SUMMER TRAINING REPORT EVALUATION SYSTEM

Design Report

Instructor: Eray Tüzün

Teaching Assistant(s): Muhammad Umair Ahmed, Yahya Elnouby

Group Name: Quokka

Berkay AKKUŞ - 21903321

İdil ATMACA - 22002491

Emre Melih GÜVEN - 22003944

Mehmet Emre KANTAŞ - 22003693

Yusuf ÖZYER - 21903339

Deniz SUN - 22003981

May 5, 2023

1. Introduction

1.1 Purpose of the System

Summer Training Report Evaluation System is a web application with the aim of automating and easing the summer training report evaluation process of Bilkent University Faculty of Engineering. The current system relies both on paperwork and digital platforms such as Google Drive, e-mail, Microsoft Excel, and Moodle. This application will help the academic staff complete the evaluation process faster and more reliably while improving time efficiency. The administrative assistants will upload the company evaluation forms to the system. Students will upload their summer training reports and instructors can fetch them from the system to grade them. Students can track their own submission progress and the instructors can track the submissions of their assigned students. Additionally, administrative assistants, instructors, and summer training coordinators will be able to see the statistical data for all submission stages and their completion status.

1.2 Design Goals

1.2.1 Usability

Users can reach the functionalities of the system easily without needing to navigate through many pages. The main functionalities are added to a sidebar that is specific to user types with their related actions. By gathering the functionalities on a sidebar, the content organization and navigation options are enhanced, which improves the overall user experience.

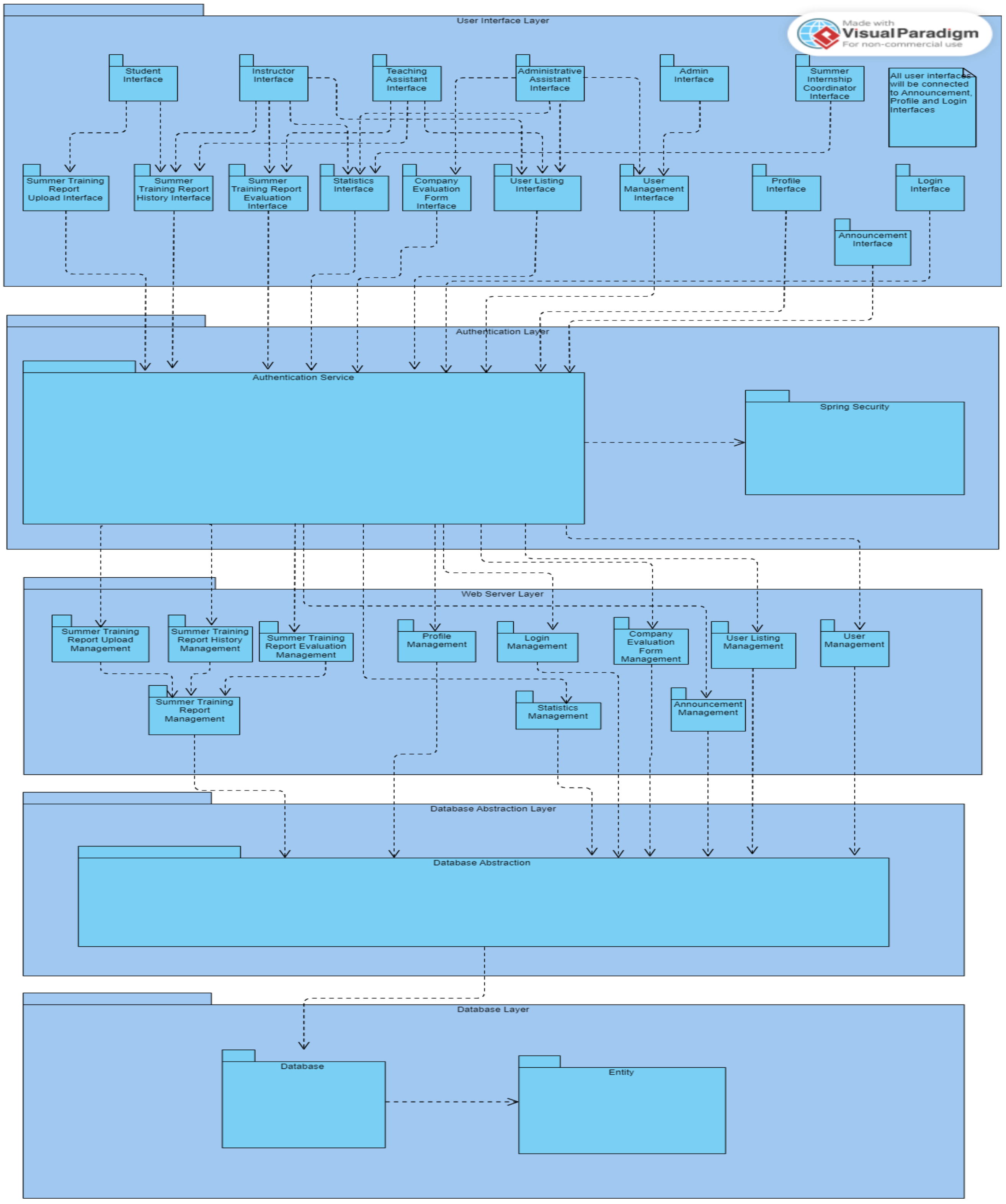
1.2.2 Automation

The current system is very time-consuming as there are a lot of platforms and paperwork involved. This system will reduce the workload of administrative assistants and summer training coordinators. It also reduces possible human errors related to paperwork and grading. As the system will keep track of each student's progress, it will also improve the trackability of the summer training reports' approval states.

2. High-Level Software Architecture

2.1. Subsystem Decomposition

https://online.visual-paradigm.com/share.jsp?id=323434323430362d31#diagram:workspace=wnzurcod&proj=0&id=1


User Interface Layer
Made with Visual Paradigm For non-commercial use
Student Interface
Instructor Interface
Teaching Assistant Interface
Administrative Assistant Interface
Admin Interface
Summer Internship Coordinator Interface
All user interfaces will be connected to Announcement, Profile and Login Interfaces
Summer Training Report Upload Interface
Summer Training Report History Interface
Summer Training Report Evaluation Interface
Statistics Interface
Company Evaluation Form Interface
User Listing Interface
User Management Interface
Profile Interface
Login Interface
Announcement Interface
Authentication Layer
Authentication Service
Spring Security
Web Server Layer
Summer Training Report Upload Management
Summer Training Report History Management
Summer Training Report Evaluation Management
Profile Management
Login Management
Company Evaluation Form Management
User Listing Management
User Management
Summer Training Report Management
Statistics Management
Announcement Management
Database Abstraction Layer
Database Abstraction
Database Layer
Database
Entity

2.2. Hardware/Software Mapping

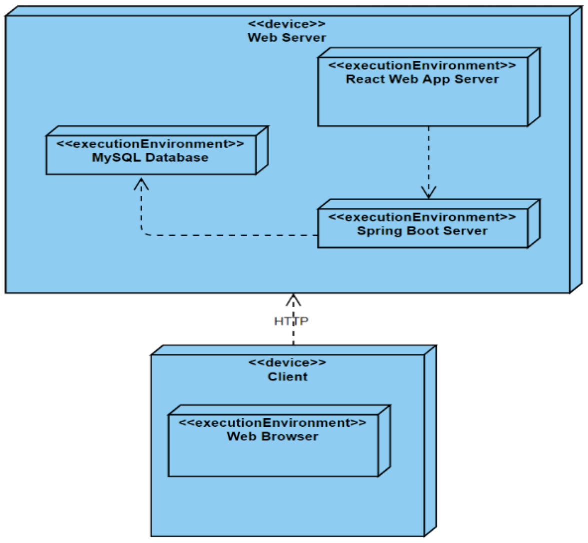
We chose to use React in the front end for the Summer Training Report Evaluation System. Along with React, we will be using HTML5 and CSS3, that is built inside the React code. React offers high performance and speed. It is easier to maintain because of its modular structure and ease of learning. The variety of packages inside it makes it easier to set up the system and add new functionalities.

