Bilkent University
Department of Computer Engineering

CS319 Object-Oriented Software Engineering

BILTERN: Summer Training Course Management Application

Project Design Report

- Cenk Merih Olcay 22002408
- Arshia Bakhshayesh 22001468
- Enes Bektaş 22002401
- Faruk Uçgun 22003016
- Faaiz Khan 22001476

Instructor: Eray Tüzün
Teaching Assistant(s): Muhammad Umair Ahmed and Yahya Elnoub

Contents

1 Introduction

1.1 The purpose of the system

Biltern is a web application that aims to facilitate the process of summer training report evaluation at Bilkent University. Currently, there isn't a single system that does the job completely. Students use Moodle/Webmail to upload their reports, and then the feedback is provided to them via Moodle/Webmail as well. Reuploads of the reports involve another email to the teaching assistant/grader. And the final results of the students are uploaded on a different website as soon as the grader has graded them. All of these tasks can be easily done via an automated digital system, in this case: Biltern. Biltern aims to make the entire process of summer training easier and more efficient for all actors involved.

1.2 Design Goals

The main purpose of the system is to provide a feasible and reliable way of efficiently processing the summer report grading procedure in one intuitive-to-use application. The aim is to eliminate the current difficult multi-application process which does not allow much communication between the actors. The two goals that we have deemed most important are usability and functionality. We will also keep the maintainability of the project in mind while designing it. The system will have multiple users: undergraduate students, teaching assistants, graders, coordinators, secretaries, and the BCC admin. The system aims to include functionalities for the aforementioned users to make their respective jobs easier, quicker, and more reliable.

1.2.1 Usability

One of the main goals of this system is to make it easy to use for all parties involved. This requires a good flow of information and QOL(Quality Of Life) features so everyone can do their job efficiently. As the system will have to account for an average of 200 students from each department every year, it is imperative that everyone is able to perform their tasks without much struggle. An easy-to-navigate user interface, intuitive and familiar instructions for each procedure such as grading

and uploading reports, and parallelizing the web application with the already-in-place system are some of the ways that help the application to be more usable.

1.2.2 Functionality

Another main goal of this system is to provide a way for the users to communicate with each other without having to write an email every time something is needed. To do this, Biltern will implement - amongst other things - a notification system that will automatically reach out to interested parties when a development in the summer training process occurs. This could be an undergraduate student sending a report, a teaching assistant providing feedback, or a grader grading the report. This will also make the jobs of the companies as well as the graders easier as they would be able to fill the required grading forms by making use of interactive pre-filled forms.

1.2.3 Maintainability

As the curriculum evolves and the guidelines change, it is vital to the function of the system that it can be easily maintained. The system will be used every year, meaning that it will have to be updated regularly to suit the needs of the university as well as those of the students. The maintainability of the initial application and updates is facilitated through the use of design patterns (section 3.5), the usage of clean code conventions (naming conventions, avoidance of nested logic), and documentation. The frameworks used (Springboot and React) also provide their own utilities for communicating the purpose of each part of functionality through the use of tags and naming conventions.

2 High-level software architecture

2.1 Subsystem decomposition

Our subsystem decomposition has a 4 layered architecture. We have User Interface, Application, Database Abstraction, and Database Layers. This way, we separated the system into 4 parts where there are similar packages interacting with each other.

This approach seemed a good fit to make our system more secure, maintainable, and usable.

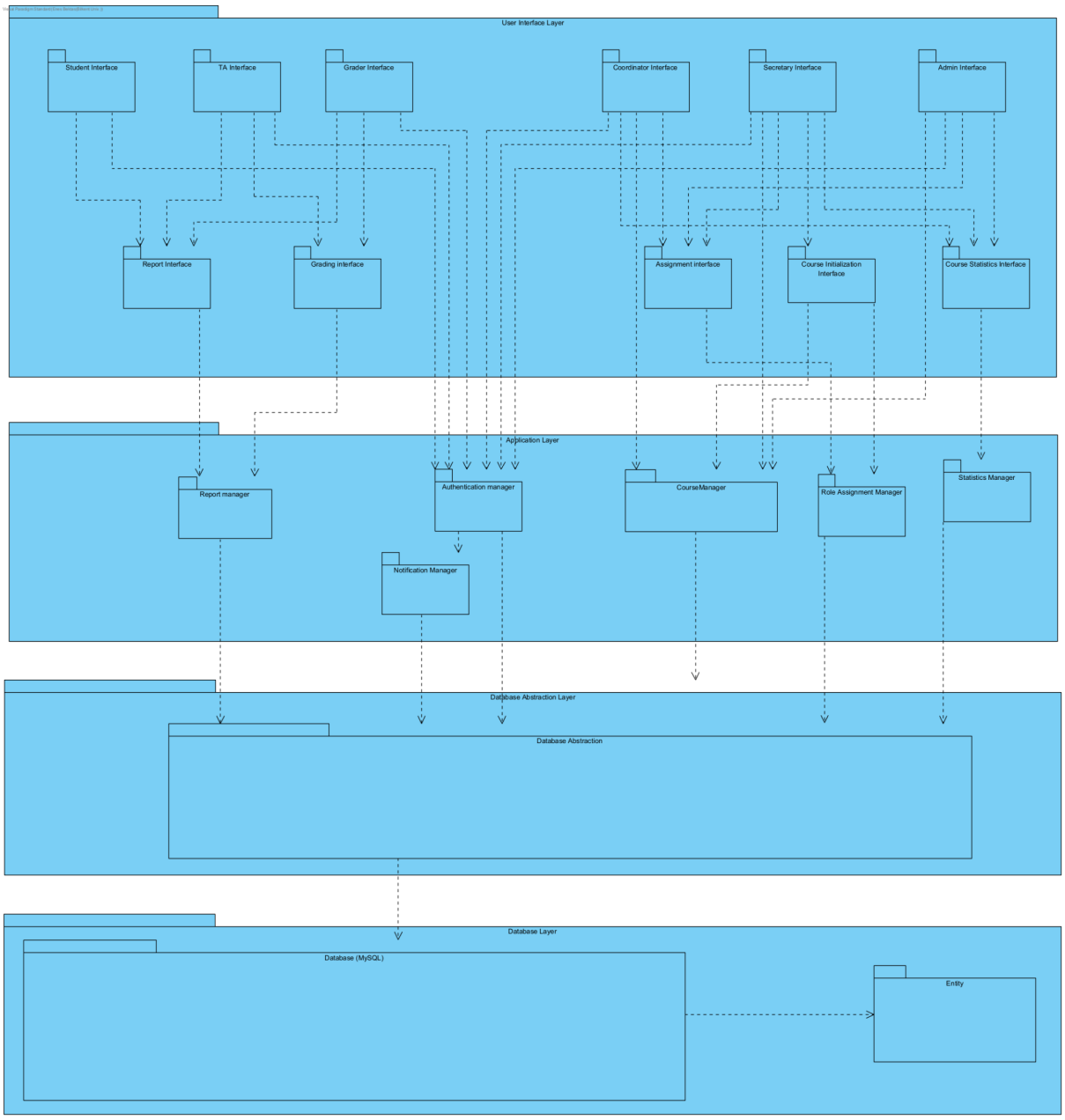

Figure 1. Subsystem Decomposition Diagram

2.1.1 User Interface Layer

User Interface layer refers to the interaction between users and boundary objects through which they can use the application. They can view information on the related pages provided and input information for the services to do operations and grant requests. We have divided the User Interface Layer according to user roles and some general packages that combine similar functionalities that represent the main

features of the program. Different roles have access to different combination of interfaces representing the main features of the program. This way, users have access to functionalities that they are allowed only.

