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

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 a large number of users and data, 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 auditability 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

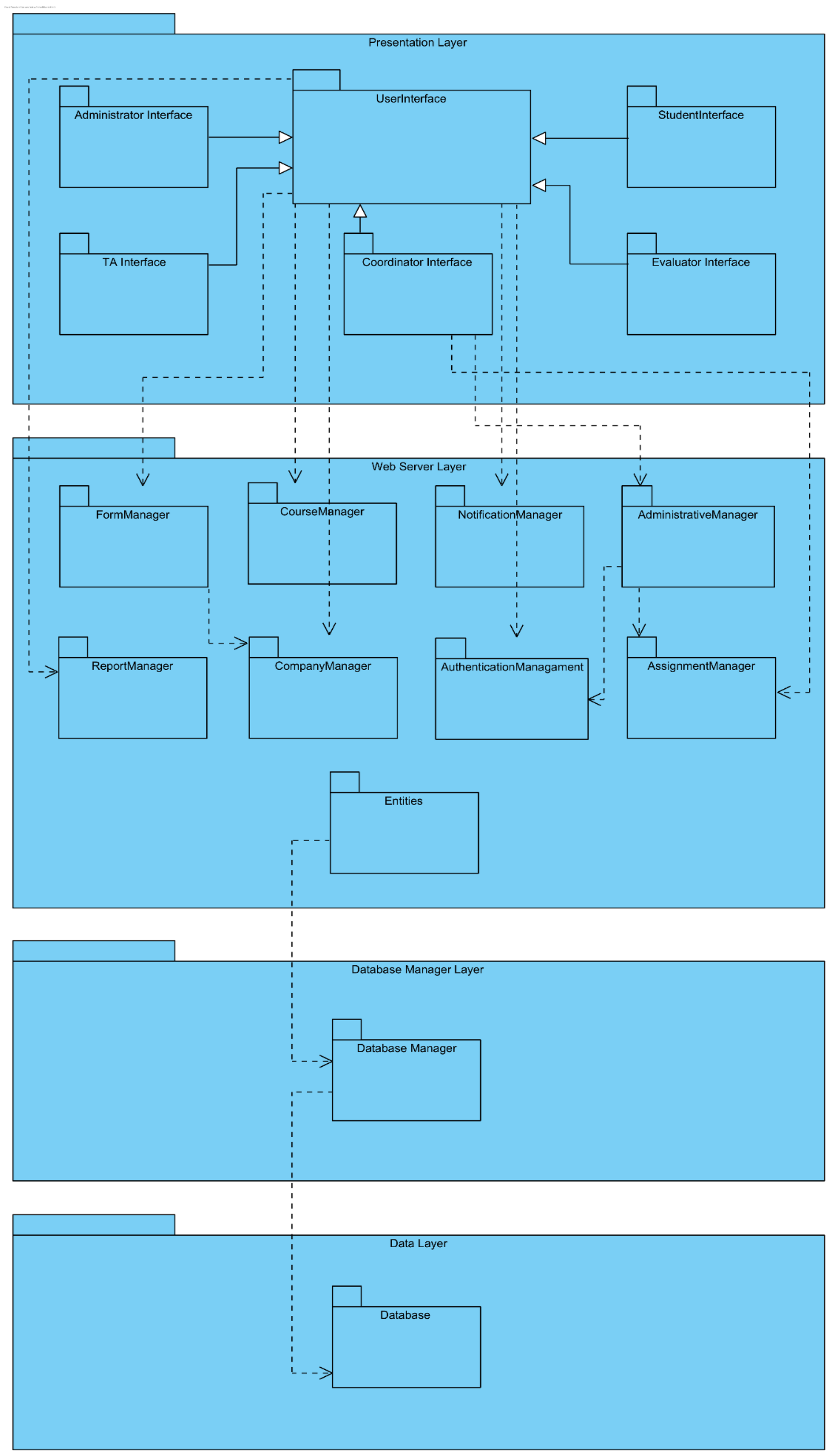


Figure 1. Subsystem Decomposition Diagram

We decided to work with four different layers in terms of codability and efficient use of our system, these are the Presentation layer, Web Server Layer, Database Manager 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 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.