According to 2022-2023 academic year data, the total of second and third-year students is 469 for CS. The courses for summer training, CS299 and CS399, are taken at the end of the second and third years. The number of students taking their department's summer training course is 375, 406, and 198 for IE, EEE, and ME, respectively. The sum of engineering faculty students taking these classes is approximately 1448. Still, adding up the exceptions and other users like instructors, administrative assistants, and summer training coordinators can lead to 1600 users. Moreover, the system should be scalable enough to involve all engineering faculty users and work great with scalability. So, we have considered the requirements for such a system to have at least 8 GB of RAM, an SSD with at least 500 GB of storage, with room for growth, a 64-bit operating system, and a CPU with a clock speed of at least 2.5 GHz.

We will be using Java, Spring Boot, and MySQL for the backend. Java was the natural choice since our group is familiar with it. Spring Boot is an ideal framework for web applications built on Java, making it a great fit for our project. We selected MySQL as our database because it's widely used with projects using React and Spring Boot, letting the team utilize useful packages in Spring Boot for MySQL.

Moreover, we will be using local service for our MySQL database since it will provide us complete control over the security of our data, and performance is faster compared to a cloud-based server. To enhance usability, we will deploy the system from one machine only. The user will connect through the HTTPS link from their computer and run the system on the web server. In this way, the user would only need a device and a stable internet connection to run the system.

Deployment diagram:



2.3. Persistent Data Management

For our internship management system's database, we carefully considered both MongoDB and MySQL due to their widespread use and abundance of documentation. While MongoDB allows for a more flexible schema design, MySQL is better suited for fixed schema designs and structures. After careful consideration, we decided to proceed with MySQL for our internship management system because we chose React for the front end, and connecting MySQL with React is relatively easier and has more resources available for it.

We will implement a fixed data schema for our database that will include tables for students, instructors, and other necessary entities such as name, surname, id, and user type. Students will also have a class code associated with them. We will create relationships between these tables, such as assigning students to teaching assistants and instructors. A fixed and structured data schema will allow us to manage and maintain the system more easily than a flexible schema design.

Moreover, we will be using the Spring Data JPA package that Spring Boot provides. This package enables an abstraction layer to connect MySQL and Spring Boot with minimal query usage.

2.4. Access Control and Security

Our group prioritizes access control systems and security. We will be using the data sheets provided by the school and adding them to the system. Adding and removing a user from the system can only be done by Admins or Administrative assistants. We do not provide a signup option for our users regarding security. Instead, after the admin or administrative assistant adds the user to the system, they will receive their encrypted password via their Bilkent email. When a user is added to the system, the person who is adding them will be responsible for assigning their role, which will then be stored in the database layer. In this way, we will be able to differentiate various kinds of users. We are aware that a user can have two roles. This is common for summer training coordinators being an instructor as well. So, we will be providing an extra page after the login to these users that gives them a chance to switch between roles.

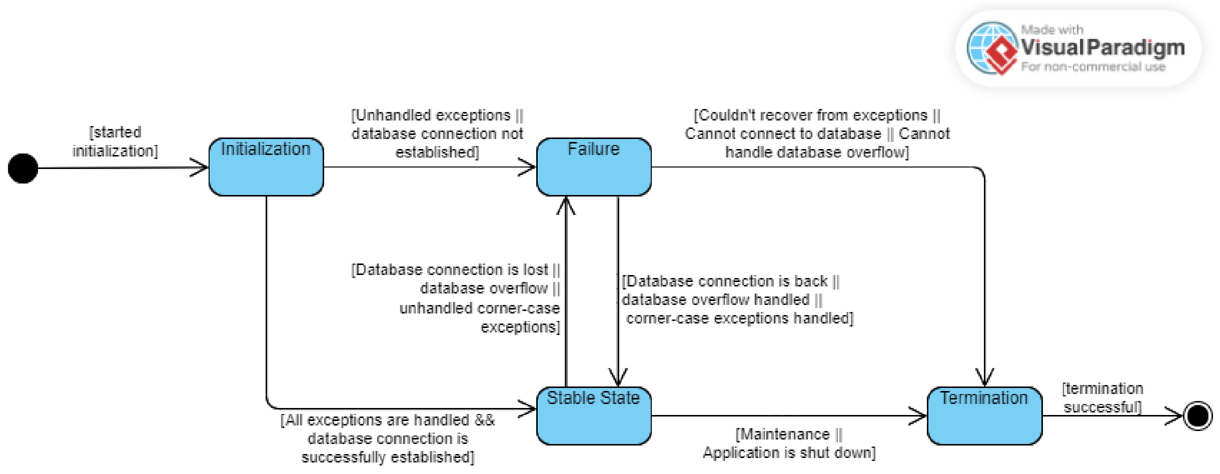
For the front end, users will only be shown pages that are related to them via the sidebar. We will be applying restrictions to pages to block a user from accessing a page that is not meant for them via changing the links in the browser. For example, a student cannot see the report evaluation page, whereas the instructor cannot see the upload reports page. Furthermore, whenever a user logs in to the system, we will get their role from the database and render the sidebar accordingly.

For security's sake, users have to log in to perform any of the use cases. We have assigned crucial acts to the actors that have high reliability. For example, only an administrative assistant would be able to send a reminder to an instructor that is falling behind in the report evaluation process. We have thought about giving this functionality to students as well, but we have decided that it could be a feature that could easily be abused. Moreover, the statistics page can only be seen by administrative assistants, summer training coordinators, and instructors. However, instructors can only see their own students, while administrative assistants and summer training coordinators have a more widespread view. Furthermore, since company forms are confidential, only administrative assistants will have full access to them. We will be using encryption to protect these forms. This ensures that only the students' assigned instructors will be able to access the form to enter their grades.

	Student	Instructor	Teaching Assistant	Summer Training Coordinator	Administrative Assistant	Admin
Login	X	X	X	X	X	X
Logout	X	X	X	X	X	X
View notifications	X	X	X	X	X	X
Make announcements to all users				X	X	
Make announcements to assigned users		X	X			
Change password	X	X	X	X	X	X
View reports' revision request	X					
View current status of a report	X	X				
View reports	X	X	X			
Edit/remove report	X					
Upload report with description	X					
Send revision request		X	X			
Assign due dates		X	X			
Fill/edit summer training grade form		X				
Upload e-signature		X				
Upload feedback		X	X			
View a student's current status		X		X	X	
View evaluation phases of grade forms		X		X	X	

Upload company evaluation form					**X**	
Enter grades for the company evaluation form		**X**				
View lists for all users				**X**	**X**	
View lists for assigned users		**X**	**X**			
View statistics for all students				**X**	**X**	
View statistics for assigned students		**X**				
Send reminders to instructors					**X**	
Assign/edit instructors and TAs to students					**X**	
Add/remove/edit all users					**X**	**X**
Add a data sheet for new users					**X**	**X**

2.5. Boundary Conditions


Made with Visual Paradigm For non-commercial use
[started initialization]
Initialization
[Unhandled exceptions || database connection not established]
Failure
[Couldn't recover from exceptions || Cannot connect to database || Cannot handle database overflow]
[Database connection is lost || database overflow || unhandled corner-case exceptions]
[Database connection is back || database overflow handled || corner-case exceptions handled]
[All exceptions are handled && database connection is successfully established]
Stable State
[Maintenance || Application is shut down]
Termination
[termination successful]

2.5.1. Initialization

- The system source files need to execute without any uncaught errors.
- The database needs to be ready and connected to the system for the system to be initialized.
- If there are unhandled exceptions or the database connection is not established, the system shifts to the failure state.
- Else, if all exceptions are handled and the database connection is established with no complications, the system shifts to the stable state.

2.5.2. Stable State

- After the system is initialized with a successful database establishment and with no uncaught errors, the system shifts to this state.
- While the system is in the stable state, if problems occur with the database (connection is lost or overflow occurs) or if there are unhandled exceptions from corner cases, the system shifts to the failure state.
- If the system is in the failure state and the problems mentioned above are resolved, the system shifts back to the stable state.
- If there is maintenance or the application is scheduled to shut down, the system will shift to the termination state.

2.5.3. Termination

- Admins can schedule maintenance that will terminate the system. 15 minutes before the system is down, a notification will be sent to users to let them know that the system will be under maintenance along with a countdown on the application. If any users are still logged in when the countdown ends, their data will be saved and they will be logged out of the system. During maintenance, new logins from users will not be allowed.
- Admins can shut the system down completely.
- If one of these two conditions is met, the system terminates successfully.
- System data will be kept in the database for later initializations after the termination.