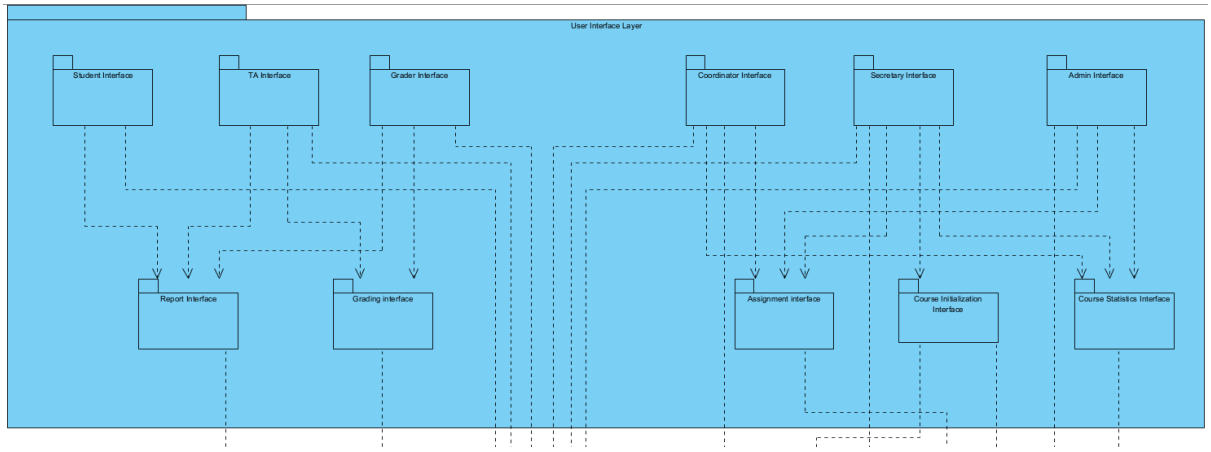

Figure 2. User Interface Layer Diagram

2.1.2 Application Layer

Application Layer consists of controller and service classes that control the underlying logic of the program. It processes the data gathered from the user through the User Interface Layer and can alter data in the database whenever necessary through its connection to the repositories.

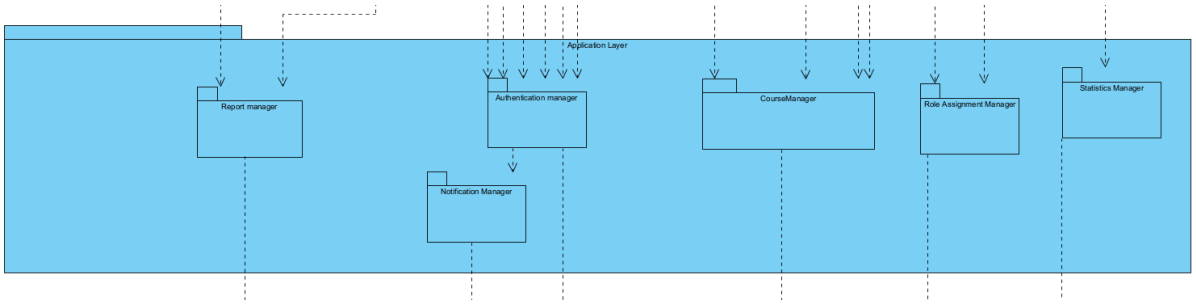

Figure 3. Application Layer Diagram

2.1.3 Database Abstraction Layer

Database Abstraction Layer contains the repositories required to interact with Database Layer. Each repository is connected to the JPA repository and uses JPA API to construct the objects from the database(entities). It also can insert new entity instances into the database.

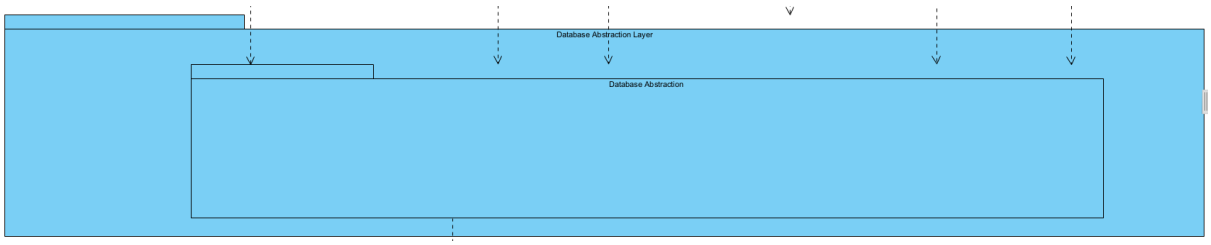

Figure 4. Database Abstraction Layer Diagram

2.1.4 Database Layer

This layer represents the entity objects, their attributes, and their relations in the database. The repository will be responsible for changing the states of these and fetching the states of these entity objects.

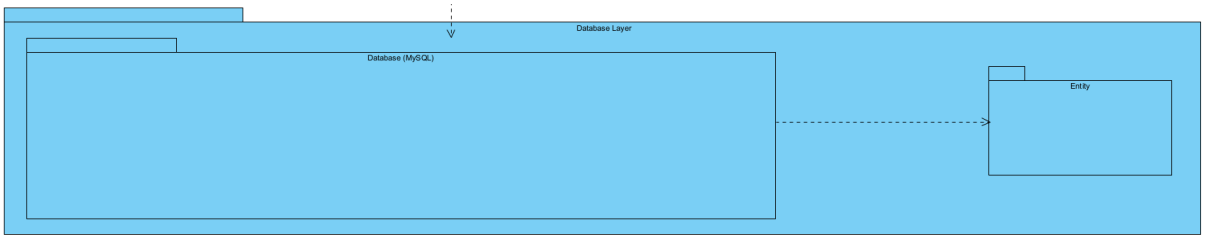

Figure 5. Database Layer Diagram

2.2 Hardware/software mapping

We will be using React with JavaScript on the front end. We will also use HTML5 and CSS, which all modern browsers support. Hence, the only requirement for using Biltern is a stable version of a modern internet browser and a stable internet connection.