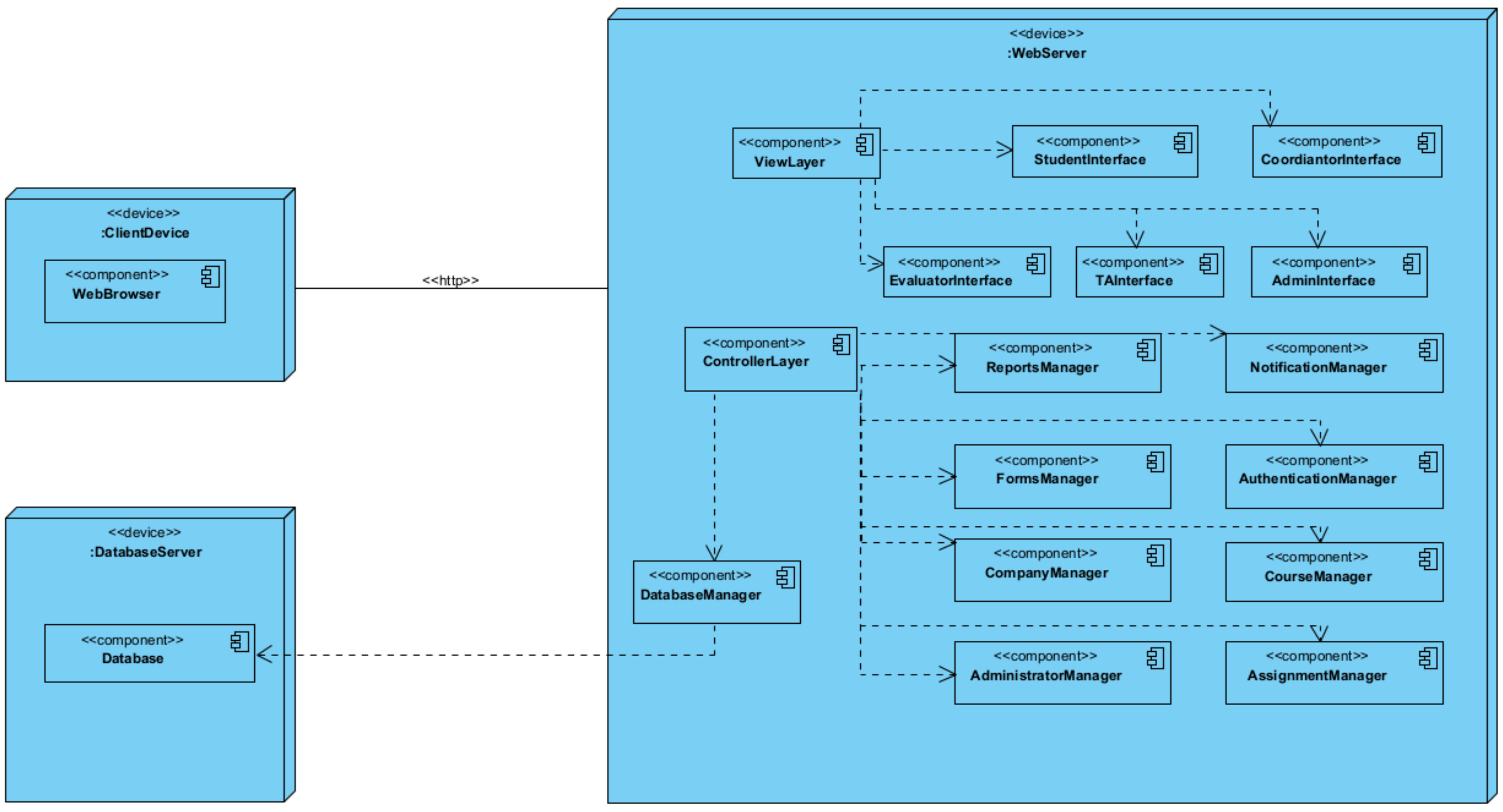


Figure 2. Deployment Diagram

2.3 Hardware/Software Mapping

We preferred to use PHP in the backend infrastructure of our application, the main reason for this is that it provides us with flexibility in terms of storage management of 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 in browsers. Apache and MySQL environments will be used to deploy the website and as a database, MongoDB will be preferred.

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.

2.4 Persistent Data Management

The use of a secure and efficient database is crucial for data security and 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 informations, uploaded reports, feedbacks and grades will be stored on 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 SRS system, securing information is fatal. Authentication will be conducted over HTTPS. HTTPS ensures data encryption between 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, teaching assistants (TA). To give an example, evaluators are only able to see the assigned students' reports. This system is same with TA's. Evaluators are not allowed to access other evaluator's student report and feedbacks. Same with students, a student can only access their own report and feedbacks. Seeing information about other student 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 little failures as possible. We intend to implement a recovery system that can jump back on its feet to get to the stable state in fastest way. Below, is the boundary conditions state diagram of the system we intend to implement.