2.5.4. Failure

- If there are problems with the database (connection is lost or overflow occurs) or if there are unhandled exceptions from corner cases, the system shifts to this failure state.
- If the entry condition(s) above are not resolved, a message will be displayed to the users and the system will terminate shortly after letting the users know. Users will see a pop-up that states why the system has crashed and when it will be operational again. Then they will be logged out of the system. After the system is operational again, the last save of the users will be restored from the database.

3. Low-Level Design

3.1. Object Design Trade-offs

3.1.1. Usability vs Functionality

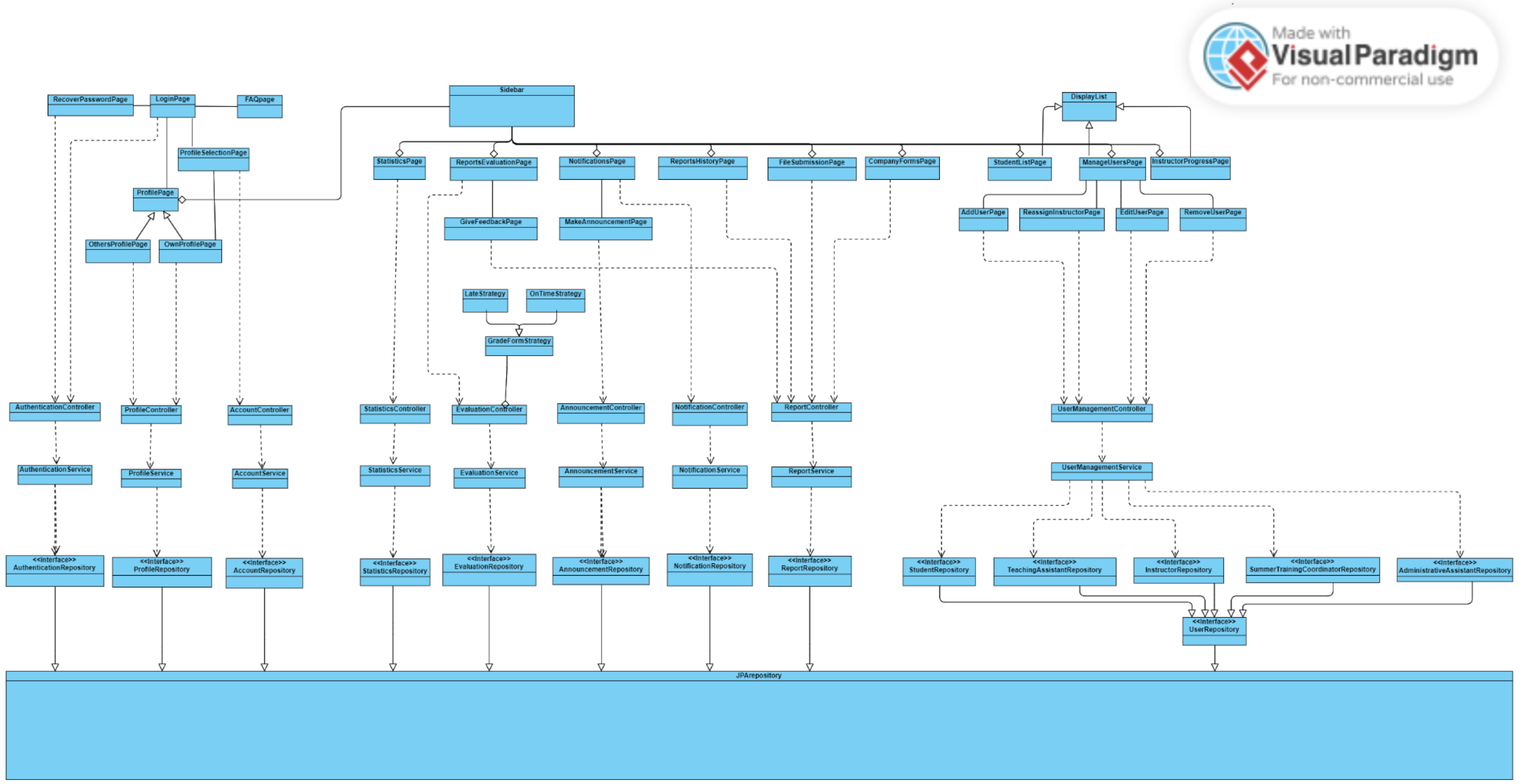
This application is made for the use of all academic staff that deals with summer training and students that are taking summer training courses. As each user needs to use the application easily, we did not add extra features that could make the user experience complicated.

3.1.2. Automation vs Rapid Development

This application decreases the workload of users by reducing the need for repetitive tasks between academic staff and students. To make the procedures easier and faster, the “automation” design goal is prioritized for the sake of rapid development.

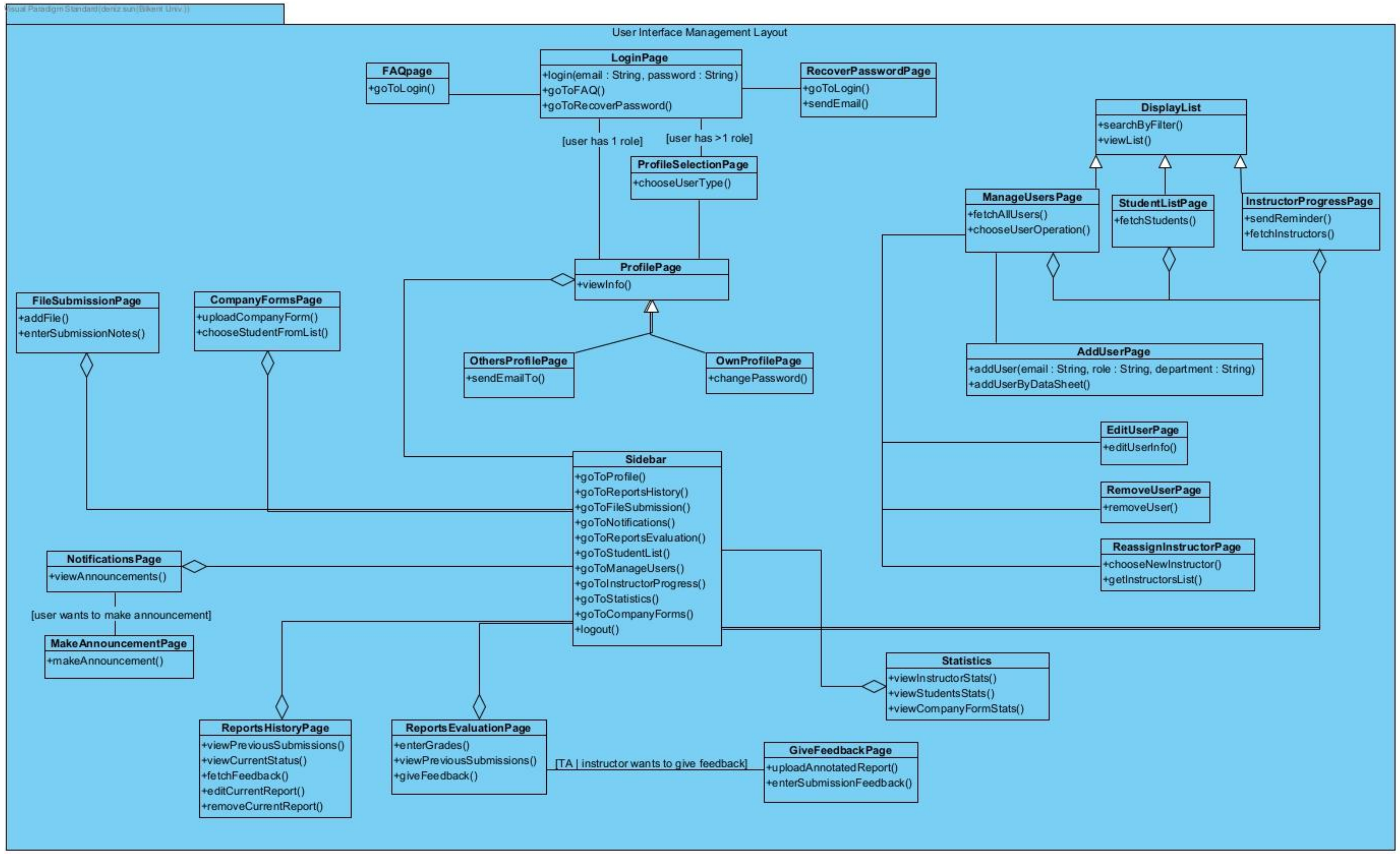
3.2 Final Object Design

https://online.visual-paradigm.com/share.jsp?id=323434323338352d31#diagram:workspace=hylibbej&proj=1&id=1



