Bilkent University
Department of Computer Engineering

CS319 Object-Oriented Software Engineering

Internship Management System

Project Design Report

Ahmet Alperen Yılmazyıldız 22002712
Ahmet Berke Gökmen 22002105
Erdem Eren Çağlar 22002359
Mahmut Mert Gençtürk 22003506
Cem Apaydın 21802270

Instructor: Eray Tüzün
Teaching Assistant(s): Yahya Elnouby, Muhammad Umair Ahmed and Tolga Özgün

Second Iteration

May 21, 2023

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the CS319

Contents

1 Introduction 3
- 1.1 Purpose of the System 3
- 1.2 Design Goals 3
  - 1.2.1 Information Security & Privacy 4
  - 1.2.2 Maintainability 4

2 High Level Software Architecture 5
- 2.1 Overview 5
- 2.2 Subsystem Decomposition 5
- 2.3 Deployment Diagram 6
- 2.4 Hardware/Software Mapping 7
- 2.5 Persistent Data Management 8
- 2.6 Access Control and Security 9
- 2.7 Boundary Conditions 12
  - 2.7.1 Initialization 12
  - 2.7.2 Termination 13
  - 2.7.3 Failure 13

3 Low Level Design 14
- 3.1 Object Design Trade-offs 14
- 3.2 Final Object Design 16
- 3.3 Layers 17
  - 3.3.1 User Interface Management Layer 17
  - 3.3.2 Web Server Layer 18
  - 3.3.3 Data Management Layer 19
- 3.4 Design Patterns 21
  - 3.4.1 Builder Pattern 21
  - 3.4.2 Facade Pattern 21

4 Glossary 22

5 Improvement Summary 23

6 References 25

Project Design Report

Internship Management System

1 Introduction

This report explains the purpose of the software and its design goals, such as information security & privacy, functionality, maintainability, and usability. It gives further information about the software's design by indicating its high-level software architecture. The subsystem decomposition, data management, access control & security throughout the system, and boundary conditions are stated in detail. Additionally, the report provides the low-level design of the software by specifying its design patterns, class & object designs, and object design trade-offs.

1.1 Purpose of the System

The internship management application is a browser-based system that will automate the processes currently done by hand and benefit the students and the academic staff participating in the internship report evaluation process. The internship report evaluation process is managed inefficiently in the current system - through three different platforms: Moodle, Google Drive, and email. The current system requires lots of by-hand operations. Evaluators and teaching assistants have been checking their assigned submissions & the submissions' company reports through Google Drive by visiting many folders and obtaining the report to be evaluated from Moodle. The secretary has been conducting each process for every internship course. Hence, this web-based project will take on the secretary's workload by providing the following services.

In the new system, evaluators & teaching assistants can see their assigned submissions, download & evaluate the students' internship and company evaluation reports, request changes, and see previous reports and feedback (an annotated version of the report and comments). Evaluators will also be able to see the teaching assistant's feedback. The system will inform the students about their submission updates in each process. The secretary will be able to initialize the courses, assign evaluators & teaching assistants to student submissions, upload students' company evaluation reports, and export the submissions when the semester ends. Therefore, the system offers a mechanism that combines each minor process into one application, reducing the time spent on these internship report evaluation operations.

1.2 Design Goals

The requirements specified below will mainly aim for the application's viability and reliability. Applying these non-functional requirements to our application makes the system more user-friendly and robust in many aspects.

1.2.1 Information Security & Privacy

Bilkent University accepts 160 engineering students per year per department, hence the number of students that will use the system is expected to be between 350-400 per department [2]. As there are four engineering departments, the system will host approximately 1600 students including the evaluators, teaching assistants and admins each semester. Hence, the application needs to store confidential information of approximately 1600 users, including their passwords and emails. The application also includes many more data that are not vitally sensitive, yet should be private such as their reports, feedback, and their grades. Therefore, it needs to be ensured that no student could access another's reports or status. The system will ensure that users can only operate within the scope that they are authorized. Since the users would not want the data they have provided to the application to be shared with others or accessed by third parties, the application needs to secure these data thoroughly throughout the application. By this means, we have decided to take the following measures to strengthen the security of the program:

- We will use MongoDB and AWS since these choices reduce possible security vulnerabilities, such as AWS encrypting the database information using SSL/TSL [3].
- The passwords of users will be hashed using BCrypt - which is a password encoding mechanism, and the hashed version will be saved into the database [4].
- There will be no register option. The admin will be able to create the student profiles upon the semester beginning - ensuring that no other user could register for the application. The admin will upload an excel sheet including the user information (Bilkent IDs, names, emails, roles, & departments). Bulk insert method is used to load the provided data in bulk [5]. Hence, the users will be created accordingly, all at once. The default passwords will be emailed to the provided user emails.
- The students will be able to have only one account - created by the admin and associated with the student's Bilkent ID and email.