We will use Java, Spring Boot, and MySQL on the backend. The backend will be hosted locally. Our database will be on AWS for ease of development and use. Around 1250 people in total from computer engineering, industrial engineering, electrical and electronics engineering, and mechanical engineering departments. We can deduce this information by looking at the number of people who took XX299 and XX399 courses in the previous years. Using the AWS free tier, we will be able to store and serve the data of these people.

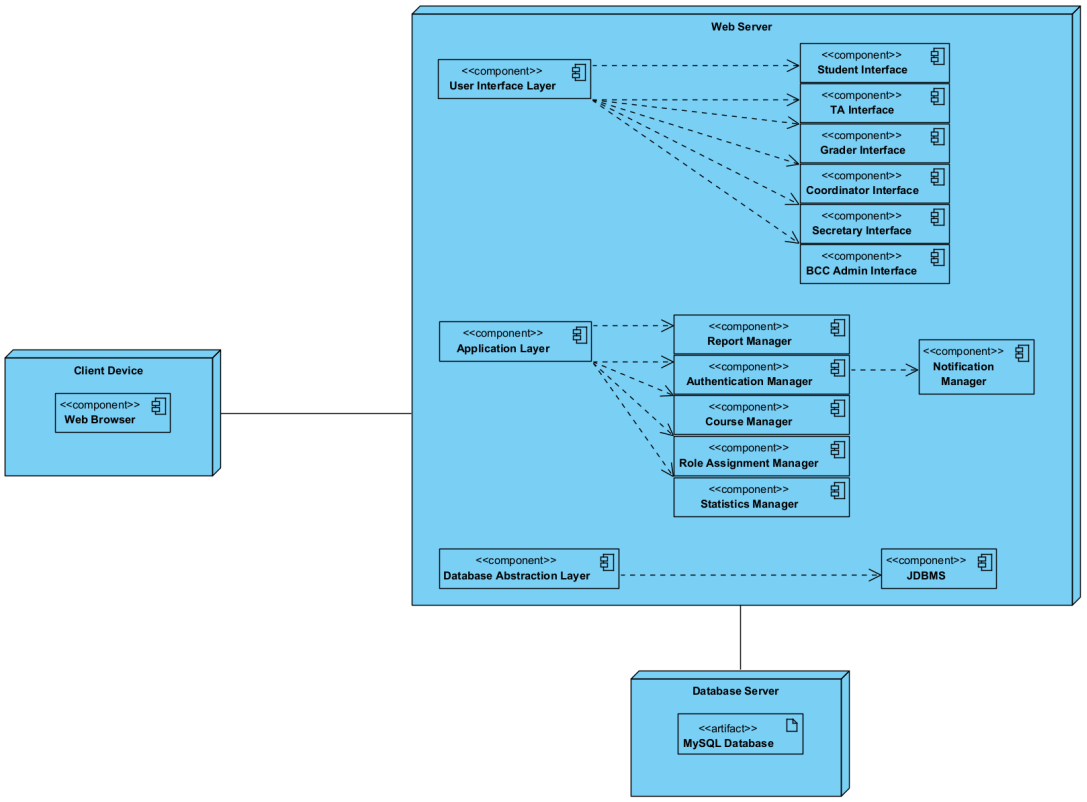

Figure 6. Deployment Diagram

2.3 Persistent data management

For our project, MySQL set up on an Amazon RDS(Relational Database Service) instance will be used as the database technology for a multitude of reasons: scalability, flexibility, performance, robust transactional support, and strong data protection [1]. As Biltern will have to be able to manage a variety of complex data: reports, users, courses, etc., MySQL seems like the right choice.

The MySQL database will store important information about the users, reports, grading forms, and courses. Complex data types like report pdfs will be stored as blobs and decoded when used in a client.

Since the data will be related to each other, i.e., reports to users and courses, etc., the need for a fast and efficient relational database management system is also fulfilled by MySQL, as the application requires a high performance to ensure the process is made easier for all parties involved.

2.4 Access control and security

One of the main design goals of Biltern project is to provide secure and reliable connections to avoid unnecessary or unexpected behaviors of users within their interactions with the Biltern system.

Biltern provides role-based views in the front end to present only necessary interfaces according to the user's role. Front-end accesses user roles through the response body return after the successful authentication which includes type of role. This is achieved through conditional rendering of components inside the client's React project.

Also, Biltern uses role-based authentication in the back-end server to provide restrictions to the type of requests clients can send through user actions. To achieve this feature, use of Spring Security's user interfaces as well as service interfaces will be in use when managing application users and their authorizations.

	Methods	Student	Secretary	Department Coordinator	Teaching Assistant	Grader	BCC ADMIN
Authentication Manager	getAuthenticationToken	X	X	X	X	X	X
	changePassword	X	X	X	X	X	X
	forgotPassword	X	X	X	X	X	X
Report Manager	saveReportPreviewFeedback				X		
	saveReportFeedback					X	
	removeFeedback					X	
	removePreviewFeedback				X		
	removeReport	X					
	addReport	X					

	updateGradingForm					X	
	addCompanyContact	X					
	issueCompanyContactRequest					X	
	changeReportDueDates					X	
	getCourseReports	X			X	X	
	removeCompanyContactRequest					X	
	changeReportApprovalDueDate					X	
NotificationManager	sendDeadlineNotification					X	
	sendCourseRegistrationNotification		X				
	sendStudentRegisteredNotification		X				
	sendTAChangedNotification		X				
	sendGraderChangedNotification		X				
	sendCompanyContactRequestN					X	
CourseManager	createCourse		X				
	endCourse		X				
	changeCourseEndDate		X				
	removeFromCourse		X				
	addParticipantToCourse		X			X	
	swicthGraders		X				
Statistics Manager	getCourseStatistics			X			

	getGraderStatistics			X			
	getTAStatistics			X			
RoleAssignment Manager	assignRole		X	X			X
	assignRoleToMultipleUsers		X	X			
	registerUser		X	X			X

Figure 7. Access Control Table

2.5 Boundary conditions

Overview of the application state:

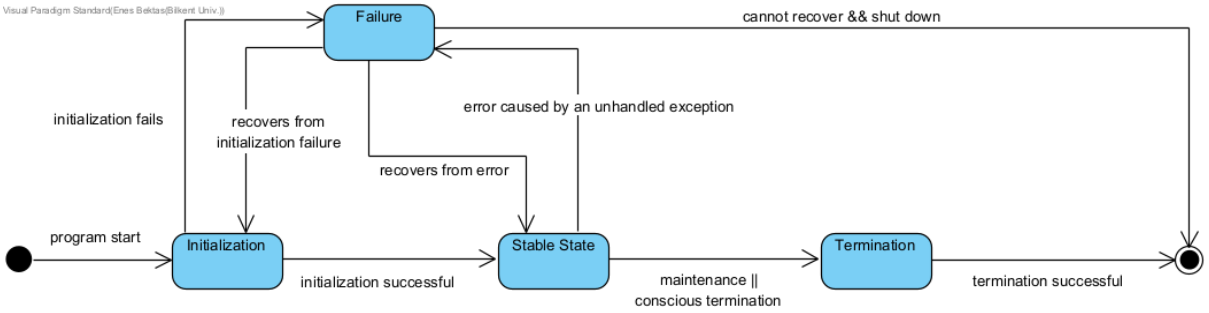

Figure 8. Boundary Conditions Diagram

2.5.1 Initialization

- For the initialization of the whole application, an Amazon RDS instance will be started by the developers, and the AWS initializations for hosting the front and back end will be initiated (deployment, DNS configuration, SSL certificate).
- Each time the web page is accessed, the static components used in the front end will be checked for availability.
- Using component lifecycle techniques, the correct rendering of the page is checked.
- A health endpoint of the AWS server on which the spring boot backend is hosted will be pinged.