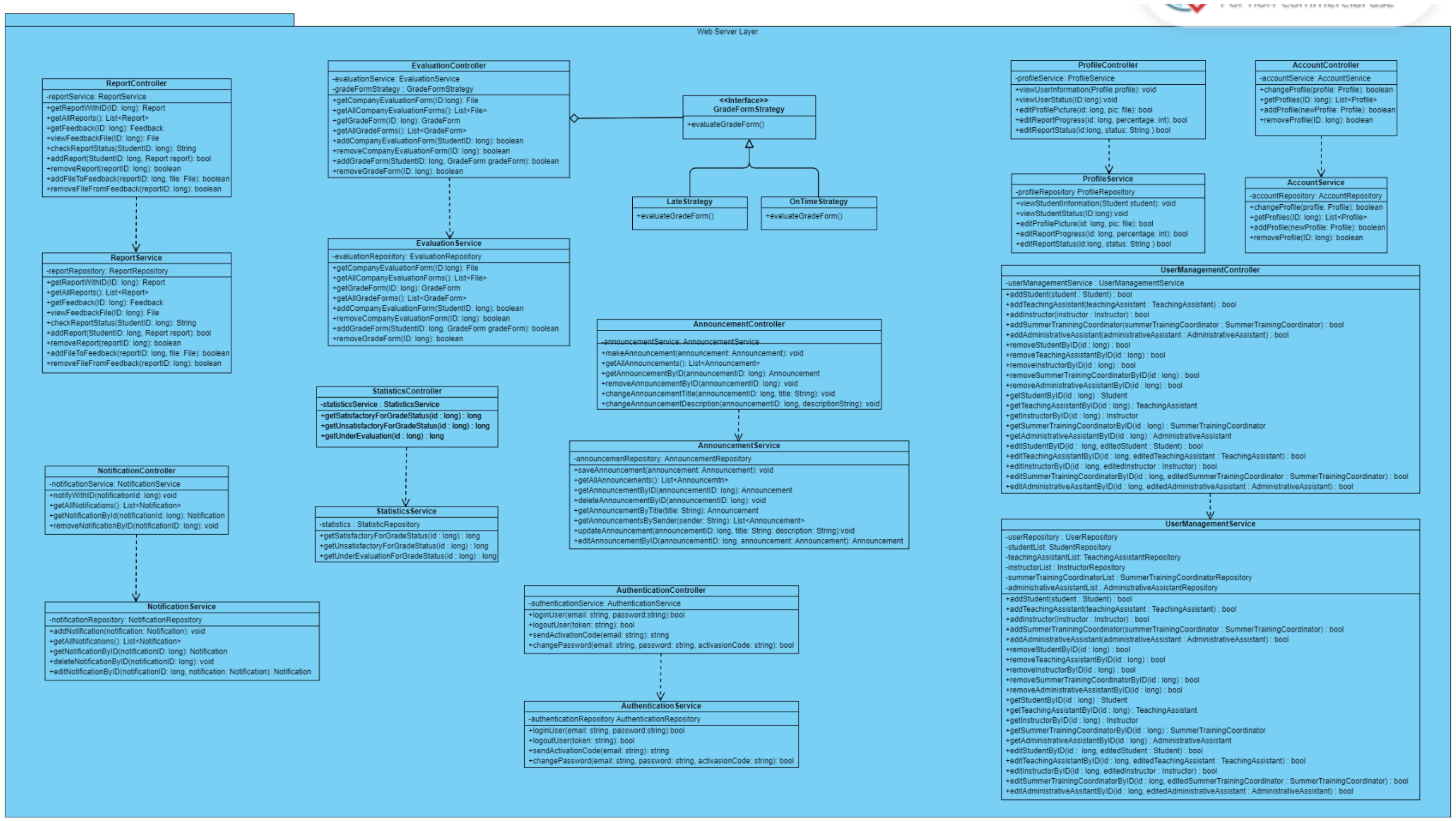
3.3. Layers

3.3.1 User Interface Management Layer



3.3.2 Web Server Layer

https://online.visual-paradigm.com/share.jsp?id=323234343632302d3131#diagram:workspace=enuekdrh&proj=2&id=11