1.2.2 Maintainability

The internship report evaluation's criteria or the process itself may change over time. One more step might be added or eliminated from the process, the questions to evaluate the reports might change, or a new department may be added to the system so that the system needs to be adjusted in such a way that satisfies this new department's needs for the internship report evaluation process. In order to make the system maintainable, design patterns and state-of-the-art techniques will be integrated so that the system will be able to extend easily according to the new criteria and needs.

2 High Level Software Architecture

2.1 Overview

In this section, subsystem decomposition is explained by providing a general picture of how our systems work. In addition to that, the deployment diagram is shown, explaining the interactions between client, server and its dependencies. Also, the hardware / software mapping section explains which additional services are required to run the application as well as providing information about minimum requirements. As for data, the Persistent Data Management section explains how the data flows in our application and how it's handled, processed and persisted. After that, the Access Control and Security section specifies the roles in the application and how the actions are distributed among the actors. Lastly, the Boundary Conditions section explains what happens on initialization, termination and failure of the application.

2.2 Subsystem Decomposition

High resolution version of the diagram can be seen here.

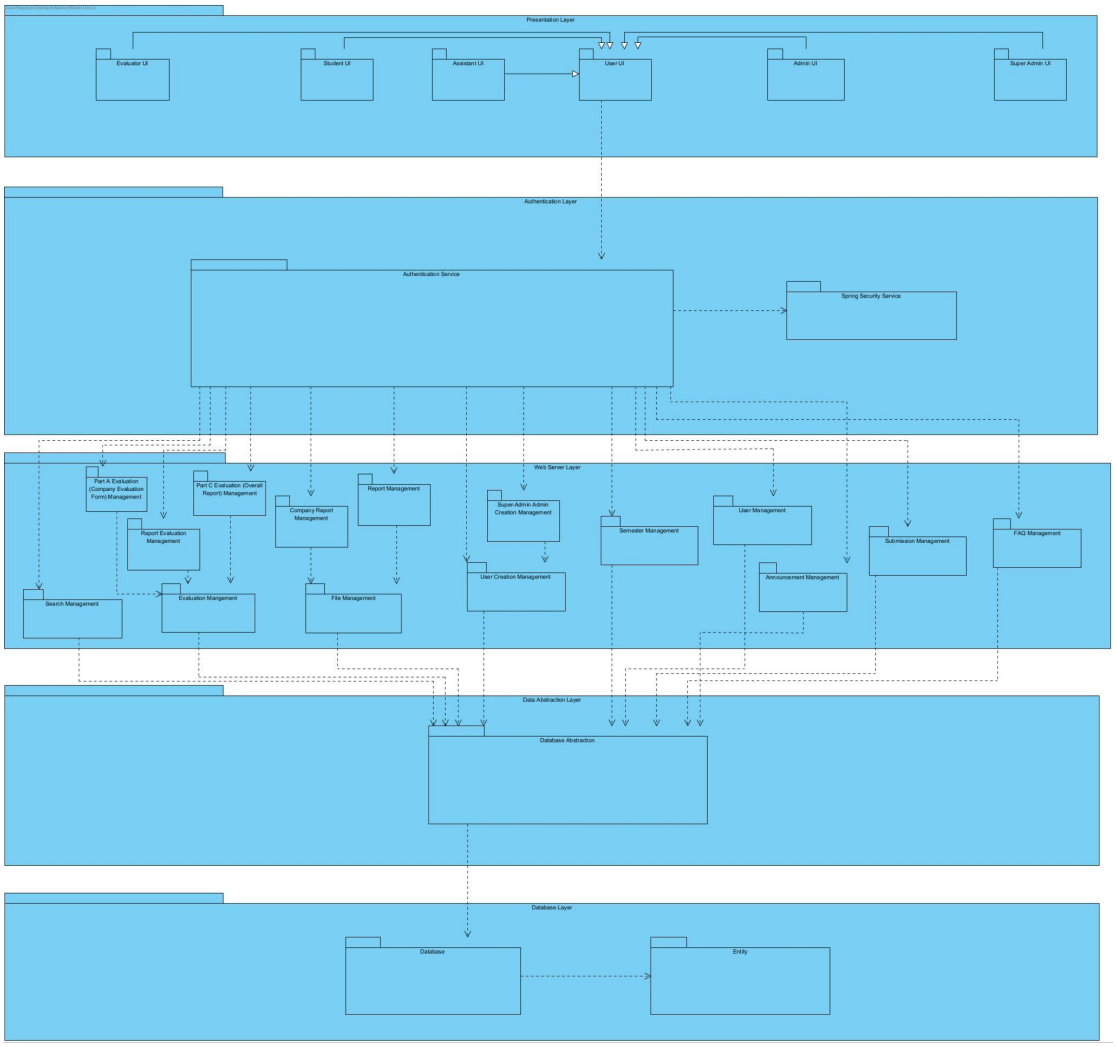

Fig. 1: Subsystem decomposition diagram

Our architecture has five layers. In order to provide scalability and maintainability to our system we used this layer separation. We gave severe attention to how our subsystems are as loosely coupled as possible among themselves.

The Presentation Layer contains all the subsystems that interact with the user. Therefore, they actually act as the boundary objects of the system. Every subsystem has its specific controller object in the Web Server Layer.