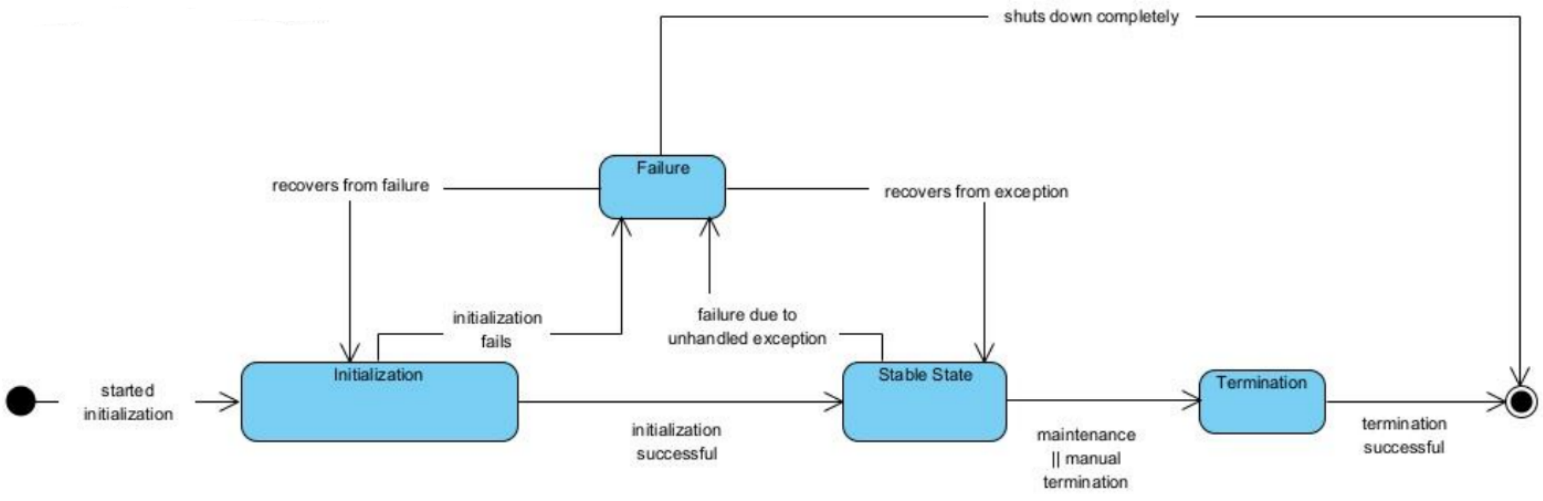


Figure 3. Boundary Conditions State Diagram

2.6.1 Initialization

Quaso internship application does not require any set up since it is a website. Users log in using their STARS id and password. The application is started by running the server from Apache. We use APACHE and mySQL environments to deploy, and load all other external dependencies on the server that we use. Once user logs in relevant data is fetched from the database to initialize the system. Users who do not have STARS id and password can reach certain types of information such as general information about courses and documents, announcements and contact information.

2.6.2 Termination

Possible terminations within the system will cause complete shutdown. This can be done by admin or unexpected operations such as security breaches or too much demand. We expect that there will be regular backups being taken of the server files and database, and in the case of a shut down, the system can be reloaded from a recent backup. We will be using a python script to take regular backups of server files using 'tar' to create gzip and other relevant backup types and store them on another machine. This is for server files. For the database, we will use MongoDB's 'mongodump' command to create regular DB backups too. This will also run with a python script. The last uploaded versions of files and information will be recovered. In the case of a shutdown user views a message that says: “System is not available for use at the moment. Please try in 10 minutes”.

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 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 compiled language, which generally makes it faster and more performant than interpreted languages like JavaScript. Due to the expected storage processes in terms of PDF files and profile images etc., we decided to go with PHP as a main backend language as we have more server-side logic to handle.

3.1.2 Developer Knowledge vs. Time-to-Market:

We could have used newer language frameworks for most of the logic as it would have made things a lot easier (Laravel, jQuery), because they come with added functionalities that would otherwise not be available in Vanilla languages. As most of the logic is done in Vanilla PHP and JavaScript, and the developers available did not have enough experience to successfully use the newer frameworks available to the market, we decided to go with Vanilla Languages for ‘most’ of the logic.

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.

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 is 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.

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.

3.1.6 Security vs. Performance

We could have used basic hashing techniques when user’s input their passwords, but considering the amount 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., so the same password is used for SRS too. Hence, we are going with **password_compat (PHP Library)** that 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.