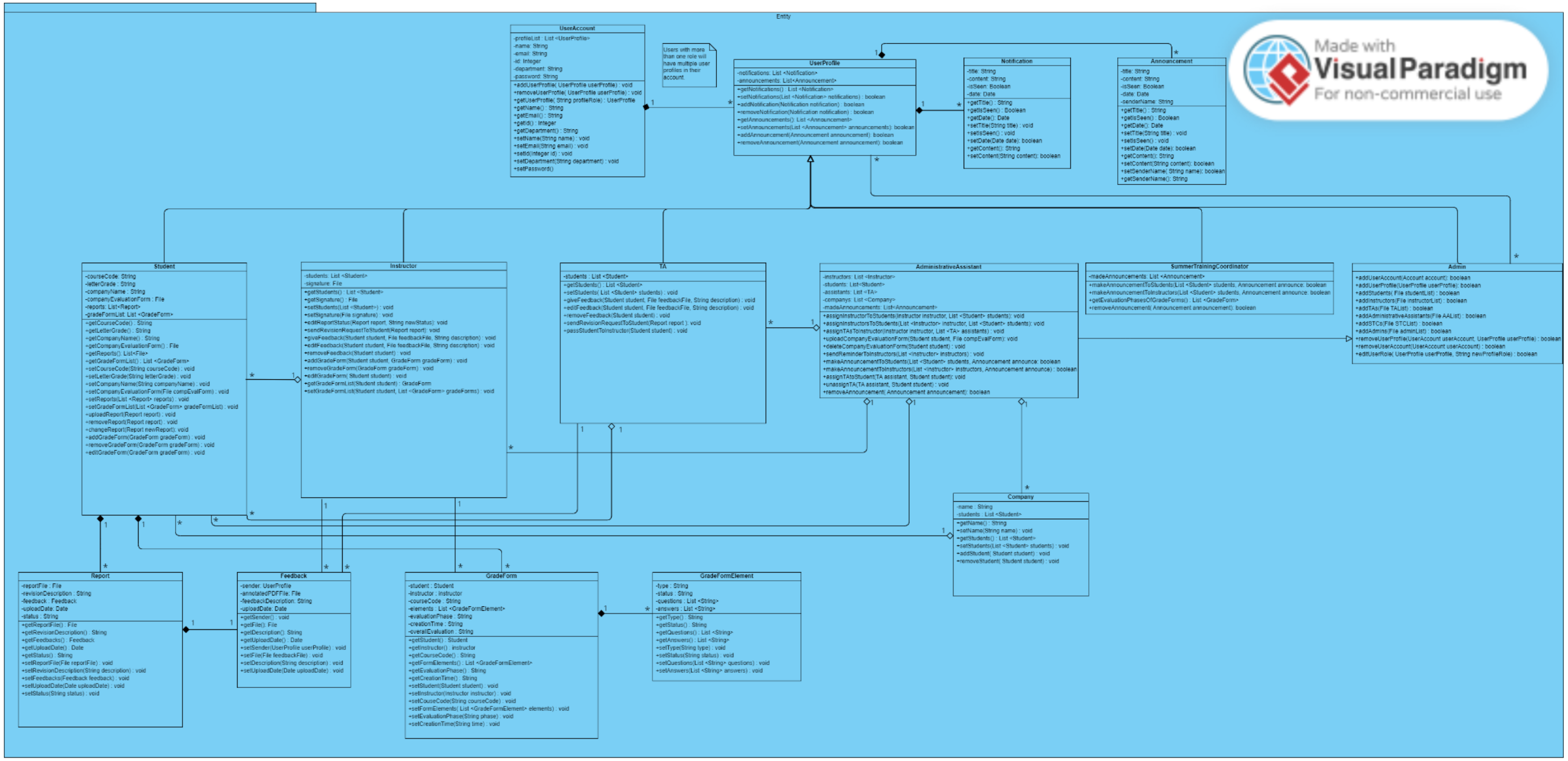


Web Server Layer
ReportController
-reportService: ReportService
+getReportWithID(ID: long): Report
+getAllReports(): List<Report>
+getFeedback(ID: long): Feedback
+viewFeedbackFile(ID: long): File
+checkReportStatus(StudentID: long): String
+addReport(StudentID: long, Report report): bool
+removeReport(reportID: long): boolean
+addFileToFeedback(reportID: long, file: File): boolean
+removeFileFromFeedback(reportID: long): boolean
ReportService
-reportRepository: ReportRepository
+getReportWithID(ID: long): Report
+getAllReports(): List<Report>
+getFeedback(ID: long): Feedback
+viewFeedbackFile(ID: long): File
+checkReportStatus(StudentID: long): String
+addReport(StudentID: long, Report report): bool
+removeReport(reportID: long): boolean
+addFileToFeedback(reportID: long, file: File): boolean
+removeFileFromFeedback(reportID: long): boolean
EvaluationController
-evaluationService: EvaluationService
-gradeFormStrategy : GradeFormStrategy
+getCompanyEvaluationForm(ID:long): File
+getAllCompanyEvaluationForms(): List<File>
+getGradeForm(ID: long): GradeForm
+getAllGradeForms(): List<GradeForm>
+addCompanyEvaluationForm(StudentID: long): boolean
+removeCompanyEvaluationForm(ID: long): boolean
+addGradeForm(StudentID: long, GradeForm gradeForm): boolean
+removeGradeForm(ID: long): boolean
EvaluationService
-evaluationRepository: EvaluationRepository
+getCompanyEvaluationForm(ID:long): File
+getAllCompanyEvaluationForms(): List<File>
+getGradeForm(ID: long): GradeForm
+getAllGradeForms(): List<GradeForm>
+addCompanyEvaluationForm(StudentID: long): boolean
+removeCompanyEvaluationForm(ID: long): boolean
+addGradeForm(StudentID: long, GradeForm gradeForm): boolean
+removeGradeForm(ID: long): boolean
<<Interface>>
GradeFormStrategy
+evaluateGradeForm()
LateStrategy
+evaluateGradeForm()
OnTimeStrategy
+evaluateGradeForm()
StatisticsController
-statisticsService : StatisticsService
+getSatisfactoryForGradeStatus(id : long) : long
+getUnsatisfactoryForGradeStatus(id : long) : long
+getUnderEvaluation(id : long) : long
StatisticsService
-statistics : StatisticRepository
+getSatisfactoryForGradeStatus(id : long) : long
+getUnsatisfactoryForGradeStatus(id : long) : long
+getUnderEvaluationForGradeStatus(id : long) : long
NotificationController
-notificationService: NotificationService
+notifyWithID(notificationId: long) void
+getAllNotifications(): List<Notification>
+getNotificationById(notificationId: long): Notification
+removeNotificationByID(notificationID: long): void
NotificationService
-notificationRepository: NotificationRepository
+addNotification(notification: Notification): void
+getAllNotifications(): List<Notification>
+getNotificationByID(notificationID: long): Notification
+deleteNotificationByID(notificationID: long): void
+editNotificationByID(notificationID: long, notification: Notification): Notification
AnnouncementController
~~-announcementService: AnnouncementService~~
+makeAnnouncement(announcement: Announcement): void
+getAllAnnouncements(): List<Announcement>
+getAnnouncementByID(announcementID: long): Announcement
+removeAnnouncementByID(announcementID: long): void
+changeAnnouncementTitle(announcementID: long, title: String): void
+changeAnnouncementDescription(announcementID: long, descriptionString): void
AnnouncementService
-announcemenRepository: AnnouncementRepository
+saveAnnouncement(announcement: Announcement): void
+getAllAnnouncements(): List<Announcemtn>
+getAnnouncementByID(announcementID: long): Announcement
+deleteAnnouncementByID(announcementID: long): void
+getAnnouncementByTitle(title: String): Announcement
+getAnnouncementsBySender(sender: String): List<Announcement>
+updateAnnouncement(announcementID: long, title: String: description: String):void
+editAnnouncementByID(announcementID: long, announcement: Announcement): Announcement
AuthenticationController
-authenticationService: AuthenticationService
+loginUser(email: string, password:string):bool
+logoutUser(token: string): bool
+sendActivationCode(email: string): string
+changePassword(email: string, password: string, activasionCode: string): bool
AuthenticationService
-authenticationRepository AuthenticationRepository
+loginUser(email: string, password:string):bool
+logoutUser(token: string): bool
+sendActivationCode(email: string): string
+changePassword(email: string, password: string, activasionCode: string): bool
ProfileController
-profileService: ProfileService
+viewUserInformation(Profile profile): void
+viewUserStatus(ID:long):void
+editProfilePicture(id: long, pic: file): bool
+editReportProgress(id: long, percentage: int): bool
+editReportStatus(id:long, status: String) bool
ProfileService
-profileRepository ProfileRepository
+viewStudentInformation(Student student): void
+viewStudentStatus(ID:long):void
+editProfilePicture(id: long, pic: file): bool
+editReportProgress(id: long, percentage: int): bool
+editReportStatus(id:long, status: String) bool
AccountController
-accountService: AccountService
+changeProfile(profile: Profile): boolean
+getProfiles(ID: long): List<Profile>
+addProfile(newProfile: Profile): boolean
+removeProfile(ID: long): boolean
AccountService
-accountRepository: AccountRepository
+changeProfile(profile: Profile): boolean
+getProfiles(ID: long): List<Profile>
+addProfile(newProfile: Profile): boolean
+removeProfile(ID: long): boolean
UserManagementController
-userManagementService : UserManagementService
+addStudent(student : Student) : bool
+addTeachingAssistant(teachingAssistant : TeachingAssistant) : bool
+addInstructor(instructor : Instructor) : bool
+addSummerTraniningCoordinator(summerTrainingCoordinator : SummerTrainingCoordinator) : bool
+addAdministrativeAssistant(administrativeAssistant : AdministrativeAssistant) : bool
+removeStudentByID(id : long) : bool
+removeTeachingAssistantByID(id : long) : bool
+removeInstructorByID(id : long) : bool
+removeSummerTrainingCoordinatorByID(id : long) : bool
+removeAdministrativeAssistantByID(id : long) : bool
+getStudentByID(id : long) : Student
+getTeachingAssistantByID(id : long) : TeachingAssistant
+getInstructorByID(id : long) : Instructor
+getSummerTrainingCoordinatorByID(id : long) : SummerTrainingCoordinator
+getAdministrativeAssistantByID(id : long) : AdministrativeAssistant
+editStudentByID(id : long, editedStudent : Student) : bool
+editTeachingAssistantByID(id : long, editedTeachingAssistant : TeachingAssistant) : bool
+editInstructorByID(id : long, editedInstructor : Instructor) : bool
+editSummerTrainingCoordinatorByID(id : long, editedSummerTrainingCoordinator : SummerTrainingCoordinator) : bool
+editAdministrativeAssitantByID(id : long, editedAdministrativeAssistant : AdministrativeAssistant) : bool
UserManagementService
-userRepository : UserRepository
-studentList: StudentRepository
-teachingAssistantList: TeachingAssistantRepository
-instructorList : InstructorRepository
-summerTrainingCoordinatorList : SummerTrainingCoordinatorRepository
-administrativeAssistantList : AdministrativeAssistantRepository
+addStudent(student : Student) : bool
+addTeachingAssistant(teachingAssistant : TeachingAssistant) : bool
+addInstructor(instructor : Instructor) : bool
+addSummerTraniningCoordinator(summerTrainingCoordinator : SummerTrainingCoordinator) : bool
+addAdministrativeAssistant(administrativeAssistant : AdministrativeAssistant) : bool
+removeStudentByID(id : long) : bool
+removeTeachingAssistantByID(id : long) : bool
+removeInstructorByID(id : long) : bool
+removeSummerTrainingCoordinatorByID(id : long) : bool
+removeAdministrativeAssistantByID(id : long) : bool
+getStudentByID(id : long) : Student
+getTeachingAssistantByID(id : long) : TeachingAssistant
+getInstructorByID(id : long) : Instructor
+getSummerTrainingCoordinatorByID(id : long) : SummerTrainingCoordinator
+getAdministrativeAssistantByID(id : long) : AdministrativeAssistant
+editStudentByID(id : long, editedStudent : Student) : bool
+editTeachingAssistantByID(id : long, editedTeachingAssistant : TeachingAssistant) : bool
+editInstructorByID(id : long, editedInstructor : Instructor) : bool
+editSummerTrainingCoordinatorByID(id : long, editedSummerTrainingCoordinator : SummerTrainingCoordinator) : bool
+editAdministrativeAssitantByID(id : long, editedAdministrativeAssistant : AdministrativeAssistant) : bool

