Bilkent University

Department of Computer Science

CS 319: Object-Oriented Software Engineering

INTERNSHIP MANAGEMENT SYSTEM

Design Report

Second Iteration

21 May 2023

RIGEL

Aytekin İsmail, 22003988

Ece Ateş, 22002908

İzgi Nur Tamcı, 22002682

Ömer Asım Doğan, 21903042

Zeynep Begüm Kara, 22003880

1. Introduction

Bilkent University requires second and third-year engineering students to participate in a summer training program for twenty workdays, aiming to develop students both academically and professionally. The program involves several steps, including registering with internship companies and completing x299 or x399 courses in the related department [1]. Approximately 1300 students take these courses each fall semester. However, managing a large number of students and university policies poses a significant organizational burden on department secretaries.

Students are required to write a report detailing their learning and contributions during the summer training, ensuring their engagement in the professional experience and evaluating the reliability of the companies for future candidates. Instructors evaluate these internship reports based on ABET criteria. Students whose reports do not meet all the criteria may need to revise their work multiple times under the guidance of their assigned instructor. Finally, students receive a satisfactory or unsatisfactory status based on their summer training and x299/399 course performance.

The purpose of developing the Rigel Internship Evaluation System is to create a web-based application that provides convenience for department secretaries, evaluator instructors, and registered students. The system aims to streamline the organization by integrating all aspects of the process into one platform.

This report introduces Rigel software for Bilkent Engineering Department internship management by stating its purpose and design goals, providing its high-level software architecture and low-level design components. The low-level design will provide packages, class interfaces, and design patterns. High-level software architecture consists of subsystem decomposition, hardware/software mapping, persistent data management, access control, and boundary conditions. This report concludes with a glossary and references.

1.1 Purpose of the System

The internship management system heavily relies on paperwork for both instructors and the administration. Initially, students bring an evaluation report, known as the Summer Training Evaluation Form (Evaluation Form), from the company. This report contains the supervisor's comments and the grading of the student's performance during the internship. It is considered confidential and must be submitted as a sealed document or directly from the company email. The department secretary uses this report to fill in the first part of the grade report.

Students are required to write and upload an internship report summarizing their work during the internship. Instructors provide feedback to the students throughout the semester and may request revisions. The department secretary sets deadlines for the report revisions, except for the initial deadline, which the department chair sets. Students have two weeks to edit their report, explain the changes made, and send it to the instructor for further review. Revisions must be completed before the end of the semester, or the student's report will be

considered unsatisfactory. Instructors may also deem the report unsatisfactory even after revisions, failing the student during the semester. Students who receive an unsatisfactory grade have to retake the class next term. For those who got a satisfactory grade from their internship report, instructors will fill in a criteria report indicating their grading of student reports. The last operation in a semester that Rigel software must cover is ensuring that both student reports and criteria reports are submitted.

Rigel software aims to create an online platform for internship report management for Bilkent Engineering Department. Previously, the instructors had to download student reports and give feedback about them from their computers. Rigel offers a web application to handle reports without downloading, enabling actors to complete tasks and track their progress through a single application. The expected users are students who take x299 and x399 courses, instructors, department chair, department secretary, and admin for technical maintenance purposes. The system will also provide announcements, instructor statistics, and student-instructor assignments aside from its report management functionalities.

1.2 Design Goals

Non-functional requirements for this application are narrowed down due to limited development time and budget. According to the application domain and the preliminary research, the main design goals for this application are usability and functionality.

1.2.1 Usability

Rigel software is designed with minimal web pages to handle all desired functionalities. The application aims to guide users to their desired task webpage within 4 clicks. UI (user interface) is intuitive to its actors as the basis for the design is the STARS system. Each actor will have access to all of their functionalities from their main page either from the navigation bar or drop-down menu and they can familiarize themselves with the system within 10 minutes.

1.2.2 Functionality

Rigel software aims to offer report management from the first report submitted to the instructor’s approval of the final report. Therefore, multiple actors will take part during the different stages of this process. For instance, the department secretary will be able to enter company grades into the system so that instructors will be able to start Part B of the report evaluation form. The organization of these multi-actor tasks will be handled by the system. It will also provide automatically generated forms for the instructors and they can save their progress in these forms. There are a number of novel features such as displaying the statistics of the instructors, requesting and granting extensions, and giving feedback without downloading reports within the web application.