3.2 Final Object Design

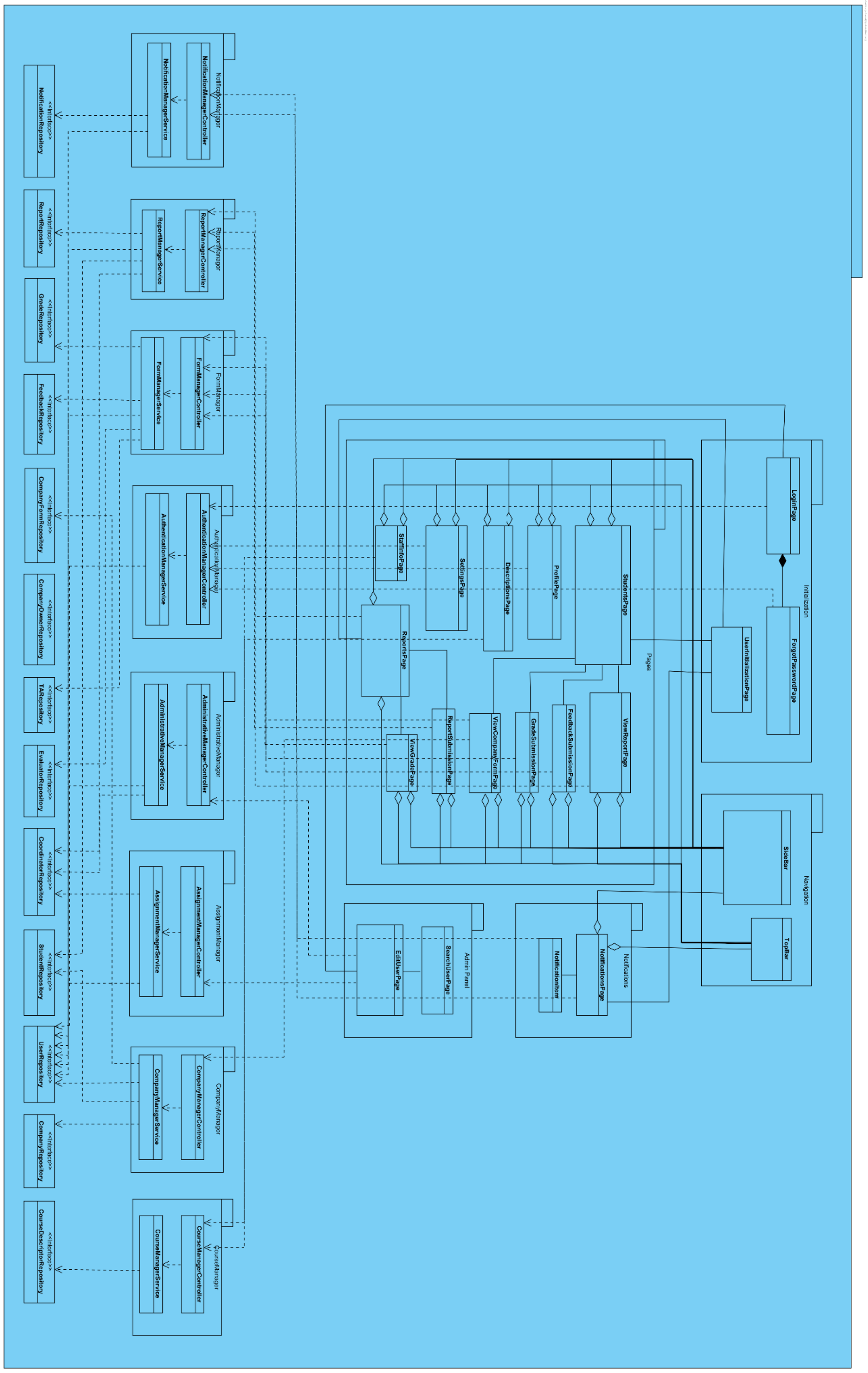


Figure 4. Final Object Design Diagram