3.3.3 Data Management Layer

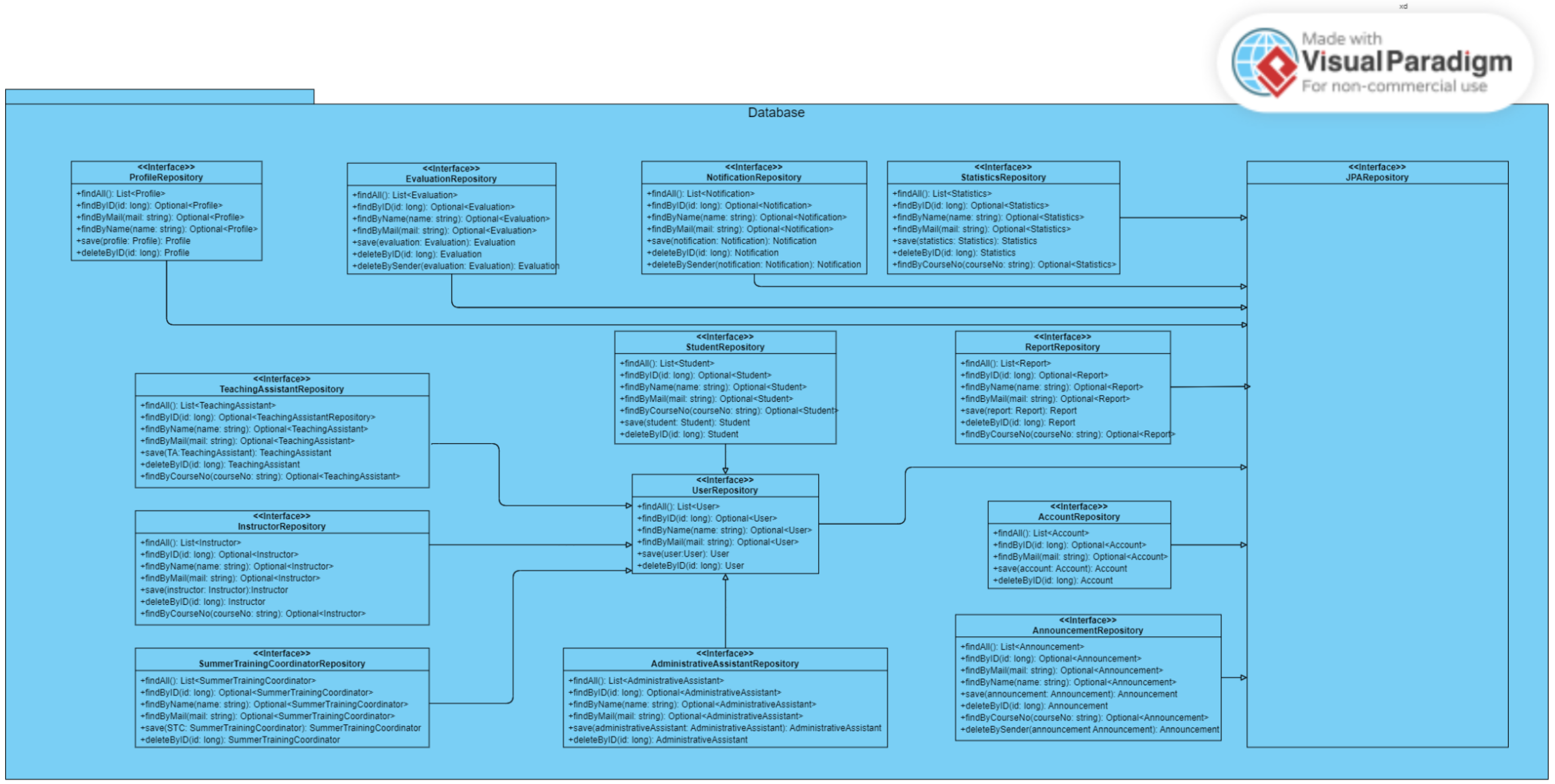
3.3.3.1 Entity

https://online.visual-paradigm.com/share.jsp?id=323234343632302d39


Entity
UserAccount
-profileList : List <UserProfile>
-name: String
-email: String
-id: Integer
-department: String
-password: String
+addUserProfile(UserProfile userProfile) : void
+removeUserProfile(UserProfile userProfile) : void
+getUserProfile(String profileRole) : UserProfile
+getName() : String
+getEmail() : String
+getId() : Integer
+getDepartment() : String
+setName(String name) : void
+setEmail(String email) : void
+setId(Integer id) : void
+setDepartment(String department) : void
+setPassword()
Users with more than one role will have multiple user profiles in their account.
UserProfile
-notifications: List <Notification>
-announcements: List<Announcement>
+getNotifications() : List <Notification>
+setNotifications(List <Notification> notifications) : boolean
+addNotification(Notification notification) : boolean
+removeNotification(Notification notification) : boolean
+getAnnouncements(): List <Announcement>
+setAnnouncements(List <Announcement> announcements): boolean
+addAnnouncement(Announcement announcement): boolean
+removeAnnouncement(Announcement announcement): boolean
Notification
-title: String
-content: String
-isSeen: Boolean
-date: Date
+getTitle() : String
+getIsSeen() : Boolean
+getDate(): Date
+setTitle(String title) : void
+setIsSeen() : void
+setDate(Date date): boolean
+getContent(): String
+setContent(String content): boolean
Announcement
-title: String
-content: String
-isSeen: Boolean
-date: Date
-senderName: String
+getTitle() : String
+getIsSeen() : Boolean
+getDate(): Date
+setTitle(String title) : void
+setIsSeen() : void
+setDate(Date date): boolean
+getContent(): String
+setContent(String content): boolean
+setSenderName(String name): boolean
+getSenderName(): String
Made with Visual Paradigm For non-commercial use
Student
-courseCode: String
-letterGrade : String
-companyName : String
-companyEvaluationForm : File
-reports: List<Report>
-gradeFormList: List <GradeForm>
+getCourseCode() : String
+getLetterGrade() : String
+getCompanyName() : String
+getCompanyEvaluationForm() : File
+getReports(): List<File>
+getGradeFormList() : List <GradeForm>
+setCourseCode(String courseCode) : void
+setLetterGrade(String letterGrade) : void
+setCompanyName(String companyName) : void
+setCompanyEvaluationForm(File compEvalForm) : void
+setReports(List <Report> reports) : void
+setGradeFormList(List <GradeForm> gradeFormList) : void
+uploadReport(Report report) : void
+removeReport(Report report) : void
+changeReport(Report newReport): void
+addGradeForm(GradeForm gradeForm) : void
+removeGradeForm(GradeForm gradeForm) : void
+editGradeForm(GradeForm gradeForm) : void
Instructor
-students: List <Student>
-signature: File
+getStudents() : List <Student>
+getSignature() : File
+setStudents(List <Student>) : void
+setSignature(File signature) : void
+editReportStatus(Report report, String newStatus): void
+sendRevisionRequestToStudent(Report report): void
+giveFeedback(Student student, File feedbackFile, String description) : void
+editFeedback(Student student, File feedbackFile, String description) : void
+removeFeedback(Student student) : void
+addGradeForm(Student student, GradeForm gradeForm) : void
+removeGradeForm(GradeForm gradeForm) : void
+editGradeForm(Student student) : void
+getGradeFormList(Student student) : GradeForm
+setGradeFormList(Student student, List <GradeForm> gradeForms) : void
TA
-students : List <Student>
+getStudents() : List <Student>
+setStudents(List <Student> students) : void
+giveFeedback(Student student, File feedbackFile, String description) : void
+editFeedback(Student student, File feedbackFile, String description) : void
+removeFeedback(Student student) : void
+sendRevisionRequestToStudent(Report report): void
+passStudentToInstructor(Student student) : void
AdministrativeAssistant
-instructors: List <Instructor>
-students: List<Student>
-assistants: List <TA>
-companys: List <Company>
-madeAnnouncements: List<Announcement>
+assignInstructorToStudents(Instructor instructor, List <Student> students): void
+assignInstructorsToStudents(List <Instructor> instructor, List <Student> students): void
+assignTAsToInstructor(Instructor instructor, List <TA> assistants) : void
+uploadCompanyEvaluationForm(Student student, File compEvalForm): void
+deleteCompanyEvaluationForm(Student student) : void
+sendReminderToInstructors(List <Instructor> instructors) : void
+makeAnnouncementToStudents(List <Student> students, Announcement announce: boolean
+makeAnnouncementToInstructors(List <Instructor> instructors, Announcement announce) : boolean
+assignTAtoStudent(TA assistant, Student student): void
+unassignTA(TA assistant, Student student) : void
+removeAnnouncement(Announcement announcement): boolean
SummerTrainingCoordinator
-madeAnnouncements: List <Announcement>
+makeAnnouncementToStudents(List <Student> students, Announcement announce: boolean
+makeAnnouncementToInstructors(List <Student> students, Announcement announce: boolean
+getEvaluationPhasesOfGradeForms() : List <GradeForm>
+removeAnnouncement(Announcement announcement): boolean
Admin
+addUserAccount(Account account): boolean
+addUserProfile(UserProfile userProfile): boolean
+addStudents(File studentList) : boolean
+addInstructors(File instructorList) : boolean
+addTAs(File TAList) : boolean
+addAdministrativeAssistants(File AAList) : boolean
+addSTCs(File STCList) : boolean
+addAdmins(File adminList) : boolean
+removeUserProfile(UserAccount userAccount, UserProfile userProfile) : boolean
+removeUserAccount(UserAccount userAccount) : boolean
+editUserRole(UserProfile userProfile, String newProfileRole) : boolean
Company
-name : String
-students : List <Student>
+getName() : String
+setName(String name) : void
+getStudents() : List <Student>
+setStudents(List <Student> students) : void
+addStudent(Student student) : void
+removeStudent(Student student) : void
Report
-reportFile : File
-revisionDescription : String
-feedback : Feedback
-uploadDate: Date
-status : String
+getReportFile() : File
+getRevisionDescription() : String
+getFeedbacks() : Feedback
+getUploadDate() : Date
+getStatus() : String
