1. Operations Summary

Name	Description
getGrades	
requestRevision	

21. ViewReportPage

21.1. Operations Summary

Name	Description
getReport	
goToFeedbackSubmissionPage	
goToGradeSubmissionPage	

3.5 Design Patterns

3.5.1 Singleton Pattern

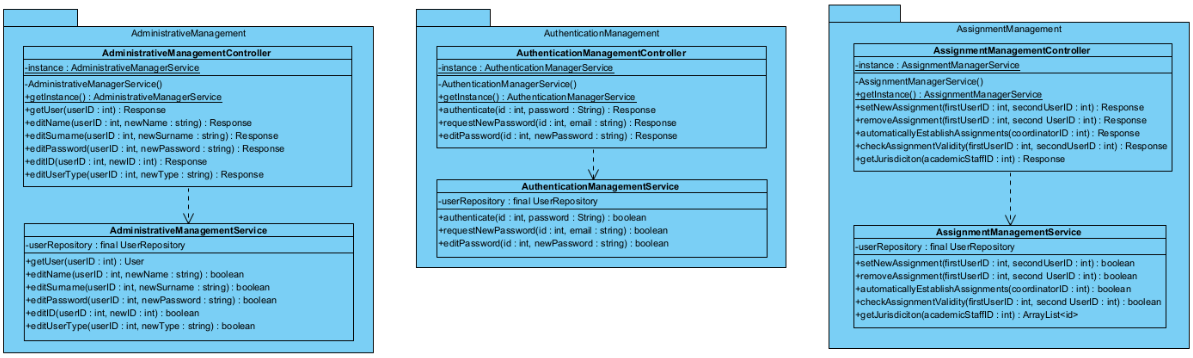

AdministrativeManagement
AdministrativeManagementController
-instance : AdministrativeManagerService
-AdministrativeManagerService()
+getInstance() : AdministrativeManagerService
+getUser(userID : int) : Response
+editName(userID : int, newName : string) : Response
+editSurname(userID : int, newSurname : string) : Response
+editPassword(userID : int, newPassword : string) : Response
+editID(userID : int, newID : int) : Response
+editUserType(userID : int, newType : string) : Response
AdministrativeManagementService
-userRepository : final UserRepository
+getUser(userID : int) : User
+editName(userID : int, newName : string) : boolean
+editSurname(userID : int, newSurname : string) : boolean
+editPassword(userID : int, newPassword : string) : boolean
+editID(userID : int, newID : int) : boolean
+editUserType(userID : int, newType : string) : boolean
AuthenticationManagement
AuthenticationManagementController
-instance : AuthenticationManagerService
-AuthenticationManagerService()
+getInstance() : AuthenticationManagerService
+authenticate(id : int, password : String) : Response
+requestNewPassword(id : int, email : string) : Response
+editPassword(id : int, newPassword : string) : Response
AuthenticationManagementService
-userRepository : final UserRepository
+authenticate(id : int, password : String) : boolean
+requestNewPassword(id : int, email : string) : boolean
+editPassword(id : int, newPassword : string) : boolean
AssignmentManagement
AssignmentManagementController
-instance : AssignmentManagerService
-AssignmentManagerService()
+getInstance() : AssignmentManagerService
+setNewAssignment(firstUserID : int, secondUserID : int) : Response
+removeAssignment(firstUserID : int, second UserID : int) : Response
+automaticallyEstablishAssignments(coordinatorID : int) : Response
+checkAssignmentValidity(firstUserID : int, secondUserID : int) : Response
+getJurisdiciton(academicStaffID : int) : Response
AssignmentManagementService
-userRepository : final UserRepository
+setNewAssignment(firstUserID : int, secondUserID : int) : boolean
+removeAssignment(firstUserID : int, second UserID : int) : boolean
+automaticallyEstablishAssignments(coordinatorID : int) : boolean
+checkAssignmentValidity(firstUserID : int, second UserID : int) : boolean
+getJurisdiciton(academicStaffID : int) : ArrayList<id>

Figure 9: Singleton pattern usage in UML diagram

The Singleton Pattern ensures that only one instance of a class is created and provides a global point of access to that instance.

Several controllers in the system if used simultaneously by different instances can cause corruption and unwanted behavior in the database. In the context of our system which contains the controllers: Administrative Management, Authentication Management, and Assignment Management; concurrent usage by different instances can potentially cause conflicts and inconsistencies in the database. This is because multiple instances may try to access and modify the same data simultaneously, leading to data corruption or incorrect results. Using the Singleton pattern, we can ensure that there is only one instance of the database connection object, which can be shared across different parts of the application. This allows for sequential access to the database and makes sure that conflicting operations do not occur simultaneously.