2. High-Level Software Architecture

2.1 Subsystem Decomposition

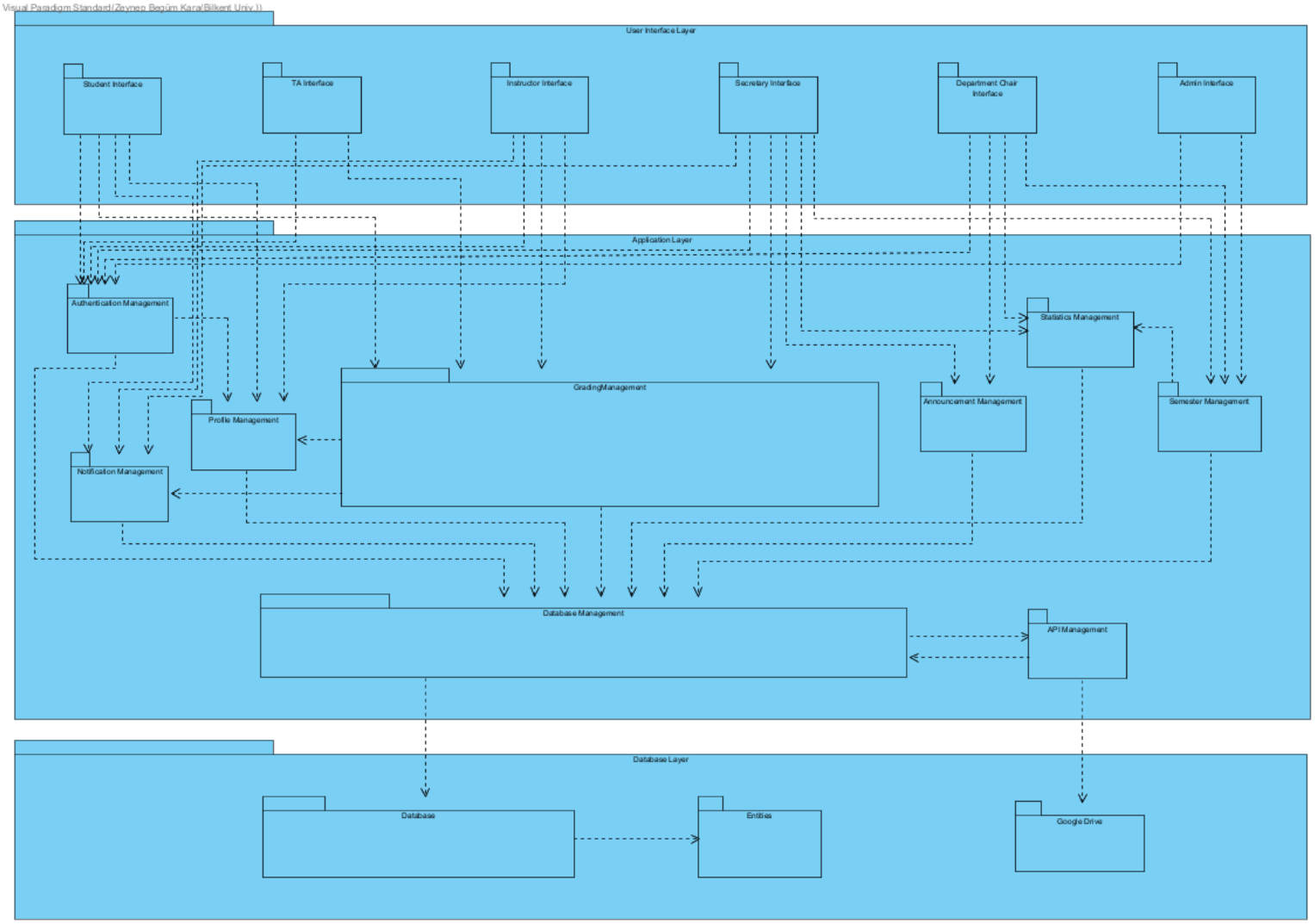

Figure 1: Subsystem Decomposition Diagram

Diagram link: https://imgur.com/a/kwF1SQt

2.1.1 User Interface Layer

The User Interface Layer, the first layer in Figure 1, includes the database and Google Drive Folder. the interface between users and screens, allowing them to interact with a program. Users can input information into designated fields, and this information is then transmitted to the Application Layer. Observe that each actor has a different interface and those interfaces consist of screens and functionalities of the related actor.

2.1.2 Application Layer

The Application Layer, the second layer in Figure 1, consists of subsystems that includes control objects that are capable of modifying data based on the retrieved information from the Presentation Layer. These control object packages interact with each other and the Database Manager.

2.1.2.1 Authentication Management Subsystem

Authentication Management Subsystem consists of control classes related to user authentication management.

2.1.2.2 Notification Management Subsystem

Notification Management Subsystem consists of control classes related to user notification management. Notice that this subsystem is related to Grading Management Subsystem, so that when a state of the changes, users can be notified.

2.1.2.3 Announcement Management Subsystem

Announcement Management Subsystem consists of control classes related to user notification management.

2.1.2.4 Grading Management Subsystem

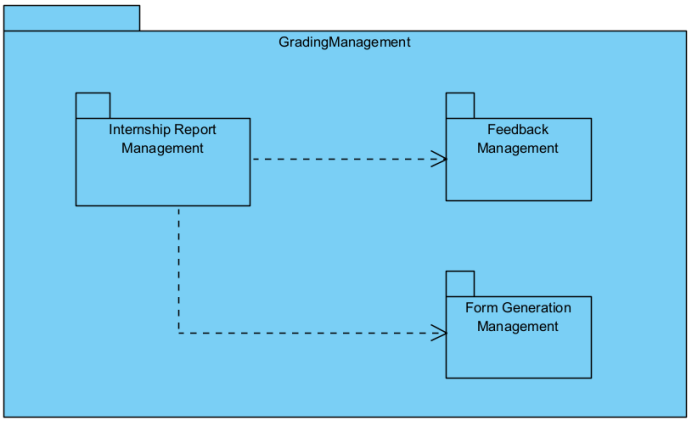

Figure 2: Grading Management Subsystem Diagram

Grading Management Subsystem consists of three individual subsystems that it owns. The internship Report Management subsystem handles operations like internship report uploads, submission deadlines, asking for extensions, and extending deadlines. Feedback Management subsystem is related to the Internship Report Management system and performs tasks such as asking revisions, initial TA checks, plagiarism detection, and giving feedback. Form Generation Management subsystem handles tasks such as filling criteria reports, generating the criteria report PDFs, completing the evaluation form, generating grade report PDFs,s and e-signing the grade form.

2.1.2.5 Statistics Management Subsystem

Statistics Management Subsystem consists of control classes related to statistics generation for ABET. Including the grade distribution of Criteria Report questions. Based on this data, evaluation statistics of students according to their registered department, faculty, and instructor are generated at the end of each semester. These statistics include the evaluation of the work and evaluation of the report of the Internship Report.

2.1.2.6 Semester Management Subsystem

Semester Management Subsystem consists of control classes related to semester management. This subsystem handles tasks such as setting semester dates based on the academic calendar, creating user accounts, and assigning students to instructors.

2.1.2.7 Profile Management Subsystem

Profile Management Subsystem consists of control classes related to the profile. This subsystem handles e-signature functionalities and tracks the progress status of users. In the case of instructors, the progress status indicates the number of students remaining to be graded. For students, the progress status reflects the current state of their report (as depicted in the entity diagram).

2.1.3 Data Layer

The Data Layer, the third layer in Figure 1, involves the database and Google Drive Folder. The database stores the properties of entity objects, while the Google Drive Folder stores uploaded internship reports from students and automatically generated PDFs. These PDFs include Grade Forms and Criteria Reports for each student and statistics for ABET generated at the end of the semester for documentation and archiving purposes. The links to the PDF documents are stored in the relevant properties of the entity objects within the database.

2.2 Hardware / Software Mapping

We will use React.js with Javascript for the front end of our project. We will also use HTML5 and CSS3. Considering that modern browsers support all of the mentioned software, it can be said that our project will only need a stable modern browser, which is minimal demand for a website application.

We expect about 1.000 students to use our system in one semester. Considering everyone else included in this process, it is safe to assume less than 2.000 people will use our website in total, but this number decreases to about 100 users if we consider the number of users that will use the website concurrently [1]. For these reasons, in our backend, we will use Java, Spring Boot, MongoDB, and Google Drive API. Google Drive can sustain up to 50.000 requests per project per day and 10 queries per second per IP address, which is quite enough for the requirements of our program, where most users won’t use it daily (except

maybe Instructors who will review and grade student reports) [2]. Considering our program's simple requirements, we've decided that the Intel Core i5, or AMD Ryzen processor, which are some of the most common processors currently, would be enough and well-equipped to run our application. Since these processors are already highly advanced, they can easily handle the website even if our application's requirements increase in the future. Otherwise, our website does not require any additional hardware. We believe that it can be run on both computers and mobile phones (that can run a modern browser) without any problems.