+setReportFile(File reportFile) : void
+setRevisionDescription(String description) : void
+setFeedbacks(Feedback feedback) : void
+setUploadDate(Date uploadDate) : void
+setStatus(String status) : void
Feedback
-sender: UserProfile
-annotatedPDFFile: File
-feedbackDescription: String
-uploadDate: Date
+getSender() : void
+getFile(): File
+getDescription(): String
+getUploadDate() : Date
+setSender(UserProfile userProfile) : void
+setFile(File feedbackFile) : void
+setDescription(String description) : void
+setUploadDate(Date uploadDate) : void
GradeForm
-student : Student
-instructor : instructor
-courseCode : String
-elements : List <GradeFormElement>
-evaluationPhase : String
-creationTime : String
-overallEvaluation : String
+getStudent() : Student
+getInstructor() : instructor
+getCourseCode() : String
+getFormElements() : List <GradeFormElement>
+getEvaluationPhase() : String
+getCreationTime() : String
+setStudent(Student student) : void
+setInstructor(Instructor instructor) : void
+setCouseCode(String courseCode) : void
+setFormElements(List <GradeFormElement> elements) : void
+setEvaluationPhase(String phase) : void
+setCreationTime(String time) : void
GradeFormElement
-type : String
-status : String
-questions : List <String>
-answers : List <String>
+getType() : String
+getStatus() : String
+getQuestions() : List <String>
+getAnswers() : List <String>
+setType(String type) : void
+setStatus(String status) : void
+setQuestions(List <String> questions) : void
+setAnswers(List <String> answers) : void

3.3.3.2 Database

https://online.visual-paradigm.com/share.jsp?id=3234343234313


Made with Visual Paradigm
For non-commercial use
Database
<<Interface>>
ProfileRepository
+findAll(): List<Profile>
+findByID(id: long): Optional<Profile>
+findByMail(mail: string): Optional<Profile>
+findByName(name: string): Optional<Profile>
+save(profile: Profile): Profile
+deleteByID(id: long): Profile
<<Interface>>
EvaluationRepository
+findAll(): List<Evaluation>
+findByID(id: long): Optional<Evaluation>
+findByName(name: string): Optional<Evaluation>
+findByMail(mail: string): Optional<Evaluation>
+save(evaluation: Evaluation): Evaluation
+deleteByID(id: long): Evaluation
+deleteBySender(evaluation: Evaluation): Evaluation
<<Interface>>
NotificationRepository
+findAll(): List<Notification>
+findByID(id: long): Optional<Notification>
+findByName(name: string): Optional<Notification>
+findByMail(mail: string): Optional<Notification>
+save(notification: Notification): Notification
+deleteByID(id: long): Notification
+deleteBySender(notification: Notification): Notification
<<Interface>>
StatisticsRepository
+findAll(): List<Statistics>
+findByID(id: long): Optional<Statistics>
+findByName(name: string): Optional<Statistics>
+findByMail(mail: string): Optional<Statistics>
+save(statistics: Statistics): Statistics
+deleteByID(id: long): Statistics
+findByCourseNo(courseNo: string): Optional<Statistics>
<<Interface>>
JPARepository
<<Interface>>
StudentRepository
+findAll(): List<Student>
+findByID(id: long): Optional<Student>
+findByName(name: string): Optional<Student>
+findByMail(mail: string): Optional<Student>
+findByCourseNo(courseNo: string): Optional<Student>
+save(student: Student): Student
+deleteByID(id: long): Student
<<Interface>>
ReportRepository
+findAll(): List<Report>
+findByID(id: long): Optional<Report>
+findByName(name: string): Optional<Report>
+findByMail(mail: string): Optional<Report>
+save(report: Report): Report
+deleteByID(id: long): Report
+findByCourseNo(courseNo: string): Optional<Report>
<<Interface>>
TeachingAssistantRepository
+findAll(): List<TeachingAssistant>
+findByID(id: long): Optional<TeachingAssistantRepository>
+findByName(name: string): Optional<TeachingAssistant>
+findByMail(mail: string): Optional<TeachingAssistant>
+save(TA:TeachingAssistant): TeachingAssistant
+deleteByID(id: long): TeachingAssistant
+findByCourseNo(courseNo: string): Optional<TeachingAssistant>
<<Interface>>
UserRepository
+findAll(): List<User>
+findByID(id: long): Optional<User>
+findByName(name: string): Optional<User>
+findByMail(mail: string): Optional<User>
+save(user:User): User
+deleteByID(id: long): User
<<Interface>>
InstructorRepository
+findAll(): List<Instructor>
+findByID(id: long): Optional<Instructor>
+findByName(name: string): Optional<Instructor>
+findByMail(mail: string): Optional<Instructor>
+save(instructor: Instructor):Instructor
+deleteByID(id: long): Instructor
+findByCourseNo(courseNo: string): Optional<Instructor>
<<Interface>>
AccountRepository
+findAll(): List<Account>
+findByID(id: long): Optional<Account>
+findByMail(mail: string): Optional<Account>
+save(account: Account): Account
+deleteByID(id: long): Account
<<Interface>>
AnnouncementRepository
+findAll(): List<Announcement>
+findByID(id: long): Optional<Announcement>
+findByMail(mail: string): Optional<Announcement>
+findByName(name: string): Optional<Announcement>
+save(announcement: Announcement): Announcement
+deleteByID(id: long): Announcement
+findByCourseNo(courseNo: string): Optional<Announcement>
+deleteBySender(announcement Announcement): Announcement
<<Interface>>
SummerTrainingCoordinatorRepository
+findAll(): List<SummerTrainingCoordinator>
+findByID(id: long): Optional<SummerTrainingCoordinator>
+findByName(name: string): Optional<SummerTrainingCoordinator>
+findByMail(mail: string): Optional<SummerTrainingCoordinator>
+save(STC: SummerTrainingCoordinator): SummerTrainingCoordinator
+deleteByID(id: long): SummerTrainingCoordinator
<<Interface>>
AdministrativeAssistantRepository
+findAll(): List<AdministrativeAssistant>
+findByID(id: long): Optional<AdministrativeAssistant>
+findByName(name: string): Optional<AdministrativeAssistant>
+findByMail(mail: string): Optional<AdministrativeAssistant>
+save(administrativeAssistant: AdministrativeAssistant): AdministrativeAssistant
+deleteByID(id: long): AdministrativeAssistant

3.4. Class Interfaces

3.4.1. User Interface Layer Class Interfaces

3.4.1.1. LoginPage

+login(email:String, password: String): On click, a request is sent to the web server in order to authenticate the user's credentials.
+goToFAQ(): Goes to the FAQ page. (Frequently Asked Questions)
+goToRecoverPassword(): On click, goes to the password recovery page.