3.5.2 Factory Method Pattern

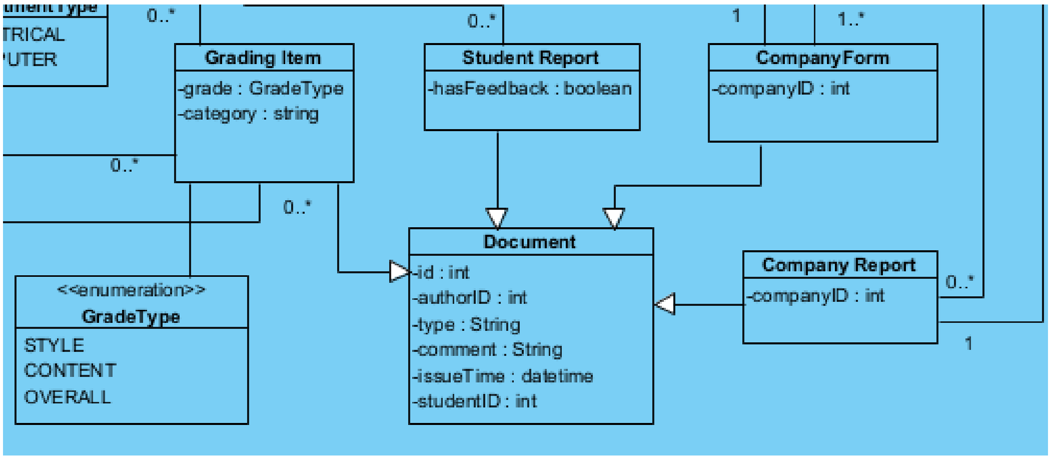


Figure 10: Factory pattern usage in UML diagram

The Factory Method Pattern defines an interface for creating objects, but lets subclasses decide which class to instantiate.

There are many cases in the app where we are required to use already-made classes which are inherited by more children classes that modify the functionality of the parent class given their requirements and needs. For example, the Document class functionality is being inherited by all document types as they all have similar items such as id, authorID, issueTime, etc. The controller simply needs to call the document creation function and give a document type as a variable. This allows for easy extension and addition of new document types in the future and the client code only needs to know how to use the factory method. In summary, we are implementing this pattern as we do have abstract classes that define common attributes for all subclasses.

3.5.3 Façade Pattern

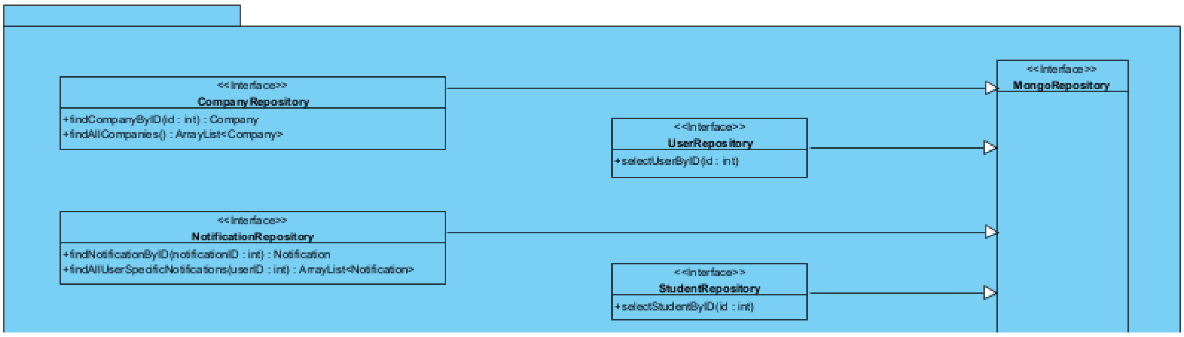

<<Interface>>
CompanyRepository
+findCompanyByID(id : int) : Company
+findAllCompanies() : ArrayList<Company>
<<Interface>>
UserRepository
+selectUserByID(id : int)
<<Interface>>
NotificationRepository
+findNotificationByID(notificationID : int) : Notification
+findAllUserSpecificNotifications(userID : int) : ArrayList<Notification>
<<Interface>>
StudentRepository
+selectStudentByID(id : int)
<<Interface>>
MongoRepository

Figure 11: Façade pattern usage in UML diagram

The Facade Pattern provides a simplified interface that encapsulates the complexity of a subsystem and provides a unified interface. It hides the complexities and interactions of the underlying subsystem, making it easier to interact with the system.