3.2.1 User Interface Layer

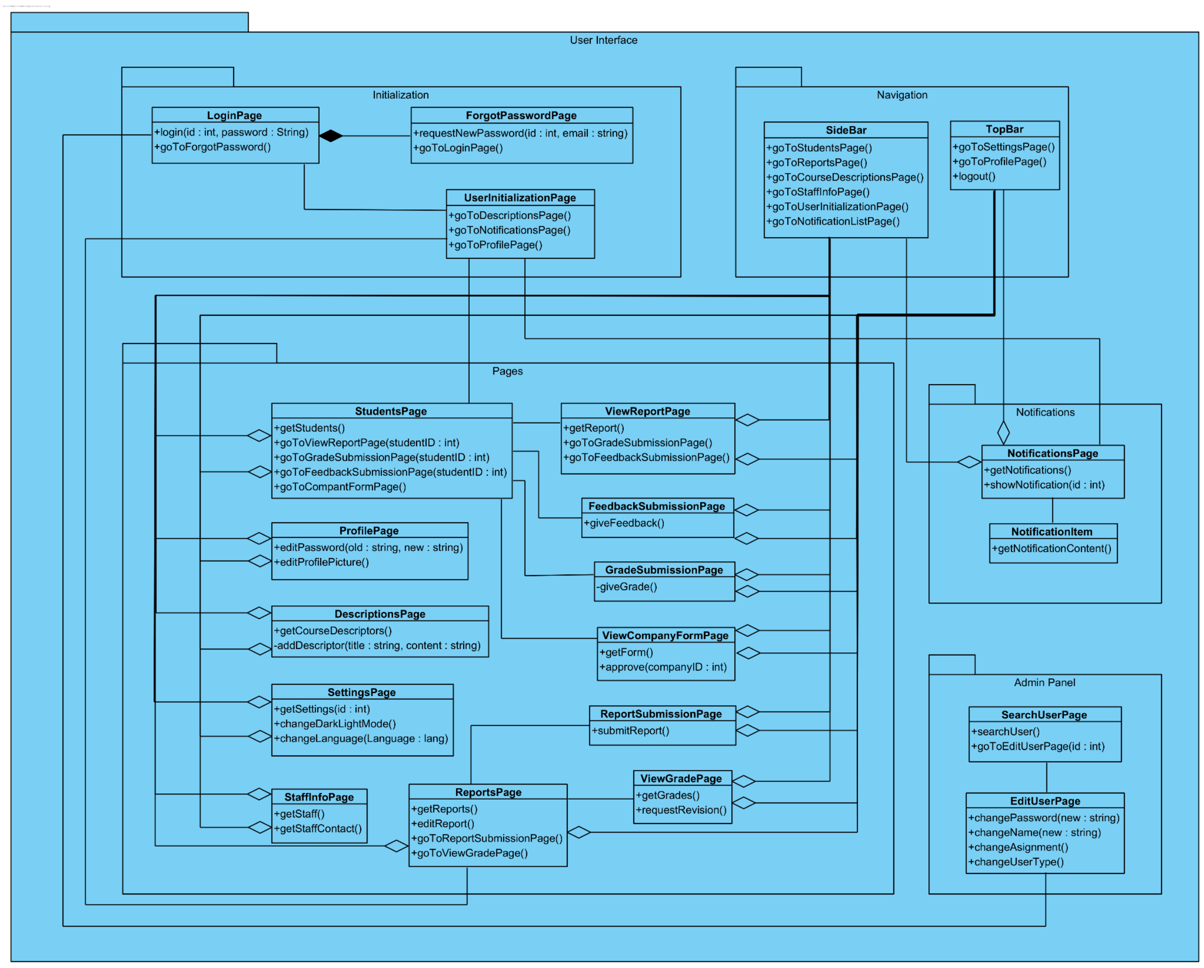


Figure 5. User Interface Management Layer