- The spring boot backend will then attempt a connection to the RDS server and check the validity of this connection.
- At any point, if any of the checks have failed, the program will enter a failure state.
- If the checks and validations are successful then the application will enter a stable state.

2.5.2 Failure state

- At any point, if a mismatch of data or data corruption instances happens from a stable state (for example, a user uploading a file and the application not being able to resolve the file to an existing user), the application will enter a failure state. It will use AWS API to figure out possible failures.
- If the AWS API does not provide adequate resources for handling the failure, the application will terminate.
- For connection checks and problems, AWS and RDS APIs will be utilized to check the server's or database's state. If the connection can not be made, the application terminates otherwise the page will reload to a stable state.
- The maintenance and securing of the data are done by the RDS Amazon instance and Amazon AWS server itself in case of loss of data and other database failures.
- In case of a failed component rendering, the application will enter a failure state.
- The system will enter termination if after multiple refreshes and component lifetime checks (React) the components can not be rendered completely.
- The login token is authenticated for each request before granting a request. meaning that in case of users going out of the bounds of their role, such as trying to access a URL that does not exist in the application, an error page will be shown and they will be redirected to the dashboard or login page which is in a stable state. This situation can also be triggered when roles try to access other role’s URLs(for example a Student trying to access an instructor grading form).
- Before entering the failure state, the user inputs will be locally saved until the failure is resolved.

2.5.3 Stable state

- The process of data transfer between the user and the application will be monitored.
- If the AWS endpoints return valid data on each request, the application will stay in a stable state.
- The stable state consists of successfully fulfilling the routine tasks of the application and indicates the application is functioning within predicted scenarios.

2.5.4 Termination

- In case of a session token expires, the application terminates and returns to the login page while saving the data of the last page the user was using.
- The user can be logging out expires their session token
- The failure state can force the termination of the application. In this case, an appropriate error page is shown
- In case of scheduled maintenance, users will be informed beforehand by the system to eliminate inconvenience and the system will not be usable for a set amount of time.

3 Low-level design

3.1 Object design trade-offs

3.1.1 Functionality vs. Rapid Development

As Biltern is not an app that is used differently through the passage of years, it makes sense that the functionality aspect of the system is given more importance than the rapid development aspect.

There is more focus being directed toward streamlining the summer training process with automated notifications/emails, gradings, and resubmissions. Of course, this means that the system would be slow to develop as the focus is more on providing a functional system for the university to organize and manage the grading of the summer training reports. The main goal is to provide a stable and automated system rather than the best one that is constantly adapting to technological changes.

3.1.2 Usability vs. Maintainability

At this stage of development, the purpose of the system is to provide intuitive connections for the internship report management process and replace the current already-in-place system. The technologies used (Springboot, React) have shown remarkable relevance throughout the years, and the assumption is that this will continue until the near foreseeable future. For this reason, to facilitate a welcoming and easy-to-use environment so that it becomes natural to use our main users for going through the internship report grading process, we have prioritized usability over maintainability. As summer trainings happens once every year (twice on very rare occasions) and at the same time, the system focuses more on adding features that make the process easier for everyone involved. These features include: uploading and reuploading reports, notifications, feedback and grading, and many more.

3.1.3 Low-cost vs. Performance

In the Biltern project, to implement and deploy the project with a more reasonable budget, we have chosen an approach that favors the use of low-cost external services over expensive counterparts that would provide high performance. As an example, we will use free tier plans of email (Gmail SMTP) and deployment (AWS EC2) services to keep the project's monetary cost under a moderate threshold.