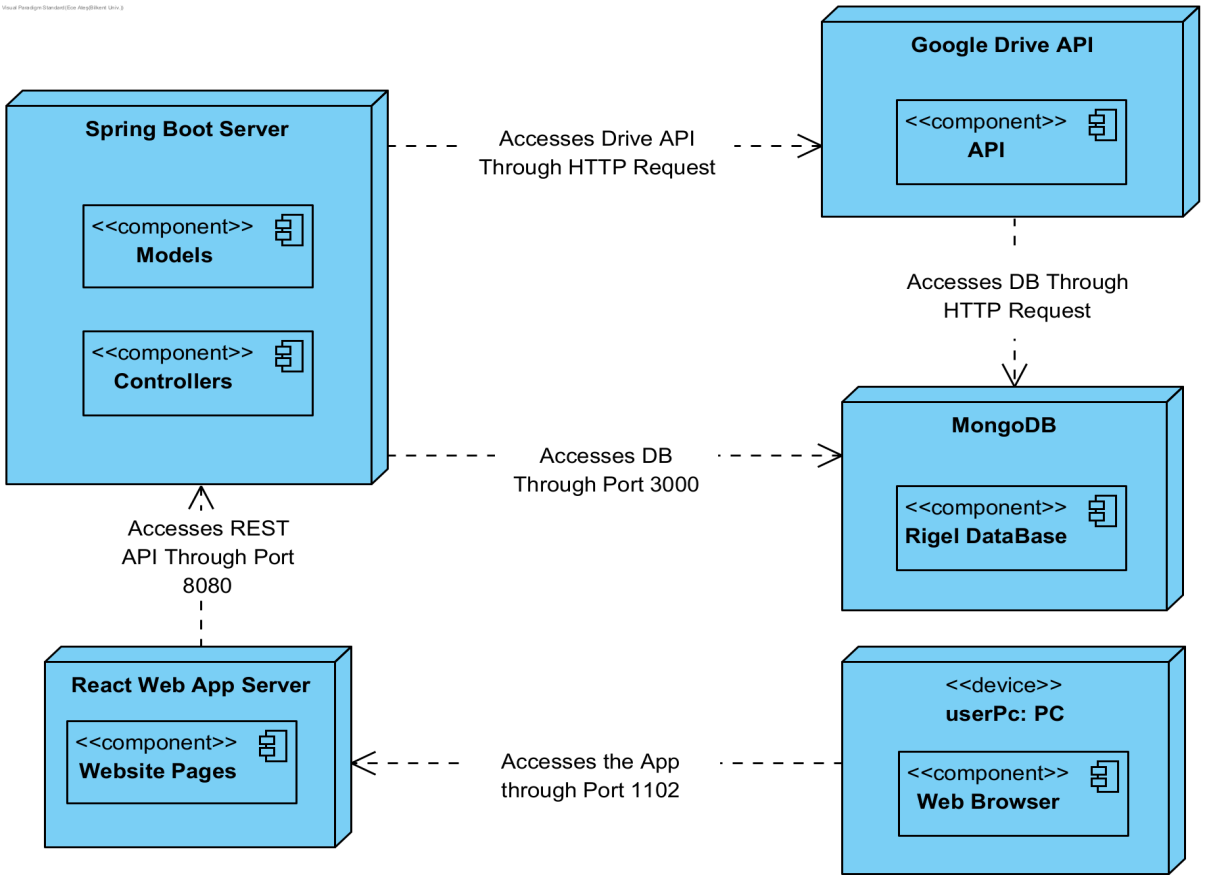

Figure 3: A diagram showing the relations between the program's hardware and software components.

As can be seen in the above diagram, our software interacts with both Google Drive API services and the MongoDB database. Also, Google Drive API will use MongoDB to reach information about the users.

2.3 Persistent Data Management

Since our program is mostly about editing, uploading, and downloading different kinds of reports and reviews, we made file reaching a priority. We've decided to use Google Drive to store the relevant files. The free version of Google Drive allows us to store 400.000 items, including files, folders, and shortcuts, and it gives a total of 30GB of storage area [2] [3]. Considering that on average a 35-page report pdf (which is an acceptable amount of pages for a student's internship report) is about 1.5 MB = 0.0015 GB, we can store about 20.000 such files. This number isn't enough to hold all the reports for all the semesters, but it is

enough to store the files for one semester. When the semester ends, the files can be transferred into another, larger database for achieving purposes, thus clearing the space for the next semester.

We've decided to keep the user information and the basic information of the reports on MongoDB due to its high performance, fault tolerance, and ease of use. Since MongoDB can run on several servers, it also has high reliability. Since it replicates the data across those servers, it is relatively easy to recover data in the case of a crash. The data we will store on MongoDB will be the essential information of the users, such as email, password, etc. Also, we will store a basic representation of the reports as objects. They will have data about the report deadline, current status, instructor, etc.

2.4 Access Control and Security

In our Rigel Internship Report Management System, we emphasize the security of the student and administration's information and thus set an access control system. The access control system is based on the roles of the users which are given either by the admin or department secretary. Among these roles, there are unregistered users, students, instructors, TA, department secretary, department chair, and admin. As mentioned above, the admin is responsible for creating the user and assigning the role of department secretary for each engineering department. Afterward, each department's secretary creates users with the assigned student, instructor, department chair, and TA roles.

On the client side, our plan is to allow access to different UI to their users with roles by protected routing. When a user logs in, they are led through their own pacific routes according to their access permission based on their role. For example, a student would not have access to UI where the instructor sees all of the submitted reports.

On the backend side, the system logs in the user based on the roles that were given to them in the sign-up. Therefore, this role attribute is sent to the system's logic which ensures that certain features are functional only in their appropriate users' interfaces. For example, while similar student lists are shown to both instructors and department chairs, only the latter would have the functionality of withdrawing the student.

When it comes to access control to our database and backend code, since we plan to implement our system with STARS, we opted to not use any firewall in our security architecture. We aim to run the project on the school's local servers, thus providing a certain level of protection to our backend code automatically. Also, MongoDB automatically encrypts the data before storing it. MongoDB ensures security by strong authentication with a username/password for access to the database which better the security overall.

	Student	Admin	Department Chair	Secretary	Instructor	TA
Login/Logout	+	+	+	+	+	+
Announcement Page / Public Page	+	+	+	+	+	+

Main Page	+	+	+	+	+	+
View Profile	+				+	
Create Semester		+				
Start Courses				+		
Make Announcement		+	+	+	+	
Save/Submit Internship Report	+					
Check Submitted Report						+
View Internship Report	+		+	+	+	+
Evaluate Report					+	
Ask Extension	+					
Give Extension					+	
Create / Delete Department Secretary Accounts		+				
Create / Delete Department Chair Accounts		+		+		
Create / Delete Instructor Accounts		+		+		
Create / Delete Student Accounts		+		+		
Create / Delete Student Accounts		+		+		
View List of Users		+	+	+	+	

Change Student-Instructor Pairings				+		
Change Withdrawal Status					+	
View Calendar	+					
View Progress	+					
Enter Company Evaluation Scores				+		
View Summer Training Grade Form			+	+		
View Criteria Report			+	+		
View Statistics			+		+	
Notifications	+			+	+	

Figure 4: Use Case - Actors Relation Table

2.5 Boundary Conditions

Boundary conditions are a set of constraints that determine our web application's behavior at the edges of its usage. There are multiple types of conditions that push our system to these boundaries:

1) **Network Connectivity Issues:** Our application's requirements related to minimum network bandwidth and maximum network latency should be specified so that users can have a smooth experience. Our application aims to let users transfer and edit files while the rest of the information exchange will happen in forms in the application's interfaces. Therefore, 0.5 Mbps should be enough to exchange maximum-sized data of 4-5 pages which is 10-15 KB, between the user and the servers. Another important requirement is network latency as it affects the page load-time, as well as making the uploading process faster or slower. Therefore, considering instructors would need to edit the file in real-time within the application and students would need to upload their reports fast and easily, network latency should be less than 200 milliseconds. Conditions falling below these requirements would not result in failures but poor performance and user experience.
2) **Timeouts and Failures:** Related to network issues, timeouts, and failures can happen due to network bandwidth and latency among other causes when performing certain operations. One of these operations may be HTTP requests where the server may take longer than 10 seconds to respond and the system aborts the request, displaying a related error message. Apart from timeouts, failures can also happen due to scalability issues where servers cannot handle heavy loads with more than expected 100 concurrent users and their requests. For instance, these timeouts or direct failures can occur if 200 students try to upload their reports right before the deadline time or they go over the expected word count in their reports leading to too large of file size for the minimum network bandwidth of the application. Similarly, issues related to databases and APIs can cause timeouts and failures. For instance, if there are some hardware issues in database servers like disk failure, the data can become unavailable or even lost. But hopefully, since MongoDB is a reliable database, this boundary condition is very unlikely to happen.
3) **Drive API Connection Issues:** Similarly, we plan to use Google Drive API both within our application UI as a pdf editor and database for storing internship reports. Therefore, it is crucial to make sure that Google Drive API connection is made with correct credentials and running before initializing the system so that the users can upload or access the reports. During the stable state of the system, the issue might be throttling and rate of Google Drive API limiting which limits the number of requests to 100 per hour per project [2]. Additionally, the API may limit the amount of data that can be retrieved or modified in a single request or over a given period of time. Therefore, in cases of overload of users, a "Quota Exceeded" error may be received leading to system failures. This failure would lead to an important data loss due to the ungraceful termination of the system. Therefore, periodic data

saving to the API will be implemented so that the data of instructors are protected against crashes.