In our system, we have a database with multiple entities, and we use the Facade Pattern to create a simplified interface for accessing and managing these entities. The MongoRepository acts as a single entry point or a single repository that contains repositories for all entities. By using the Facade Pattern, we no longer need to interact directly with individual repositories for each entity. Instead, we can simply interact with the facade, which internally handles the interactions and operations.

3.5.4 Adapter Pattern

The Adapter Pattern allows objects with incompatible interfaces to work together by providing a common interface that can translate and adapt the calls between them.

This pattern will be implemented as a future option. As it is expected that the application may be integrated with SRS in the future where there is no need to have a separate system for student-related operations that also affect their grade. The process will be to first identify the interface of the SRS system, create an adapter class that implements our application’s interface, and wraps the SRS interface. Then, in the adapter class, we will create methods that translate the SRS interface class into our application’s interface calls, and update our current application’s code to communicate with the SRS.

5. Improvement Summary

The deployment diagram has been enhanced to be more comprehensive and inclusive, incorporating various termination and failure conditions. The initialization process has been expanded to provide a detailed account of how the system is initialized from start to finish, with a focus on the system becoming operational rather than user interaction. The termination process has been explained, outlining the steps to be taken for system termination. Additionally, the system's data storage capacity has been incorporated into its functionality. An overall opinion about the application has been included in the introduction. Pattern class diagrams have been added, while the observer pattern has been removed and the façade pattern has been introduced. All diagrams have been updated to accommodate statistics. The system decomposition has been redone. Furthermore, all instances of the old hardware/software mapping have been updated to align with the new project mapping, which includes HTML, CSS, JavaScript, Node.js, and MongoDB.

During the implementation of the application, several design decisions had to be modified based on our evolving needs, leading to valuable lessons learned throughout the process. In order to enhance usability, we made significant changes to the UI designs. Additionally, we switched our backend language from PHP to Node.js, recognizing its advantages and compatibility with the project requirements. To expedite development and streamline the process, we incorporated pre-developed npm packages that provided useful functionalities. These changes not only improved the efficiency of development but also enriched our understanding of the project's requirements and the importance of adaptability in software development.

6. Glossary

Internship: A temporary position within a company or organization where a student or trainee can gain work experience.

Report: A document that provides information or data about a particular topic.

Instructor: A person who teaches or trains students in a particular subject.

PDF: A file format used for documents that preserve the formatting and content of the original document.

Ghost Errors: Errors/Mistakes that seem small when developing a component and are mostly ignored in the case of junior developers, but keep piling up only to end up failing an app build in the future.

TA: Short for Teaching Assistant, a person who assists an instructor in teaching a course.

Coordinator: A person responsible for coordinating and organizing various activities or tasks.

Admin: Short for Administrator, a person with special privileges who can manage and control the app.

Application: A software program designed to perform a specific task or function.

Student: A person who is enrolled in an educational institution or program.

Company: An organization or business entity that provides products or services.

Upload: The process of transferring data or files from a local computer to a remote server.

Revision: The act of reviewing, editing, and improving a document or work.

Pre-approval form: A document that requests approval for a specific action or task before it is carried out.

Grading: The process of evaluating and assigning a grade or score to a student's work or performance.

MongoDB: A document-oriented database program that uses JSON-like documents with optional schemas.

SweetAlert2: A JS library for pop-up notification handling.

bcrypt.js: A JS Library for hashing password inputs and making other relevant data secure. Uses the BCRYPT crypt algorithm.

7. References

The #1 Development Tool Suite. Ideal Modeling &amp; Diagramming Tool for Agile Team Collaboration. (n.d.). Retrieved May 5, 2023, from https://www.visual-paradigm.com/

Documentation. Node.js. (n.d.). Retrieved May 5, 2023, from https://nodejs.org/en/docs

decode. (n.d.). decode/bcryptjs: Compatibility with the password_* functions that ship with Node.js. NPM. Retrieved May 5, 2023, from https://www.npmjs.com/package/bcryptjs

SWEETALERT2. SweetAlert2. (n.d.). Retrieved May 5, 2023, from https://sweetalert2.github.io/

NPM. npm. (n.d.) .Retrieved May 5, 2023, from https://www.npmjs.com/