3.4.1.2. FAQpage

+goToLogin(): On click, goes to the login page.

3.4.1.3. RecoverPasswordPage

+goToLogin(): On click, goes to the Login page.
+sendEmail(): On click, sends a verification email to the user's email.

3.4.1.4. UserTypePage

+chooseUserType(): Users with more than one role have to choose which account (which role) they are logging in to.

3.4.1.5. ProfilePage

+viewInfo(): Shows user information such as name, email, courses. Additionally, students can see their assigned instructor's name and email.

3.4.1.6. OthersProfilePage

+sendEmailTo(): Allows users to send an email to the person they are viewing the profile of.

3.4.1.7. OwnProfilePage

+changePassword(): Allows users to change their account password.

3.4.1.8. FileSubmissionPage

+addFile(): Allows users to upload summer training reports as a file to the system.
+enterSubmissionNotes(): Allows users to enter comments along with their file submission.

3.4.1.9. CompanyFormsPage

+uploadCompanyForm(): Allows administrative assistants to upload a student’s company evaluation form to the system.
+chooseStudentFromList(): Allows administrative assistants to choose which student they are uploading the company evaluation form of.

3.4.1.10. NotificationsPage

+viewAnnouncements(): Allows users to view announcements.

3.4.1.11. MakeAnnouncementPage

+makeAnnouncement(): Allows administrative assistants, summer training coordinators and instructors to make announcements.

3.4.1.12. Sidebar

+goToProfile(): On click, goes to the user’s Profile Page.
+goToReportsHistory(): On click, goes to the Reports History Page.
+goToFileSubmission(): On click, goes to the File Submission Page.
+goToNotifications(): On click, goes to the Notifications Page.
+goToReportsEvaluation(): On click, goes to the Reports Evaluation Page.
+goToStudentList(): On click, goes to the Student List Page.
+goToManageUsers(): On click, goes to the Manage Users Page.
+goToInstructorProgress(): On click, go to the Instructor Progress Page.
+logout(): On click, the user is logged out of the system.
+goToStatistics(): On click, goes to the Statistics Page.
+goToCompanyForms(): On click, goes to the Company Forms Page.

3.4.1.13. ReportsHistoryPage

+viewPreviousSubmissions(): Allows students to view their previous submissions/ revisions.
+viewCurrentStatus(): Allows students to view the status of their last submitted report, which is currently being graded.
+fetchFeedback(): Shows students feedback for their previous submissions/ revisions.
+editCurrentReport(): Allows students to edit their last submitted report.
+removeCurrentReport(): Allows students to remove their last submitted report.

3.4.1.14. ReportsEvaluationPage

+enterGrades(): Allows instructors to enter grades for each part of a student's summer training report.
+viewPreviousSubmissions(): Allows instructors to view a student's previous submission details.
+giveFeedback(): Allows instructors and TAs to give feedback on a student's current report by redirecting to the Give Feedback Page.

3.4.1.15. GiveFeedbackPage

+uploadAnnotatedReport(): Allows TAs or instructors to upload the annotated version (with feedback) of a student's report.
+enterSubmissionFeedback(): Allows TAs or instructors to add comments and notes about the feedback.

3.4.1.16. StatisticsPage

+viewInstructorStats(): Shows the grading progress of instructors and statistics.
+viewStudentsStats(): Shows the submission progress of students and statistics.
+viewCompanyFormStats(): Shows the statistics of uploaded company forms.

3.4.1.17. DisplayList

+searchByFilter(): Allows users to search for another user by filters such as name and department.
+viewList(): Shows users the list of users and the results of their search, if any.

3.4.1.18. ManageUsersPage

+fetchAllUsers(): Gets all the users registered in the system.
+chooseUserOperations(): On click, allows administrative assistants to choose an operation related to a user and redirects them to its specific page: Add User Page, Edit User Page, Remove User Page.

3.4.1.19. StudentListPage

+fetchStudents(): Gets all the students registered in the system.

3.4.1.20. InstructorProgressPage

+sendReminder(): Sends a notification to instructors to remind them to proceed with grading a student's report.
+fetchInstructors(): Gets all the instructors registered in the system.

3.4.1.21. AddUserPage

+addUser(email: String, role: String, department: String): Allows administrative assistants to add a user to the system by entering the user's email, role, and department.
+addUsersByDataSheet(): Allows administrative assistants to add multiple users by uploading a data sheet that stores the users.

3.4.1.22. EditUserPage

+editUserInfo(): Allows administrative assistants to edit a registered user's information.

3.4.1.23. RemoveUserPage

+removeUser(): Allows administrative assistants to remove a registered user from the system.

3.4.1.24. ReassignInstructorPage

+chooseNewInstructor(): Allows administrative assistants to change a student's assigned instructor by choosing a new one.
+getInstructorList(): Gets the instructor list for administrative assistants to choose from when reassigning a student's instructor.

3.5 Design Patterns

3.5.1 Decorator Pattern

This application will have a lot of file uploads and downloads. There will be three different documents corresponding to each student. First one is the company evaluation forms of students, the second one is the summer training reports and the third one is the final grade forms that instructors sign and submit. The company evaluation forms and student reports may have multiple versions (revisions). To ensure there is no confusion or overriding of documents, they need to be named according to a naming convention. This will be ensured by keeping the documents by student ID, document name and course code. Keeping track of the course code is crucial for students that are taking two summer training courses at once.

The administrative assistants choose which student they are uploading the company evaluation forms of. The system will rename the file to “StudentID_CEF_CourseCode_<iterNo>”. (CEF: Company Evaluation Form) When uploading the summer training reports, students can name them however they want. The system will rename it to “StudentID_Report_CourseCode_<iterNo>”. For both of these uploads, iterNo will keep track of the revisions. It will automatically increase if a new version of the file is uploaded for the same student. For the grade forms, the system will rename the file to “StudentID_GradeForm_CourceCode”.

3.5.2 Strategy Pattern

In Summer Training Management System, there will be a lot of approvals, rejections, and iteration requests for student reports. Also, the company evaluation form affects the process. Furthermore, there are cases where company evaluation forms are late. In order to manage all these situations well, we will use the Strategy Pattern. We will have 2 strategies. The first one is the algorithm for the early/on-time company evaluation forms, and the second one is for late company evaluation forms. These 2 strategies will help us break out from the evaluation process if a late company evaluation form is insufficient for the student to proceed with part B, which is the part where teaching assistants and instructors give feedback on summer training reports.

These 2 strategies will, because of the strategy design pattern’s nature, provide us with 2 different algorithms for the evaluation process of summer training grade forms of the students. Inner control points for each strategy are different. For example, in each stage of the strategy for late company evaluations, arrival and result of the company evaluation report will affect the flow of stages, But in early/on-time company evaluations, after part A, the system will no longer look for a company evaluation form.

3.5.3 Façade Pattern

To optimize the performance of our system, we implemented the Façade pattern as a data abstraction layer for the database. With frequent access to the database and the need for simultaneous regulation of user lists and report updates, accessing the entire database every time would be redundant. Thus, we simplified the interactions between the user and the system by encapsulating the database implementation behind a simple interface.

To maintain consistency and simplicity throughout the system, we also extended the Façade pattern to the web server layer, where we implemented structures called services. This allowed us to handle changes in system boundaries without affecting the functionality of the controllers.