3.2 Final object design

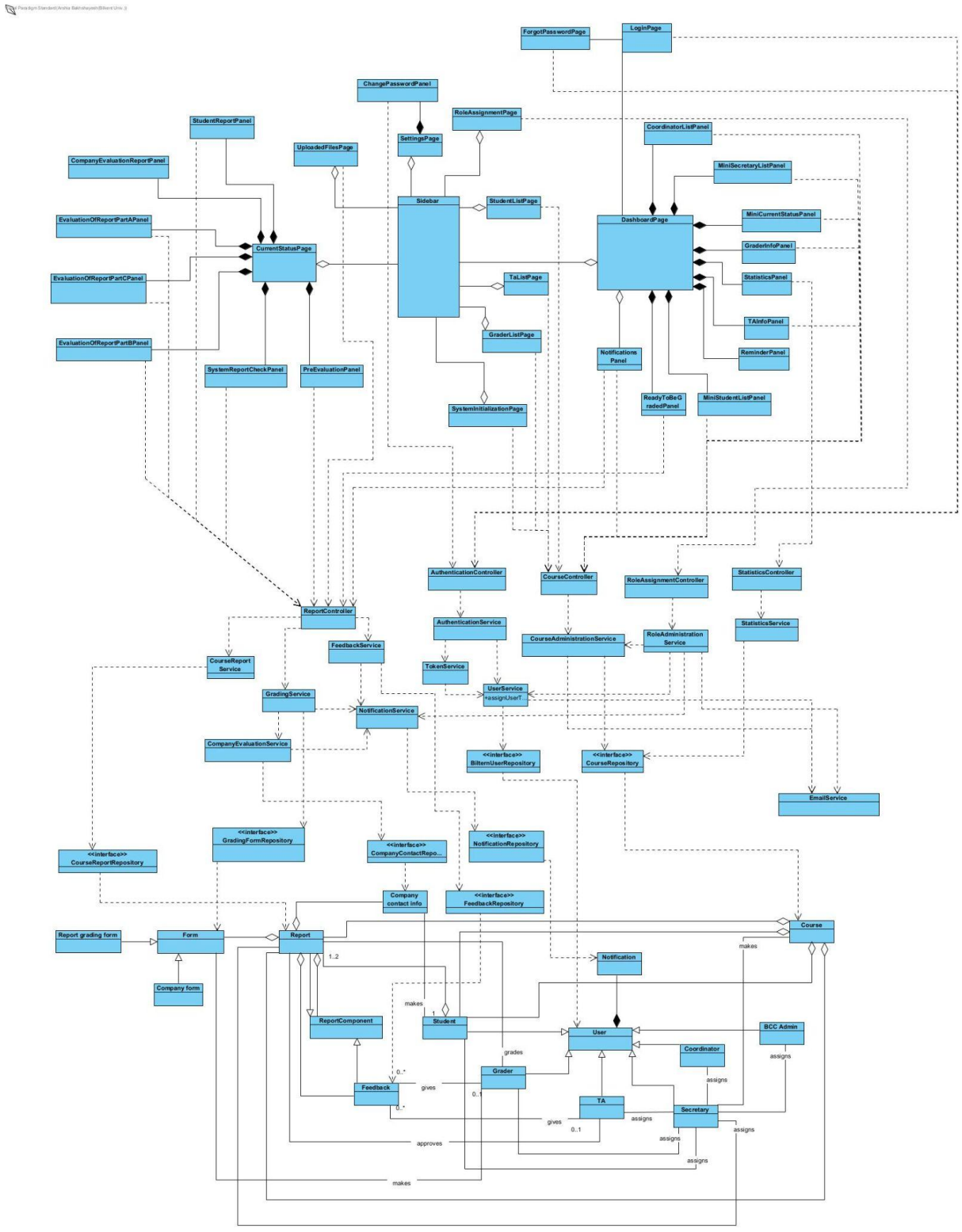

Figure 9. Final Object Design Diagram

For better resolution:
https://drive.google.com/file/d/1GdOsMs0aofSxxvkePuE5OwScsf0KVCyp/view?usp=sharing

3.3 Packages (Layers)

3.3.1 User Interface Layer

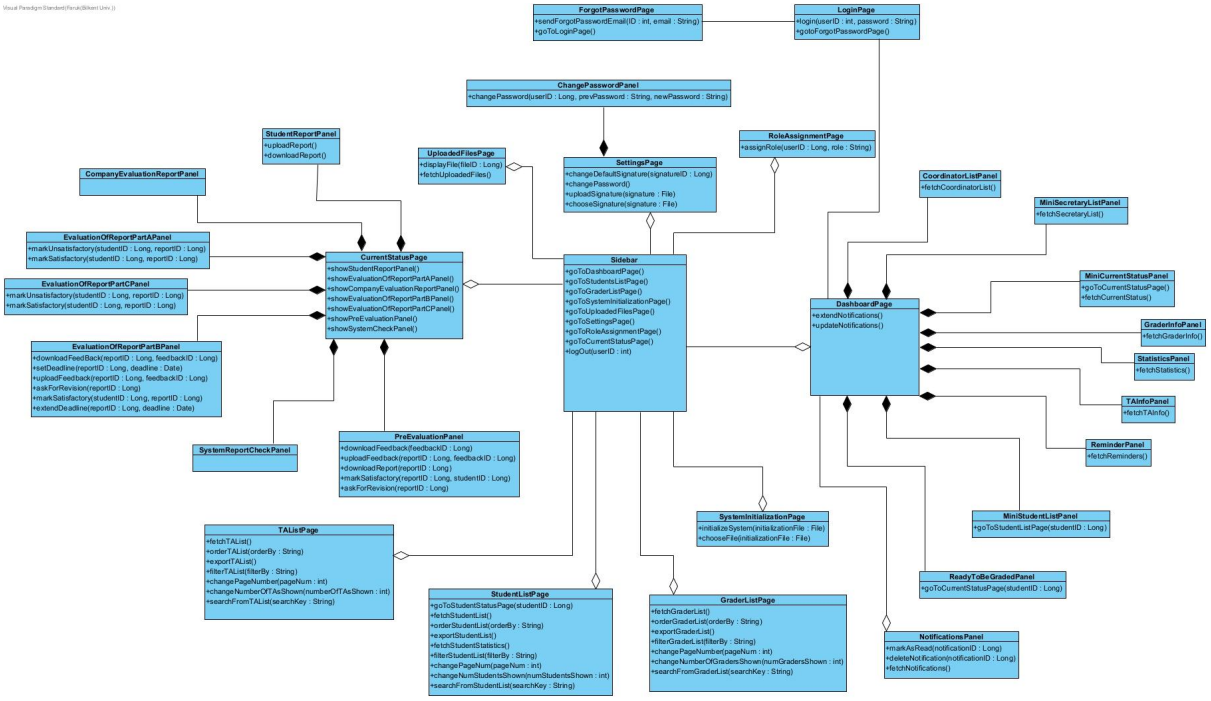

Figure 10. User Interface Object Diagram

3.3.2 Application Layer

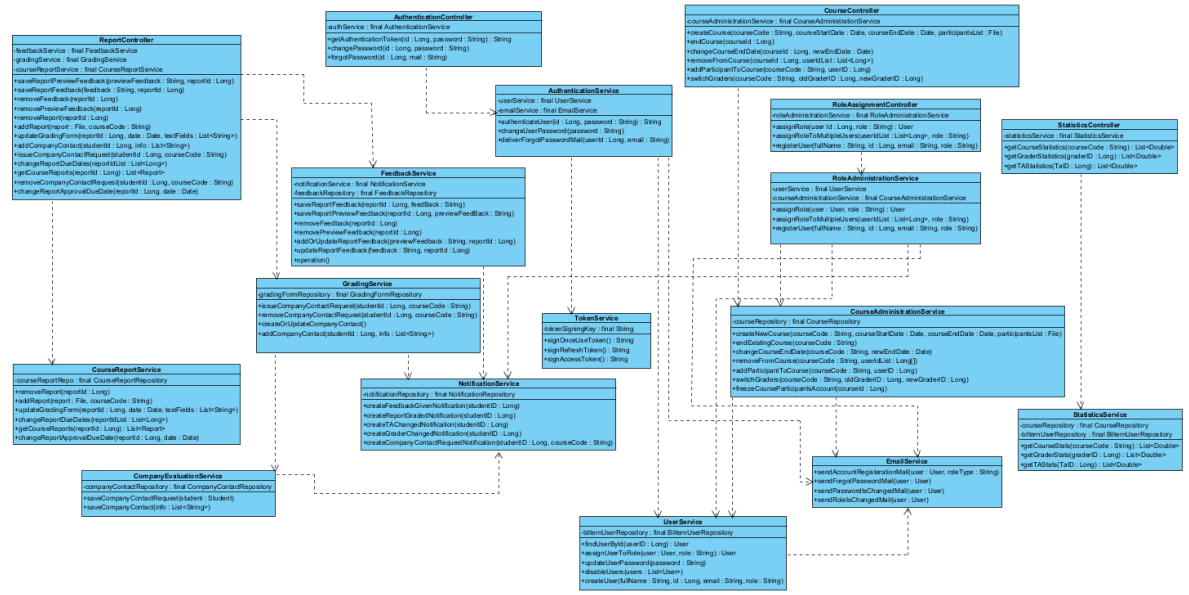

Figure 11. Application Layer Object Diagram

For better resolution:

https://drive.google.com/file/d/14tQCUt9GQ3UZqqKU5i5qrMxhtmMX5OTB/view?usp=sharing