4) **MongoDB Issues:** Two main concerns related to MongoDB are performance and database writing issues. When the system reaches its maximum number of users, queries might slow down which would cause conflict with React rendering of UI. Also, when writing to the database, power outages or network issues might cut out the process which would lead to inconsistent data.

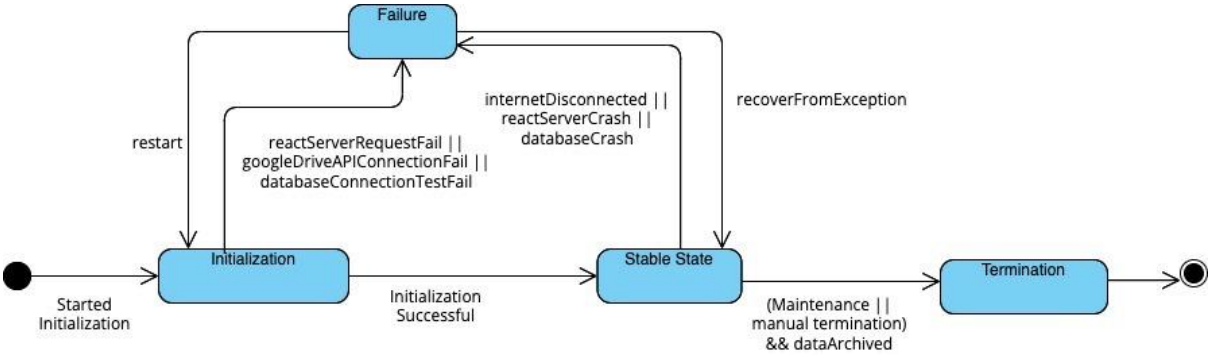

Figure 5: Program Lifecycle State Diagram

2.5.1 Initialization

For the initialization of our application, RigelV1Application should be run from IDE, and “http://localhost:3000” address should be opened in the browser. While initializing the application, the boundary conditions explained above are checked. Google Drive APIs, their connection with correct credentials, and running will be ensured. On the server side, a good internet connection will be ensured in initialization so that there will be no failures related to HTTP requests, file upload, or manipulation. Lastly, before initialization, we will make sure that the database is connected with appropriate credentials and running by implementing a database connection test before this state.

2.5.2 Termination

The admin user may terminate the system. They may terminate the system for maintenance or when the semester is over, ensuring that all of the students’ reports, instructors’ evaluations, and statistics are backed up to the university’s database.

In cases where recovery strategies did not work and termination happened with a complete shutdown, the data will have been saved by periodic auto-saving and alerts will be given to the admin user for quick action and maintenance.

2.5.3 Failure

For failures that happen in initialization and stable state, the recovery strategies will be applied. One of them is rollback procedures where the application will go back to its previous version. For instance, as seen in the diagram, initialization will occur again in case of failures. Or if it happens in the stable state, the operation will be reversed so that the system recovers back to the stable state. For instance, if a file cannot be uploaded, the application will roll back to the version without any upload attempt. Moreover, alerts will be sent to the admin user related to the failure so that rapid development is possible.

3. Low-Level Design

3.1 Object Design Trade-offs

Due to limited development time and budget, Rigel software only focuses on implementing a few design goals. In order to create the best experience while using our website we had to decide upon certain design goals to focus on, which caused other elements to become lacking.

3.1.1 Usability vs. Security

In order to provide a smoother experience to our users, we've chosen usability over security. There are various examples of this trade-off in our program. For example, we've chosen not to include a "two-factor authentication" system (which would have increased our program's security) in our login page to allow users to enter faster and more easily to sign in to their accounts (increased usability). Moreover, we've decided to remove the automatic timeout logout functionality. Again, This feature would have increased our security immensely, but we've chosen not to include it to allow users to continue using our website whenever they like (increased usability). Finally, our decision to integrate Google Drive Interface increases our website's usability since users can now use our website without needing to learn any additional functionality. However, it also causes a security hazard since the link to Google Docs can be seized by malicious people.

3.1.2 Security vs. Automation

To improve our system's security, we haven't automated some steps. For instance, the department secretary has to enter company grades manually because company reports are confidential documents; however, automating this step would significantly speed up the process. Only authorized people in Bilkent, like the department secretary, can view confidential reports. In addition, all the new users must be signed up by either the department secretary or admin to the system. We assume that this manual sign-up ensures that users will be from the Bilkent Engineering Department, which again decreases the automation of our system and the pace of the whole process

3.1.3 Usability vs. Maintenance

In order to provide our users with an easy-to-use program, we have implemented some functionalities that increase our system's usability; however, these same functionalities sometimes diminish our code's maintainability. For example, the system uses the Google Drive API to handle PDF uploading and feedback operations. By leveraging the Drive API, the usability of our system is greatly enhanced, as it offers a plethora of functionalities that are easy to use. However, this also means that our system heavily depends on another system, and if any changes occur in the Google Drive API, our system will also require changes. This feature of our program offers a lot of intuitive functionality to the user, but it greatly increases the workload of the developers