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

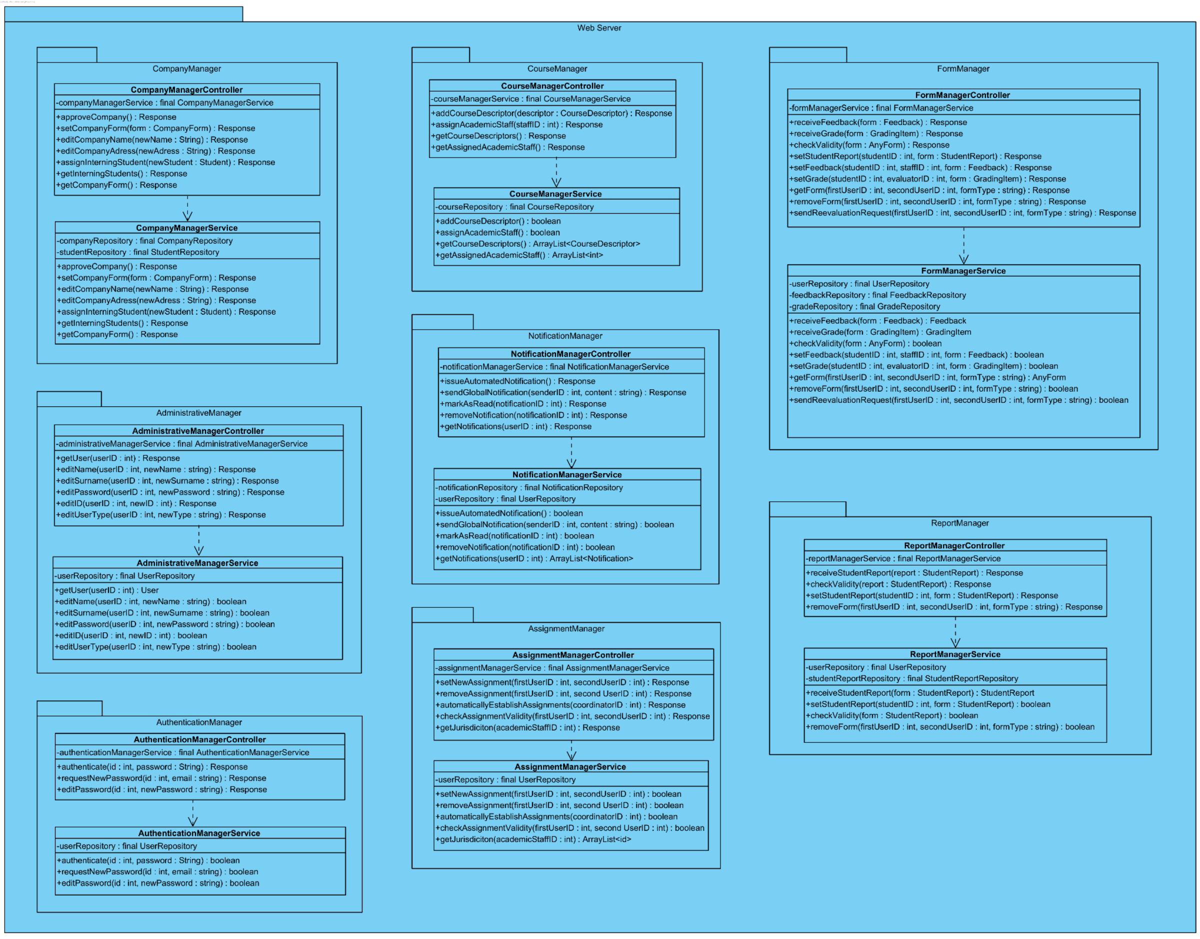
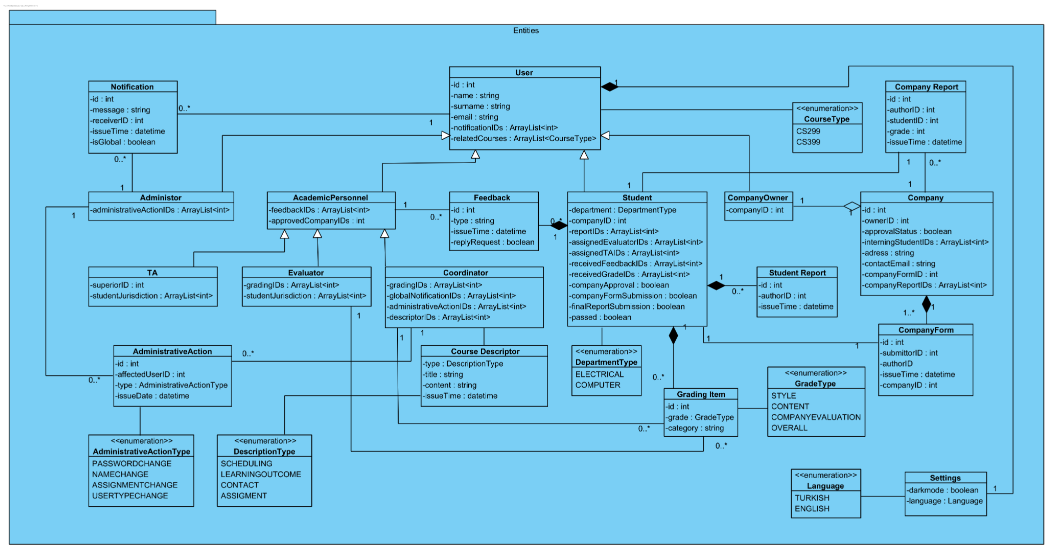