3.3.3 Database Abstraction Layer

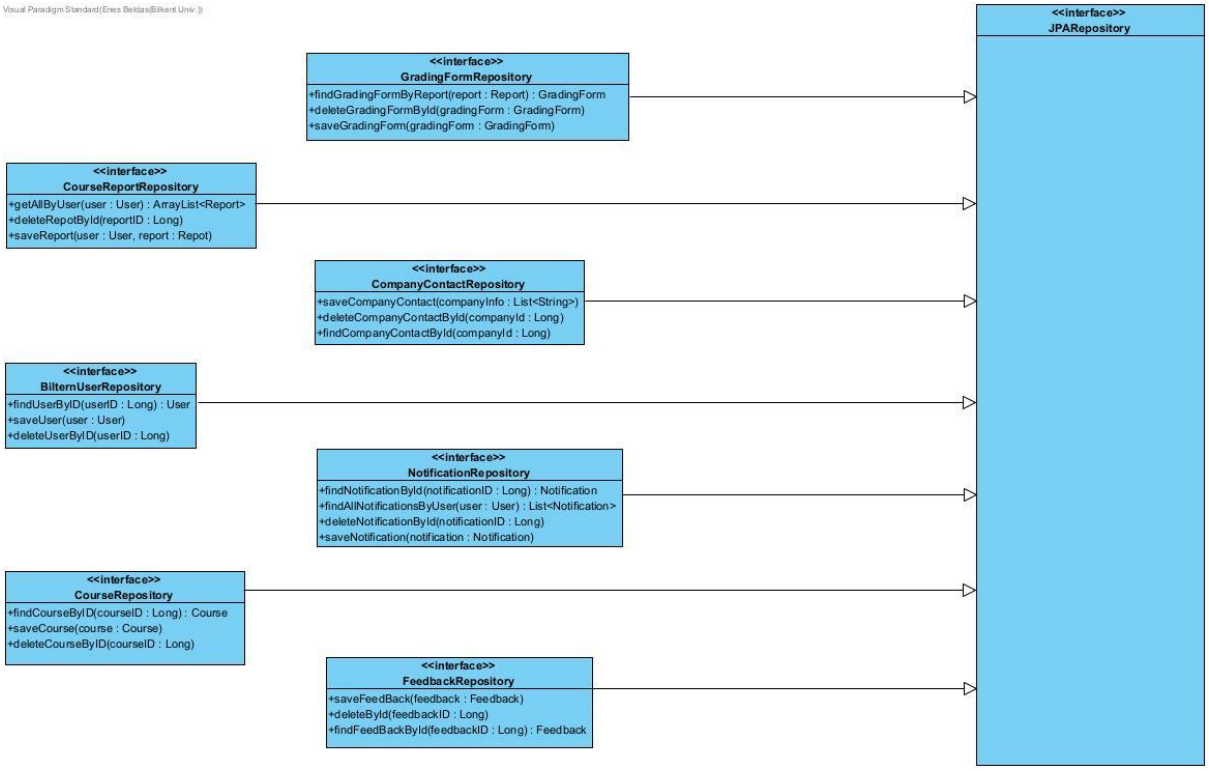

Figure 12. Database Abstraction Object Diagram

3.3.4 Database Layer(entity)

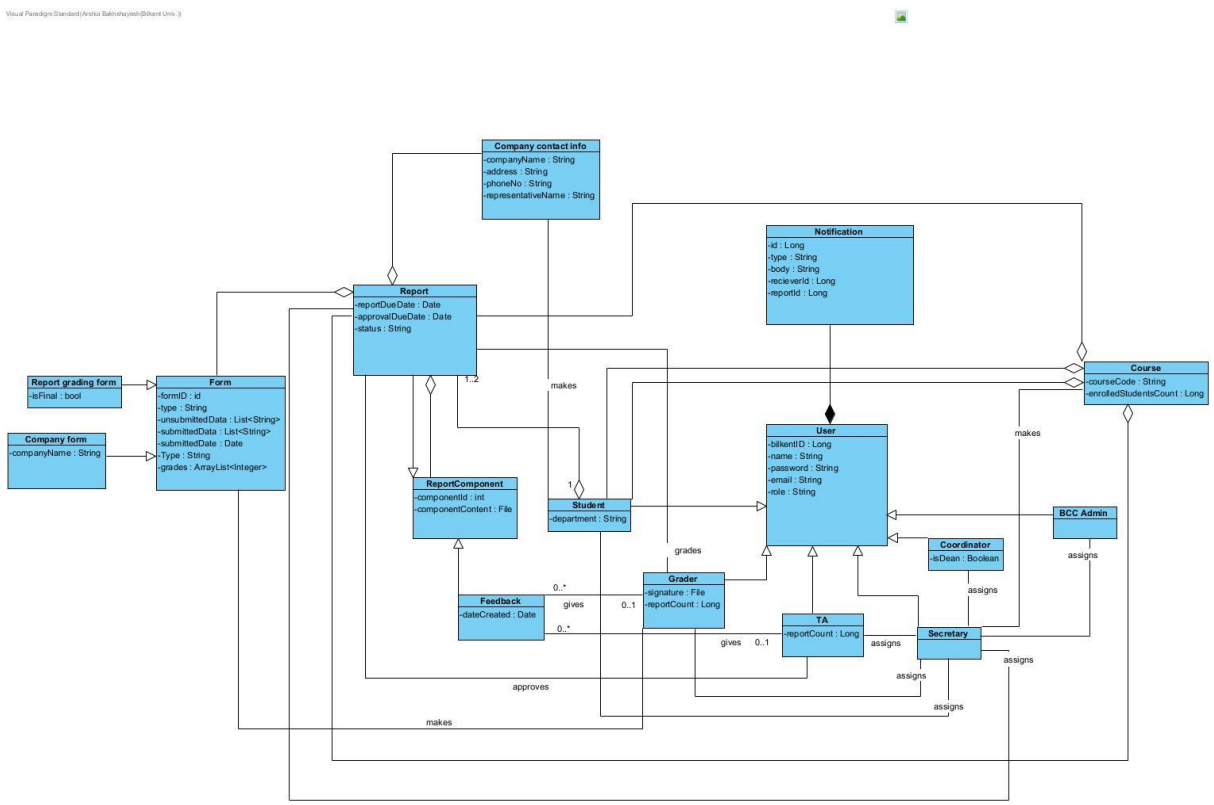

Figure 13. Database Layer Object Diagram

3.4 Class Interfaces

3.4.1 User Interface Layer Class Interfaces

LoginPage

Operations:

login(userID : Long, password : String)
gotoForgotPasswordPage()

ForgotPasswordPage

Operations:

sendForgotPasswordEmail(userID: Long, email: String)
goToLoginPage()

DashboardPage

Operations:

extendNotifications()
updateNotifications()

CoordinatorListPanel

Operations:

fetchCoordinatorList()