3.2 Final Object Design

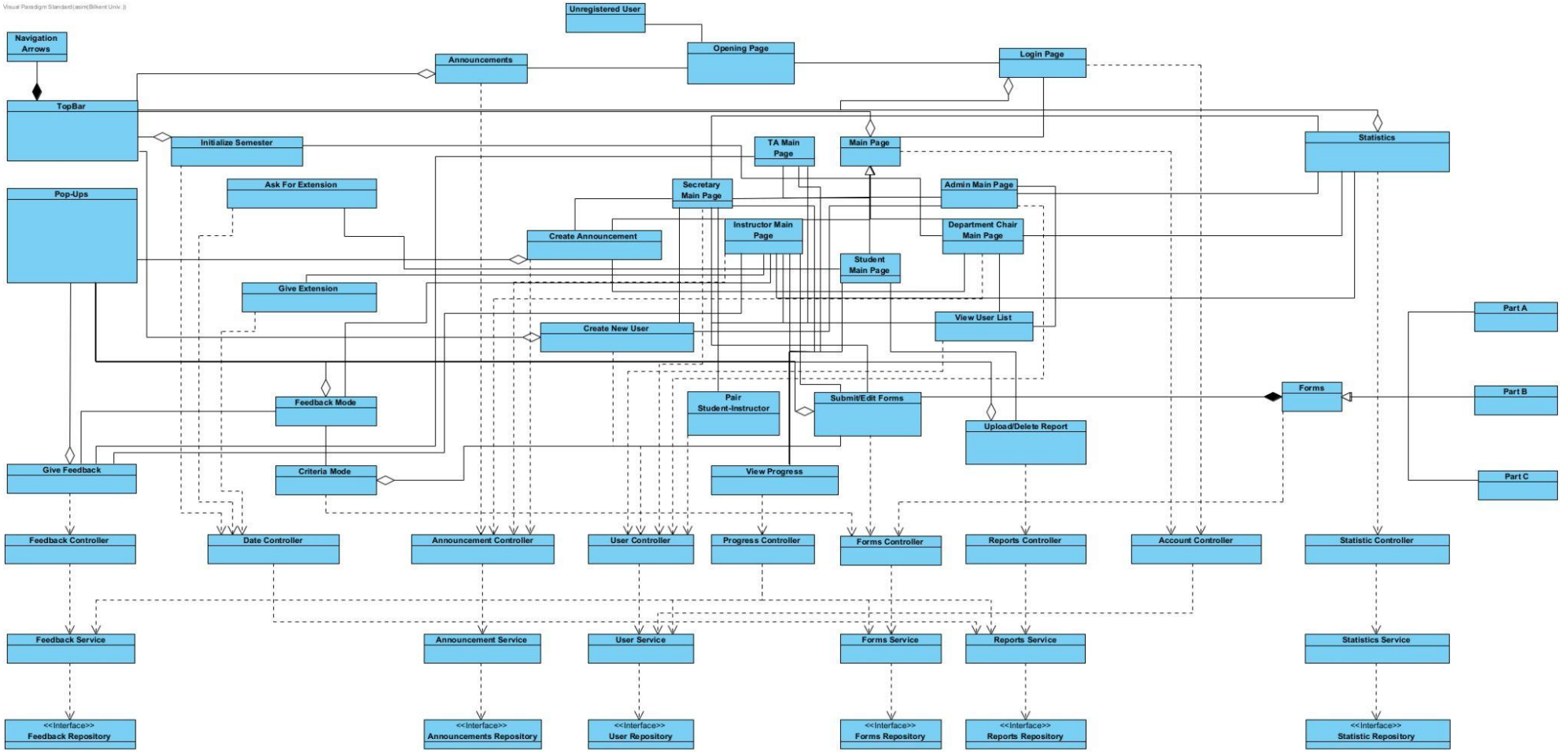

Figure 6: Final Object Design Diagram

Diagram link: https://imgur.com/SpDqowl

3.3 Layers

3.3.1. User Interface Management Layer

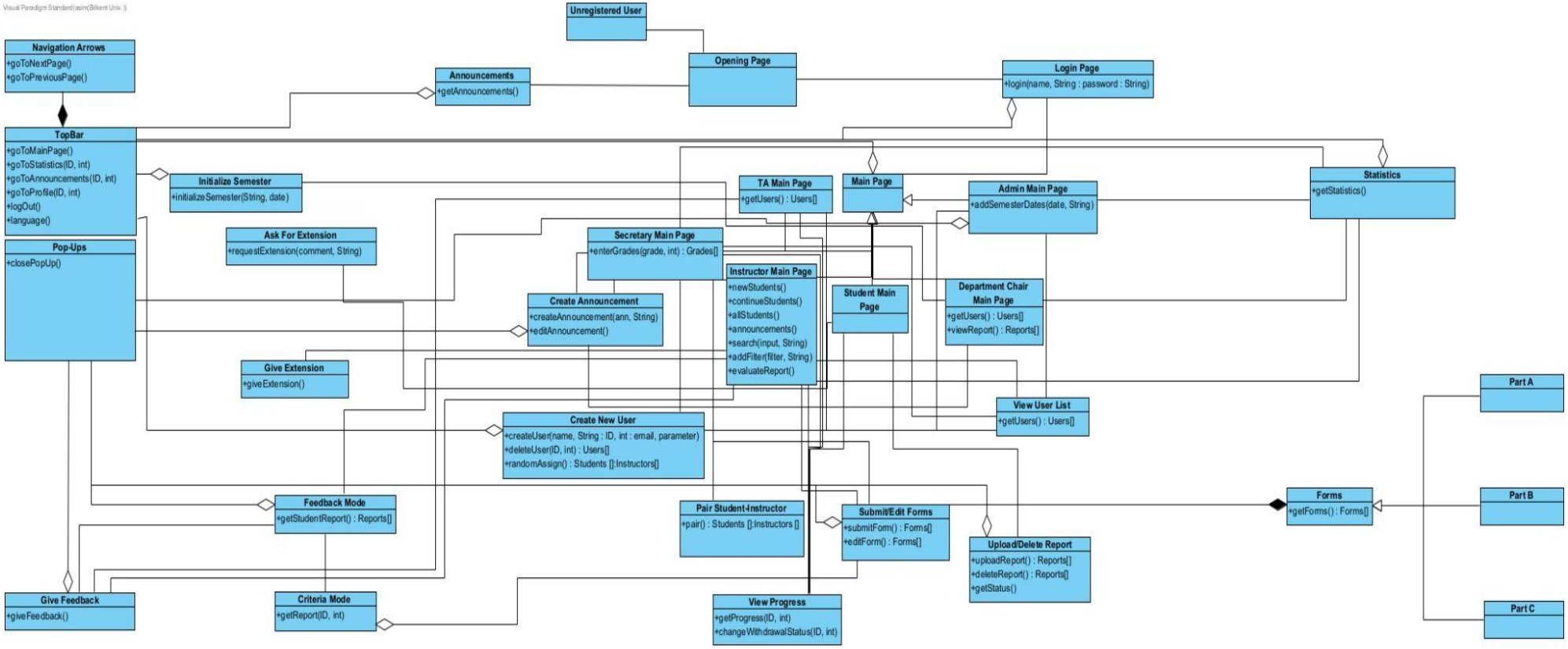

Figure 7: User Interface Management Layer Diagram

Diagram link: https://imgur.com/x12Np0p

3.3.2. Service Management Layer