The Authentication Layer contains the subsystems that provide security design goals to the system. All data coming from the user needs to be classified in this layer before it interacts with the system's control and entity objects.

The Web Server Layer contains the control objects of our system. All of the back-end operations of our system are at this layer. Management subsystems in our design contain both controller and service objects from the back-end for corresponding service.

The Data Abstraction Layer and The Database Layer are assigned with storage data of our system and contain entity objects from the back-end.

2.3 Deployment Diagram

This diagram shows the interaction between client, server, and their related dependencies. Users access the frontend server via their preferred web browser. The frontend server returns a simple HTML page allowing users to interact with the backend server using fancy UI over HTTP protocol. The backend server handles the requests from the frontend server. If a file is requested to be downloaded or uploaded, the backend server interacts with the AWS S3 bucket to perform the corresponding action over TCP/IP protocol. On requests related to an entity update, such as a user changing their password, the backend server persists the request to the database, communicating over mongodb protocol. Lastly, the backend server accesses a Redis instance over redis protocol for requests such as login or accessing the cached data.

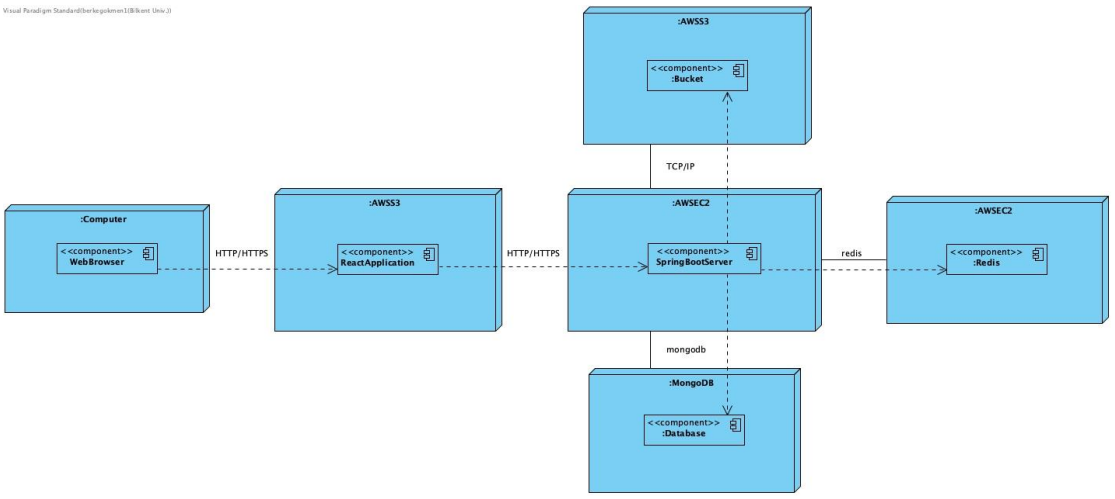

Fig. 2: Deployment diagram

2.4 Hardware/Software Mapping

The app is divided into 2 parts, frontend and backend. On the backend, the app uses Spring Framework 6.0.2 to manage and maintain backend logic in an efficient way. On the frontend, it uses React Framework 18.2.0 as this framework is easy to use, maintain and manage. Also, the fact that it allows the developers to reuse a big portion of the code makes it a good choice. Moreover, nowadays all modern browsers support react applications; thus, no individual should have trouble running the application provided they have the latest version installed of a modern browser.