MiniSecretaryListPanel

Operations:

fetchSecretaryList()

MiniCurrentStatusPanel

Operations:

goToCurrentStatusPage()
fetchCurrentStatus()

GraderInfoPanel

Operations:

fetchGraderInfo()

StatisticsPanel

Operations:

fetchStatistics()

TAInfoPanel

Operations:

fetchTAInfo()

ReminderPanel

Operations:

fetchReminders()

MiniStudentListPanel

Operations:

goToStudentListPage(studentID : Long)

ReadyToBeGradedPanel

Operations:

goToCurrentStatusPage(studentID : Long)

NotificationsPanel

Operations:

markAsRead(notificationID : Long)
deleteNotification(notificationID : Long)
fetchNotifications()

Sidebar Operations:

Operations:

goToDashboardPage()
goToStudentsListPage()
goToGraderListPage()
goToSystemInitializationPage()
goToUploadedFilesPage()
goToSettingsPage()
goToRoleAssignmentPage()
goToCurrentStatusPage()
logOut(userID : Long)

UploadedFilesPage

Operations:

displayFile(fileID : Long)
fetchUploadedFiles()

SystemInitializationPage

Operations:

initializeSystem(initializationFile : File)
chooseFile(initializationFile : File)

RoleAssignmentPage

Operations:

assignRole(userID : Long, role : String)

StudentListPage

Operations:

logOut(userID : Long)
fetchStudentList()
orderStudentList(orderBy : String)
exportStudentList()
fetchStudentStatistics()

filterStudentList(filterBy : String)
changePageNum(pageNum : int)
changeNumStudentsShown(numStudentsShown : int)
searchFromStudentList(searchKey : String)

GraderListPage

Operations:

fetchGraderList()
orderGraderList(orderBy : String)
exportGraderList()
filterGraderList(filterBy : String)
changePageNumber(pageNum : int)
changeNumberOfGradersShown(numGradersShown : int)
searchFromGraderList(searchKey : String)

TAListPage

Operations:

fetchTAList()
orderTAList(orderBy : String)
exportTAList()
filterTAList(filterBy : String)
changePageNumber(pageNum : int)
changeNumberOfTAsShown(numberOfTAsShown : int)
searchFromTAList(searchKey : String)

CurrentStatusPage

Operations:

showStudentReportPanel()
showEvaluationOfReportPartAPanel()
showCompanyEvaluationReportPanel()
showEvaluationOfReportPartBPanel()
showEvaluationOfReportPartCPanel()
showPreEvaluationPanel()
showSystemCheckPanel()

PreEvaluationPanel

Operations:

downloadFeedback(feedbackID : Long)
uploadFeedback(reportID : Long, feedbackID : Long)
downloadReport(reportID : Long)
markSatisfactory(reportID : Long, studentID : Long)
askForRevision(reportID : Long)

StudentReportPanel

Operations:

uploadReport()
downloadReport()

SystemReportCheckPanel

CompanyEvaluationReportPanel

EvaluationOfReportPartAPanel

Operations:

markUnsatisfactory(studentID : Long, reportID : Long)
markSatisfactory(studentID : Long, reportID : Long)

EvaluationOfReportPartBPanel

Operations:

downloadFeedBack(reportID : Long, feedbackID : Long)
setDeadline(reportID : Long, deadline : Date)
uploadFeedback(reportID : Long, feedbackID : Long)
askForRevision(reportID : Long)
markSatisfactory(studentID : Long, reportID : Long)
extendDeadline(reportID : Long, deadline : Date)

EvaluationOfReportPartCPanel

SettingsPage

Operations:

changeDefaultSignature(signatureID : Long)
changePassword()
uploadSignature(signature : File)
chooseSignature(signature : File)

ChangePasswordPanel

Operations:

changePassword(userID : int, prevPassword : String, newPassword : String)

3.4.2 Application Layer Class Interfaces

ReportController

Attributes:

feedbackService : final FeedbackService

gradingService : final GradingService

courseReportService : final CourseReportService

Operations:

saveReportPreviewFeedback(previewFeedback : String, reportId : Long)

saveReportFeedback(feedback : String, reportId : Long)

removeFeedback(reportId : Long)

removePreviewFeedback(reportId : Long)

removeReport(reportId : Long)

addReport(report : File, courseCode : String)

updateGradingForm(reportId : Long, date : Date, textFields : List<String>)

addCompanyContact(studentId : Long, info : List<String>)

issueCompanyContactRequest(studentId : Long, courseCode : String)

changeReportDueDates(reportIdList : List<Long>)

getCourseReports(reportId : Long) : List<Report>

removeCompanyContactRequest(studentId : Long, courseCode : String)

changeReportApprovalDueDate(reportId : Long, date : Date)

CourseReportService

Attributes:

courseReportRepo : final CourseReportRepository

Operations:

removeReport(reportId : Long)

addReport(report : File, courseCode : String)

updateGradingForm(reportId : Long, date : Date, textFields : List<String>)

changeReportDueDates(reportIdList : List<Long>)

getCourseReports(reportId : Long) : List<Report>

changeReportApprovalDueDate(reportId : Long, date : Date)

CompanyEvaluationService

Attributes:

companyContactRepository : final CompanyContactRepository

Operations:

saveCompanyContactRequest(student : Student)

saveCompanyContact(info : List<String>)

FeedbackService

Attributes:

notificationService : final NotificationService

feedbackRepository : final FeedbackRepository

Operations:

saveReportFeedback(reportId : Long, feedBack : String)

saveReportPreviewFeedback(reportId : Long, previewFeedBack : String)

removeFeedback(reportId : Long)

removePreviewFeedback(reportId : Long)

addOrUpdateReportFeedback(previewFeedback : String, reportId : Long)

updateReportFeedback(feedback : String, reportId : Long)

GradingService

Attributes:

gradingFormRepository : final GradingFormRepository

Operations:

issueCompanyContactRequest(studentId : Long, courseCode : String)

removeCompanyContactRequest(studentId : Long, courseCode : String)

createOrUpdateCompanyContact()

addCompanyContact(studentId : Long, info : List<String>)

NotificationService

Attributes:

notificationRepository : final NotificationRepository

Operations:

createFeedbackGivenNotification(studentID : Long)

createReportGradedNotification(studentID : Long)

createTAChangedNotification(studentID : Long)

createGraderChangedNotification(studentID : Long)

createCompanyContactRequestNotification(studentID : Long, courseCode : String)

AuthenticationController

Attributes:

authService : final AuthenticationService

Operations:

String getAuthenticationToken(id : Long, password : String)

changePassword(id : Long, password : String)

forgotPassword(id : Long, mail : String)

AuthenticationService

Attributes:

userService : final UserService

emailService : final EmailService

Operations:

authenticateUser(id : Long, password : String) : String

changeUserPassword(password : String)

deliverForgotPasswordMail(userId : Long, email : String)