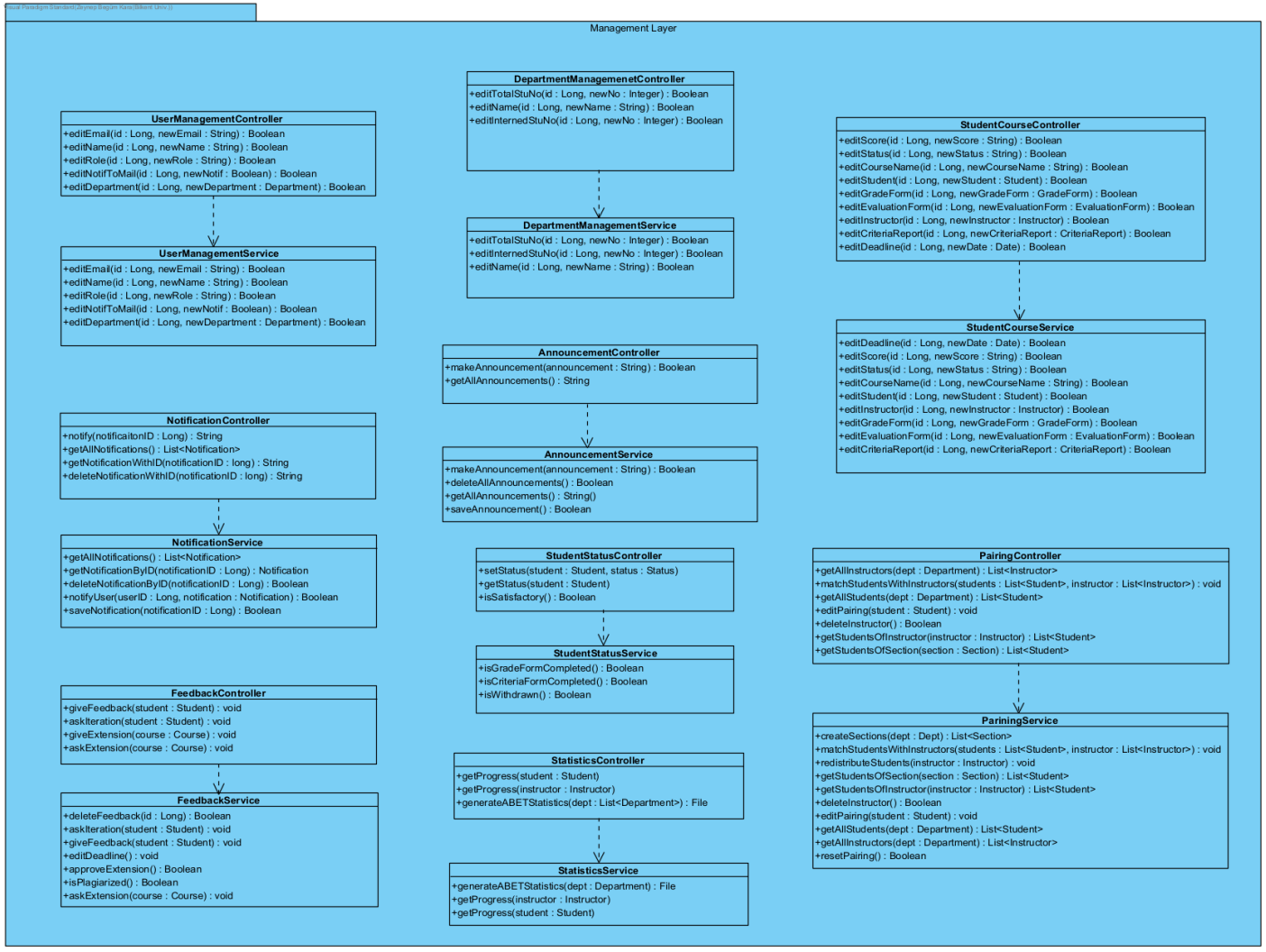

Figure 8: Service Management Layer Diagram

Diagram link: https://imgur.com/a/xvd6bf4

3.3.3. Data Management Layer

3.3.3.1 Data Management Layer: Entity Diagram

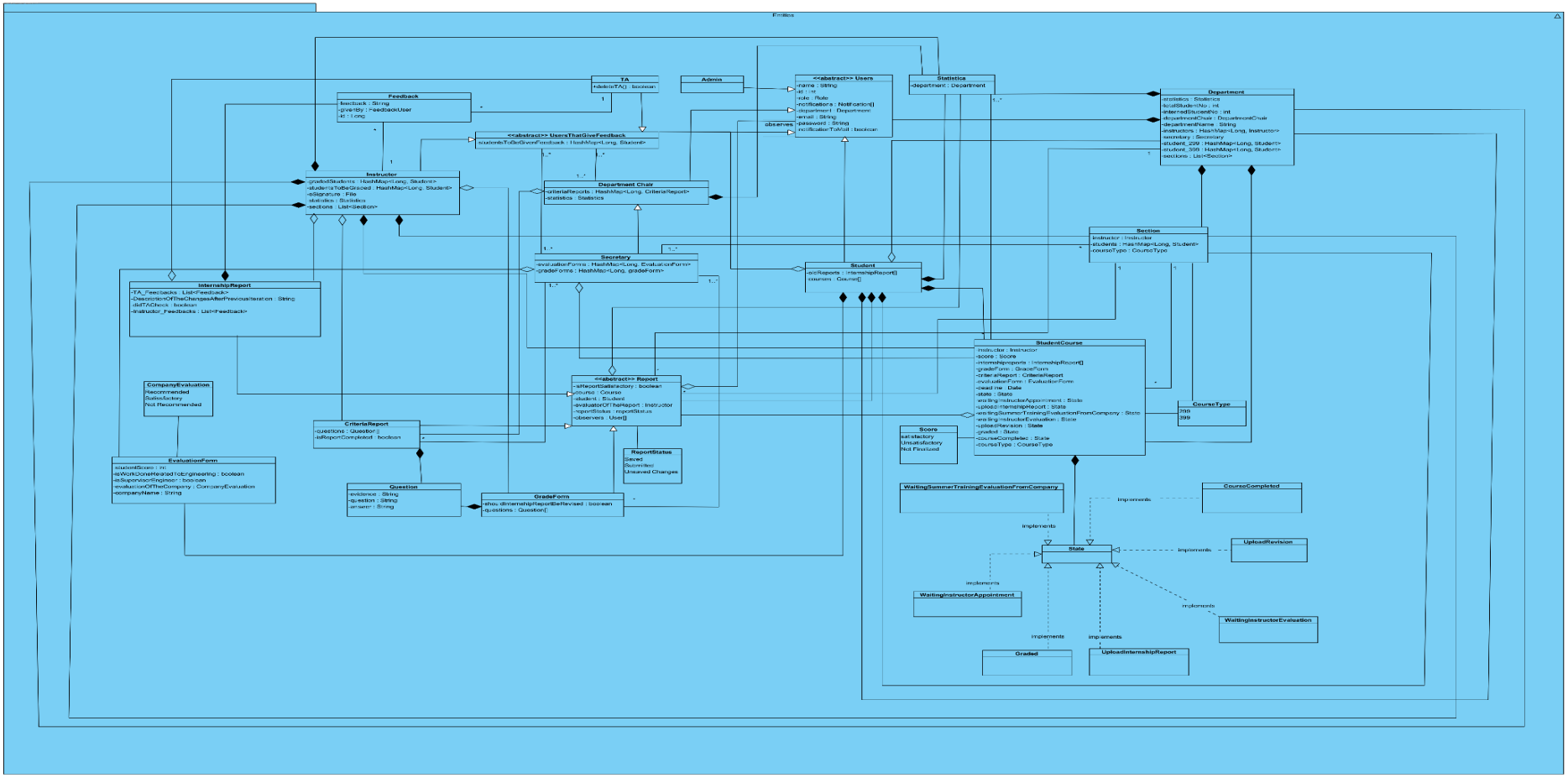

Figure 9: Data Management Layer - Entity Diagram

Diagram link:https://imgur.com/uzjBDcH
Visual Paradigm Diagram Link: Rigel Entity Diagram.vpp