Web Server
CompanyManager
CompanyManagerController
-companyManagerService : final CompanyManagerService
+approveCompany() : Response
+setCompanyForm(form : CompanyForm) : Response
+editCompanyName(newName : String) : Response
+editCompanyAdress(newAdress : String) : Response
+assignInterningStudent(newStudent : Student) : Response
+getInterningStudents() : Response
+getCompanyForm() : Response
CompanyManagerService
-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
AdministrativeManager
AdministrativeManagerController
-administrativeManagerService : final 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
AdministrativeManagerService
-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
AuthenticationManager
AuthenticationManagerController
-authenticationManagerService : final AuthenticationManagerService
+authenticate(id : int, password : String) : Response
+requestNewPassword(id : int, email : string) : Response
+editPassword(id : int, newPassword : string) : Response
AuthenticationManagerService
-userRepository : final UserRepository
+authenticate(id : int, password : String) : boolean
+requestNewPassword(id : int, email : string) : boolean
+editPassword(id : int, newPassword : string) : boolean
CourseManager
CourseManagerController
-courseManagerService : final CourseManagerService
+addCourseDescriptor(descriptor : CourseDescriptor) : Response
+assignAcademicStaff(staffID : int) : Response
+getCourseDescriptors() : Response
+getAssignedAcademicStaff() : Response
CourseManagerService
-courseRepository : final CourseRepository
+addCourseDescriptor() : boolean
+assignAcademicStaff() : boolean
+getCourseDescriptors() : ArrayList<CourseDescriptor>
+getAssignedAcademicStaff() : ArrayList<int>
NotificationManager
NotificationManagerController
-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>
AssignmentManager
AssignmentManagerController
-assignmentManagerService : final 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
AssignmentManagerService
-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>
FormManager
FormManagerController
-formManagerService : final FormManagerService
+receiveFeedback(form : Feedback) : Response
+receiveGrade(form : GradingItem) : Response
+checkValidity(form : AnyForm) : Response
+setStudentReport(studentID : int, form : StudentReport) : Response
+setFeedback(studentID : int, staffID : int, form : Feedback) : Response
+setGrade(studentID : int, evaluatorID : int, form : GradingItem) : Response
+getForm(firstUserID : int, secondUserID : int, formType : string) : Response
+removeForm(firstUserID : int, secondUserID : int, formType : string) : Response
+sendReevaluationRequest(firstUserID : int, secondUserID : int, formType : string) : Response
FormManagerService
-userRepository : final UserRepository
-feedbackRepository : final FeedbackRepository
-gradeRepository : final GradeRepository
+receiveFeedback(form : Feedback) : Feedback
+receiveGrade(form : GradingItem) : GradingItem
+checkValidity(form : AnyForm) : boolean
+setFeedback(studentID : int, staffID : int, form : Feedback) : boolean
+setGrade(studentID : int, evaluatorID : int, form : GradingItem) : boolean
+getForm(firstUserID : int, secondUserID : int, formType : string) : AnyForm
+removeForm(firstUserID : int, secondUserID : int, formType : string) : boolean
+sendReevaluationRequest(firstUserID : int, secondUserID : int, formType : string) : boolean
ReportManager
ReportManagerController
-reportManagerService : final ReportManagerService
+receiveStudentReport(report : StudentReport) : Response
+checkValidity(report : StudentReport) : Response
+setStudentReport(studentID : int, form : StudentReport) : Response
+removeForm(firstUserID : int, secondUserID : int, formType : string) : Response
ReportManagerService
-userRepository : final UserRepository
-studentReportRepository : final StudentReportRepository
+receiveStudentReport(form : StudentReport) : StudentReport
+setStudentReport(studentID : int, form : StudentReport) : boolean
+checkValidity(form : StudentReport) : boolean
+removeForm(firstUserID : int, secondUserID : int, formType : string) : boolean