TokenService

Attributes:

tokenSigningKey : final String

Operations:

signOnceUseToken() : String

signRefreshToken() : String

signAccessToken() : String

CourseController

Attributes:

courseAdministrationService : final CourseAdministrationService

Operations:

createCourse(courseCode : String, courseStartDate : Date, courseEndDate : Date, participantsList : File)

endCourse(courseId : Long)

changeCourseEndDate(courseId : Long, newEndDate : Date)

removeFromCourse(courseId : Long, userIdList : List<Long>)

addParticipantToCourse(courseCode : String, userID : Long)

switchGraders(courseCode : String, oldGraderID : Long, newGraderID : Long)

RoleAssignmentController

Attributes:

roleAdministrationService : final RoleAdministrationService

Operations:

assignRole(user Id : Long, role : String) : User

assignRoleToMultipleUsers(userIdList : List<Long>, role : String)

registerUser(fullName : String, id : Long, email : String, role : String)

RoleAdministrationService

Attributes:

userService : final UserService

courseAdministrationService : final CourseAdministrationService

Operations:

assignRole(user : User, role : String) : User

assignRoleToMultipleUsers(userIdList : List<Long>, role : String)

registerUser(fullName : String, id : Long, email : String, role : String)

CourseAdministrationService

Attributes:

courseRepository : final CourseRepository

Operations:

createNewCourse(courseCode : String, courseStartDate : Date, courseEndDate : Date, participantsList : File)

endExistingCourse(courseCode : String)

changeCourseEndDate(courseCode : String, newEndDate : Date)

removeFromCourse(courseCode : String, userIdList : Long[])

addParticipantToCourse(courseCode : String, userID : Long)

switchGraders(courseCode : String, oldGraderID : Long, newGraderID : Long)

freezeCourseParticipantsAccount(courseId : Long)

UserService

Attributes:

bilternUserRepository : final BilternUserRepository

Operations:

findUserById(userID : Long) : User

assignUserToRole(user : User, role : String) : User

updateUserPassword(password : String)

disableUsers(users : List<User>)

createUser(fullName : String, id : Long, email : String, role : String)

EmailService

Operations:

sendAccountRegisterationMail(user : User, roleType : String)

sendForgotPasswordMail(user : User)

sendPasswordIsChangedMail(user : User)

sendRoleIsChangedMail(user : User)

StatisticsController

Attributes:

statisticsService : final StatisticsService

Operations:

getCourseStatistics(courseCode : String) : List<Double>

getGraderStatistics(graderID : Long) : List<Double>

getTAStatistics(TaID : Long) : List<Double>

StatisticsService

Attributes:

courseRepository : final CourseRepository

bilternUserRepository : final BilternUserRepository

Operations:

getCourseStats(courseCode : String) : List<Double>

getGraderStats(graderID : Long) : List<Double>

getTAStats(TaID : Long) : List<Double>

3.4.3 Database Abstraction Layer Class Interfaces

GradingFormRepository

Operations:

findGradingFormByReport(report : Report) : GradingForm

deleteGradingFormById(gradingForm : GradingForm)

saveGradingForm(gradingForm : GradingForm)

CourseReportRepository

Operations:

getAllByUser(user : User) : ArrayList<Report>

deleteRepotById(reportID : Long)

saveReport(user : User, report : Repot)

CompanyContactRepository

Operations:

saveCompanyContact(companyInfo : List<String>)

deleteCompanyContactById(companyId : Long)

findCompanyContactById(companyId : Long)

BilternUserRepository

Operations:

findUserByID(userID : Long) : User

saveUser(user : User)

deleteUserByID(userID : Long)

NotificationRepository

Operations:

findNotificationById(notificationID : Long) : Notification

findAllNotificationsByUser(user : User) : List<Notification>

deleteNotificationById(notificationID : Long)

saveNotification(notification : Notification)

CourseRepository

Operations:

findCourseByID(courseID : Long) : Course

saveCourse(course : Course)

deleteCourseByID(courseID : Long)

FeedbackRepository

Operations:

saveFeedBack(feedback : Feedback)

deleteById(feedbackID : Long)

findFeedBackById(feedbackID : Long) : Feedback

3.5 Design Patterns

3.5.1 Composite pattern

For the report and feedback objects, the composite pattern was used. Each report can contain multiple unbounded iterations which are reports themselves. Each report will also contain feedback besides multiple iterations. This leads to a tree-like structure shown below:

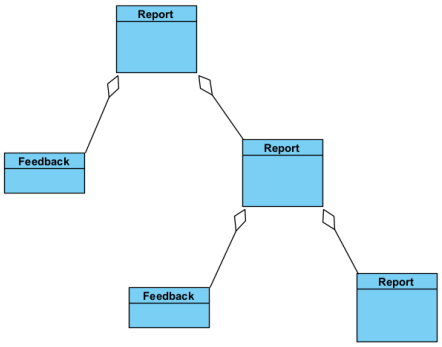

Figure 14. Tree representation of report components

The composite design pattern was applied to allow the uniform treatment of report iterations and report feedback as they both contain a PDF and their respective operations in the services.

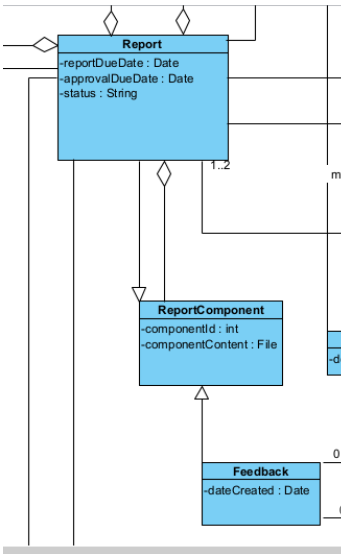

Figure 15. Composite report representation

3.5.2 Facade pattern

In our project, we will use the Facade pattern on the data layer and data abstraction layer. In particular, the user service will act as a “maker” for deciding which user instance should be created from different user types. The connection of the service to the user entity is done through the repository. This hides or in other words, abstracts the details of direct interactions through the database.

Improvements Summary

- The deployment diagram has been added.
- Explanations of the subsystem decomposition and user interface layer are improved.
- -The design goals we chose for our project are explained better including more details and their specific uses in our project are mentioned.
- Boundary conditions are improved with more transition conditions and added further explanatory bullet points for the states of the program.
- Replaced Proxy design pattern with Composite design pattern. Used diagrams to clearly indicate how we integrated it into our project.
- Fixed the cut-off application layer diagram.
-

4 References

[1] “Top Reasons to Use MySQL,” Databasequest.com, 2012, https://www.databasequest.com/index.php/product-service/mysql-dbquest [Online]

[2] “Mozilla Developer Network. (n.d.). Web technology for developers”. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web. [Accessed: May. 4, 2023].

[3] “Amazon Web Services. (n.d.).” Documentation. [Online]. https://docs.aws.amazon.com/. [Accessed: May. 4, 2023].