3.3.3.2 Data Management Layer: Repository Diagram

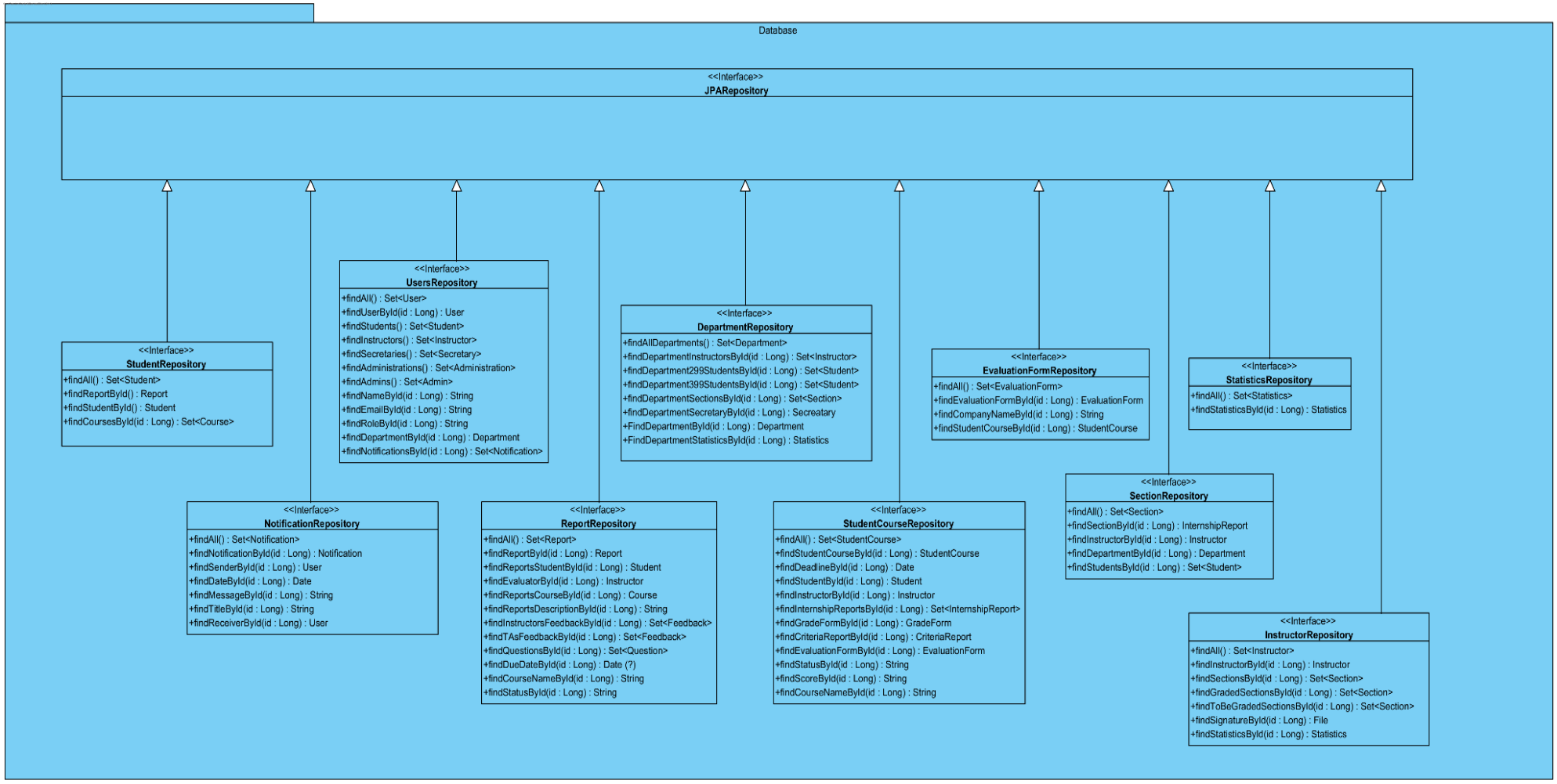

Figure 10: Data Management Layer - Repository Diagram

Diagram link: https://imgur.com/a/GuxjAMD

3.4 Packages

- **org.springframework.boot:** This package is a part of the Spring Framework and provides utilities for creating standalone, production-grade Spring-based applications.

- **com.google.api.services.calendar:** This package provides functionality for managing Google Calendar events, like creating, modifying, or deleting events.

- **com.google.api.services.drive**: This package allows developers to manage files stored in Google Drive programmatically, such as uploading, downloading, updating, and deleting files.

- **java.util:** This package contains various utility classes that provide functionality like data structures, date and time handling, input-output operations, and more.

- **React DOM:** It is a package that provides DOM-specific (document object model) methods for rendering React components on the webpage.

- **React Router:** This package allows client-side routing in a React application. It provides several components to help with the navigation and rendering of components based on the URL.

3.5 Design Patterns

There are three design patterns used in the implementation of Rigel software: State, observer, and façade patterns.

3.5.1 State Design Pattern

State pattern allows an object to alter its behavior when its internal stage changes. In Rigel software, the StudentCourse object (which represents each course each student takes [one or both of 299 and 399 courses of the student's department]) may be in various stages. It can be in a total of eight states, which are "waiting Instructor Appointment," "upload Internship Report," "waiting Summer Training Evaluation From Company," "waiting Instructor Evaluation," "upload Revision," "graded," and "course Completed." For example, in "waiting Instructor Appointment," the instructor of the studentCourse object is not yet determined, so Student users cannot upload ant report or TA users cannot try to give feedback. In "waiting Summer Training Evaluation From Company," is the state when the student-instructor matching is done, but student evaluation from the student's internship company has not yet arrived. In this case, the student may upload their reports; however, instructors cannot grade the studentCourse or give any feedback. Moreover, in the "graded" case, the instructor has given the grade for that studentCourse; therefore, the student cannot try to upload or delete any reports, but they can see their previous and graded reports. "course Completed" is mainly for corner cases. It is to handle cases where the semester is finished, but there are

still ungraded studentCourse objects. If this happens, courseObjects who have gone to the “course Completed” state with “not Finalized” score status will be automatically given an “unsatisfactory” grade. Due to this high number of internal states in studentCourse, we have chosen to use State Design Pattern.

Diagram Link: https://imgur.com/IcK6kUs

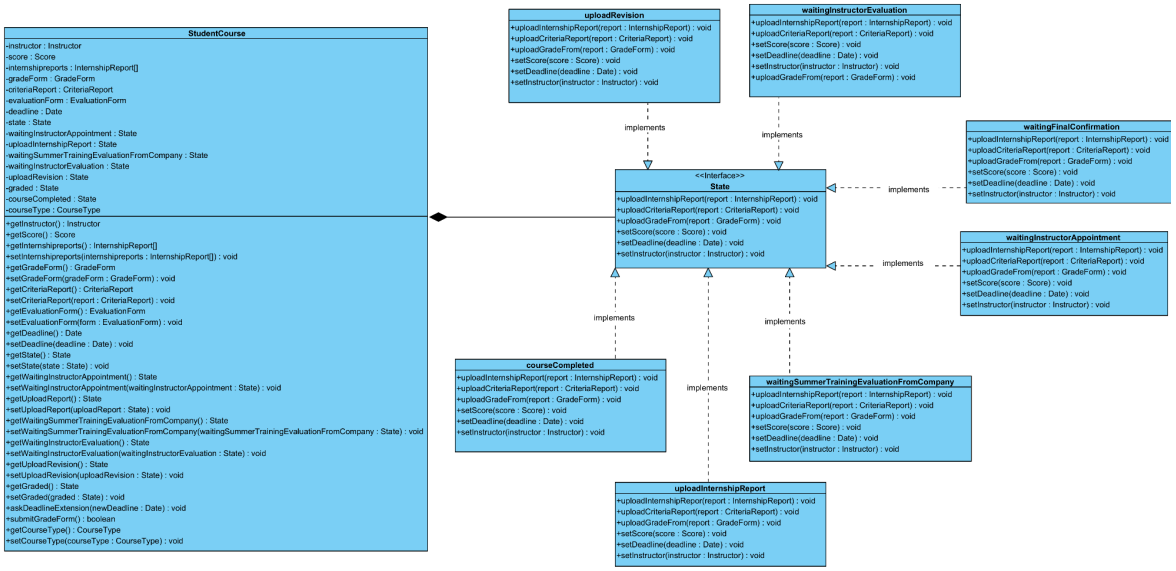

Figure 11: State Design Pattern Class Diagram - from the project’s class diagram Diagram