As for the data, we use 2 different components. One is MongoDB and the other is Redis (AWS Elasticache). MongoDB is chosen over traditional SQL databases since it is easier to run on cloud (via http://cloud.mongodb.com/) and easier to use due to its document-like data structure. Despite the general thought that databases like MongoDB, which are so called "NoSQL" databases, do not support relations between entities, it's actually not the case. NoSQL stands for "Not Only SQL" and MongoDB in fact supports relations as much as a traditional database does [6]. Also, MongoDB and other NoSQL databases support both structured and unstructured data unlike traditional SQL databases, allowing developers to work in a fast paced agile development environment. Considering the expected number of users, which is 1600 as explained in the Information Security & Privacy section our MongoDB instance is currently running on a M0 cluster [7], which has 500MB of storage. M0 cluster instance should be enough to meet the demands of 1600 users, since most of the large data is stored elsewhere, such as in S3 buckets. Moreover, the instance can be easily accessed, viewed and edited via MongoDB Compass [8], allowing for easier development, data manipulation and debugging. Redis, on the other hand, is not used for persistent data, rather it's used to store volatile data such as authentication tokens, one time password codes, etc. The reason that we used Redis and did not simply put everything in a MongoDB instance is that Redis gives significantly better performance when accessing simple data [9]. Since validating authentication tokens is done at every request, it's better to use an in memory data structure which has better performance over a normal database. Furthermore, Redis is used to cache some simple and frequently accessed information such as announcements in the application. Our Redis instance is currently managed by AWS Elasticache [10] and bearing in mind that the system should support 1600 users, it runs on a T2 Micro instance with 1GB of memory and 3.3 GHz Intel Xeon Scalable processor [11]. Since Redis is known to be fast, 1GB of memory should be enough to meet the demands of 1600 users.

In terms of hardware, this project does not require any specific hardware components in order to be used since the app runs on the web. Provided that the user has a device capable of running a web browser and accessing the web, there should be no problems accessing the app. That is, the app is usable on both computers, mobile phones, tablets and any device that can run a modern web browser[1].

[1] Some examples of recommended modern browsers are Google Chrome (versions 41.0.2272 and higher), Mozilla Firefox (versions 21 and higher), Microsoft Edge (any version), Safari (versions 6 and higher), and Opera (versions 11 and higher). As used

In order to access the app via mobile phone or tablet, the user should not need any additional hardware equipment. As for desktop computers, users may need a keyboard and mouse depending on whether their computer is a desktop or laptop.

2.5 Persistent Data Management

MongoDB will be used as the main database to store all the information regarding the Internship Management System. Unlike traditional SQL databases, MongoDB supports nested documents, which means that developers do not have to create a separate table just to create a one to one relationship. Thus, developers can directly map the entity objects into database documents while avoiding unnecessary relations [12]. This feature opens the way to faster development, and easier debugging. Additionally, Amazon S3 (Simple Storage Service) will be used as object storage, specifically to store documents such as student internship reports. It is chosen to handle all the file operations since it is costly to write and read file information into a normal database, whether it is SQL or NoSQL. File data is known to take too much space, which might degrade the database performance considering our 500MB MongoDB instance.

Users (SuperAdmin, Admin, Evaluator, Assistant and Student), Submissions, Courses and Semesters will be stored in the MongoDB cloud. Respective documents, such as report PDFs will be stored in AWS S3 service.

Since the application works dynamically and handles dynamic data, there should be no data loss between actions and user state must be persisted. The backend service works as a REST API [13], that is it does not hold user state, rather it distributes JWT tokens [14] which identify the user. The state of the user is persisted in the frontend application using Zustand [15], a library to manage global state in React applications.

technologies contain numerous modern libraries with numerous API calls, legacy browsers like IE8 and Safari 3 may present compatibility problems.