Figure 6. Web Server Management Layer

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



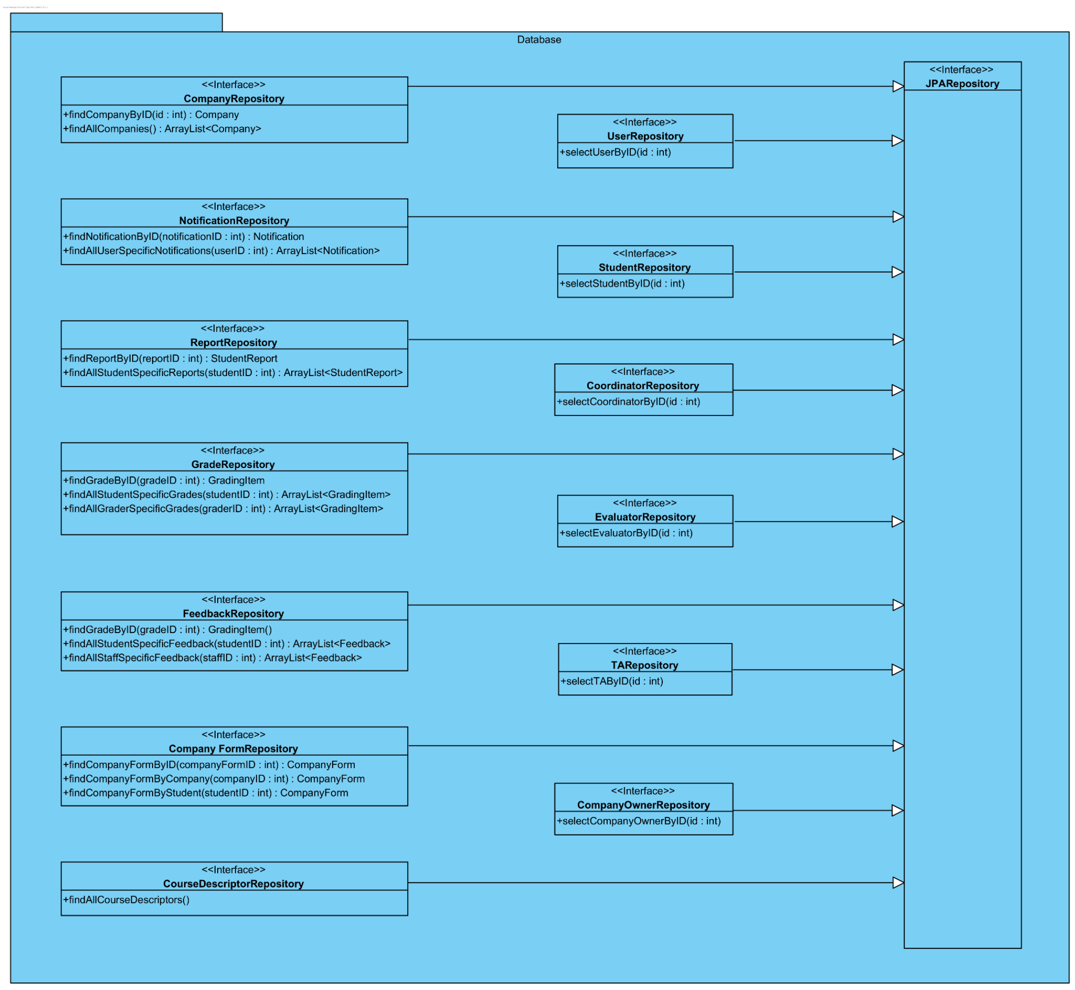

Database
<<Interface>>
JPARepository
<<Interface>>
CompanyRepository
+findCompanyByID(id : int) : Company
+findAllCompanies() : ArrayList<Company>
<<Interface>>
UserRepository
+selectUserByID(id : int)
<<Interface>>
NotificationRepository
+findNotificationByID(notificationID : int) : Notification
+findAllUserSpecificNotifications(userID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

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 object.

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, for grading or for 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. (needed)

- **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, 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, or 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 is supported in modern browsers.
- **SweetAlert2:** A JS library for pop-up notification handling.
- **password_compat (PHP Library):** For hashing password inputs and making other relevant data secure. Uses BCRYPT crypt algorithm.

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	

Name	Description
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 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 an 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 Descriptor

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	

Name	Description
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	
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	
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	

Name	Description
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, 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	
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	
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	
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

There are several cases in the application where many users may request a specific instance of a class. For example, we have a database connection object to handle all database related operations and only one instance of this connection is required to do all database operations. Using this 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. We can build a static function in our Database class that returns an instance of the database connection object to implement this pattern.

3.5.2 Factory Method Pattern

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 User class functionality is being inherited by all users as they all have similar items such as username, password, email, full name, bilkent id etc. and then there are also get/set methods for these parameters. In summary, we are implementing this pattern as we do have an abstract user class that defines all common attributes for all users and we have further sub-classes for each type of user.

3.5.3 Observer Pattern

In the case of the app, we have several events for which notifications are being sent to their respective receivers. To implement this pattern, we will be using a notification manager class that will handle all notification operations to form. For example, in the case of a Student Report Submission, a notification will be sent to all involved parties such as the Course Evaluator, the TA and the Coordinator to notify them of this event. The same will also be done in the case of a 'Revision Request' too as an example.

3.5.4 Decorator Pattern

There are cases in the working of the application where each object, which can be a user, or a report etc, need to be 'decorated' with further methods and attributes. For example, giving TA's the functionality to grade reports, or for Instructors to assign TAs to students. This pattern will also help us 'decorate' an object in the future and add/remove methods/attributes from it.

3.5.5 Adapter Pattern:

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 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. Glossary & References

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 preserves the formatting and content of the original document.

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.

password_compat: A PHP Library for hashing password inputs and making other relevant data secure. Uses BCRYPT crypt algorithm.

5.1 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. php. (n.d.). Retrieved May 5, 2023, from https://www.php.net/docs.php

Ircmaxell. (n.d.). Ircmaxell/PASSWORD_COMPAT: Compatibility with the password_* functions that ship with PHP 5.5. GitHub. Retrieved May 5, 2023, from https://github.com/ircmaxell/password_compat

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