3.5.2 Observer Design Pattern

The Observer Pattern defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically. In Rigel Software, all of our users should get notified when any change is made to the Report objects that are of interest to them. For example, a student should get notified if their instructor gave feedback about their internship report or a department secretary should be informed if a Grade Form object has been submitted. In order to create this notification connection, we’ve decided to use the Observer Design Pattern between users and reports. This way, any users can be notified of any report that concerns them. This design pattern also allows us to cover corner cases where users should be notified of a report that generally doesn’t interest them. For example, if a department chair needs to be notified of a certain Instructor’s GradeForm object, they can be easily added as an Observer.

Diagram Link: https://imgur.com/RQu9Gca

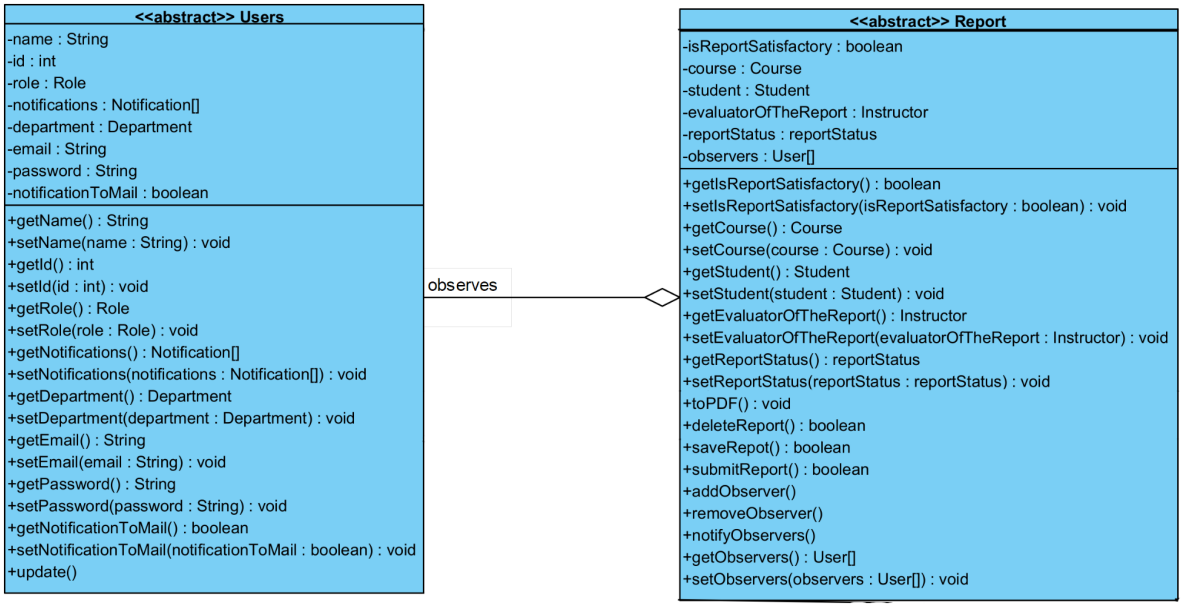

Figure 12: Observer Design Pattern Class Diagram - from the project's class diagram

3.5.3 Façade Design Pattern

The façade pattern provides a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use. In Rigel Software, we use MongoDB, and various controller objects need to interact with the database. For example, Authentication Management must interact with the database in order to approve or deny login attempts. Due to the high number of subsystems that needs to interact with the database, we've decided to utilize the Façade Design Pattern. In the program, we've constructed a "Database Management" subsystem to handle all database interaction requests and communicate with the database. This subsystem also communicates with the "API Management" subsystem to handle Database Requests from Google Drive API.

Please refer to the Figure 1 of this report for the diagram of the Façade Design Pattern.

4. Glossary

Grade Form: Grade Form is the main form where each student's XX299 or XX399 grade is determined. This report is filled once when the student's grade is determined (as "satisfactory" or "unsatisfactory")
Internship Report: The internship report is the report students prepare about their internship experience. Students get feedback from TAs and their instructor for this report if it doesn't match the "satisfactory" grade requirements.
Criteria Report: Criteria Report is filled as a part of ABET Accreditation. Instructors fill this report to show evidence of the "satisfactory" status of the student's internship report.
Evaluation Form: The evaluation Form is based on the company where the student had completed their internship. The company's executive evaluates the student, which influences

the student's grade. The company is also evaluated on certain criteria, for example, whether the student's supervisor was an engineer.

5. Improvement Summary

According to the feedback from the first iteration, we have improved our design report in many ways:

1. The subsystem diagram is more organized now as we have made progress with code implementation. Previously, we grouped the common use cases to create subsystems such as form generation, feedback, and plagiarism management. In this iteration, we have introduced a layer of abstraction with a grading management subsystem that handles all these tasks.
2. In addition, the explanations of layers and subsystems are added to the design report.
3. We have added figure numbers.
4. The image quality of the figures is improved, and high-resolution links are embedded in their respective figure titles.
5. The class diagram has been updated and according to the new class relations, design patterns have changed. Previously, we had strategy, observer, and decorator design patterns, however, in the code implementation, state and façade design patterns were more appropriate. For instance, we introduce a database abstraction layer which is the implementation of a façade design pattern. All of the design patterns are explained further.
6. In the Hardware / Software Mapping, we included communication types. All of our components communicate through HTTP requests.
7. Design trade-offs are explained further with relevant examples in the code.
8. Boundary conditions are updated and their state diagram's conditions are elaborated. Initialization is updated to include how our system gets up and running. Termination explanation is edited to exclude logging out as a termination case.

6. References

[1] "Bilkent University." [Online]. Available: https://w3.bilkent.edu.tr/web/kalite_guvencesi/faaliyet_raporu_2016-2017.pdf. [Accessed: 29-March-2023].
[2] Google, "Shared Drive Limits in Google Drive," *Google Workspace Learning Center*. [Online].Available: https://support.google.com/a/users/answer/7338880?hl=en#:~:text=File%20and%20folder%20limits%20in%20shared%20drives,-Item%20cap&text=A%20shared%20drive%20can%20contain,well%20below%20the%20strict%20limit.[Accessed: 05-May-2023].

[3] Google, “Storage and upload limits for Google Workspace,” *Google Workspace Admin Help*. [Online]. Available: https://support.google.com/a/answer/172541?hl=en.[Accessed: 05-May-2023].