2.6 Access Control and Security

Since the internship management system holds sensitive data about both users and possibly companies, it needs to be as secure as possible and bullet proof. The access control to certain resources on the backend side is handled by a sophisticated request interceptor and token filter. The token filter ensures that users with invalid (expired or blacklisted[2]) tokens cannot access the protected routes, that is they cannot use the application unless they login again and obtain a fresh new token. The request interceptor works as a role checker system. It ensures that some routes in the application are only accessible to users with certain roles. For instance, a student cannot access the list of all students taking a semester course since the route is restricted to Admins (Secretaries).

On the client side, we also use protected routes to further improve security. According to the user information provided by backend and held in the global state of the application, user's access to certain pages are restricted. For instance, a student cannot see the page of report evaluation even though the student is not able to send any requests from that page to the backend. This ensures that in addition to security measures taken on the backend side, the users are not shown any irrelevant pages, thereby further strengthening the system.

In terms of sensitive data protection, such as reports, passwords, etc. we use different techniques to secure each of them. The reports are held in AWS S3 service and the instance that we're using is configured such that no object in the S3 bucket is neither visible nor accessible from outside of the configured services. Only our backend service, deployed on AWS Elastic Beanstalk can access S3 the instance via the access and secret keys generated by AWS. Our database, on the other hand, is configured to reject any traffic not coming from our backend. Also, it's further protected by a password which is only used in Elastic Beanstalk instance. As for the passwords, they are hashed using a password encoder called Bcrypt and then saved into the database [5].

[2] User tokens get blacklisted when the user logs out. Blacklisting operation is handled by Redis and the blacklist records are persisted as long as the tokens are not expired on their own. On each request, the blacklist record is checked to see if the token is valid, thus ensuring that in a case of stolen token, as long as the original user logs out the attacker will not be able to use the stolen token.

Access Control Matrix

Table 1. Access Control Matrix

	Super Admin (Department Chair)	Admin (Secretary/Department Chair)	Evaluator (Faculty Member)	Assistant (TA)	Student
Login	**X**	**X**	**X**	**X**	**X**
Logout	**X**	**X**	**X**	**X**	**X**
Change Password	**X**	**X**	**X**	**X**	**X**
Request Password Change Email	**X**	**X**	**X**	**X**	**X**
Get Profile	**X**	**X**	**X**	**X**	**X**
Create Admin Account	**X**				
Initialize Semester (Register Users in Bulk)		**X**			
Add User to Semester		**X**			
Add Course Definition to the System		**X**			
List Available Course Definitions		**X**			
Get List of Active Semesters		**X**			
Get Registered Semesters			**X**	**X**	**X**
Deactivate Semester		**X**			
Get List of Users Registered To Semester		**X**			
Export Students' Submissions' Grades From Semester		**X**			
Export Statistics of Semester		**X**			
Get User Profile by Bilkent Id		**X**	**X**	**X**	
Search Users by Name		**X**			
Upload Company		**X**			

	Super Admin (Department Chair)	Admin (Secretary/Department Chair)	Evaluator (Faculty Member)	Assistant (TA)	Student
Evaluation					
Get Company Evaluation Report for Student Submission		**X**	**X**		
(Re)Upload Submission Report With Comments					**X**
View Submission Status			**X**	**X**	**X**
Approve Submission Report			**X**	**X**	
Request Changes to Submission Report			**X**	**X**	
Grade Submission			**X**		
Assign Assistant to Submission		**X**			
Assign Evaluator to Submission		**X**			
Download Reports Related to a Submission		**X**	**X**	**X**	**X**
View a Student's Submission(s)		**X**			
View My Submission(s)					**X**
View Assigned Submissions to Evaluate			**X**	**X**	
Download Finished Submission Evaluation Report		**X**	**X**		
View Announcements	**X**	**X**	**X**	**X**	**X**
View FAQ	**X**	**X**	**X**	**X**	**X**
Create/Delete Announcement		**X**			
Create/Delete FAQ		**X**			

2.7 Boundary Conditions

Since the system runs on a cloud environment, we do not expect to have any problems regarding hardware failures. However, in case of any failure, the system will try to get itself up and running again and again until either it succeeds or it has tried too many times. You can see the boundary conditions of the system below.

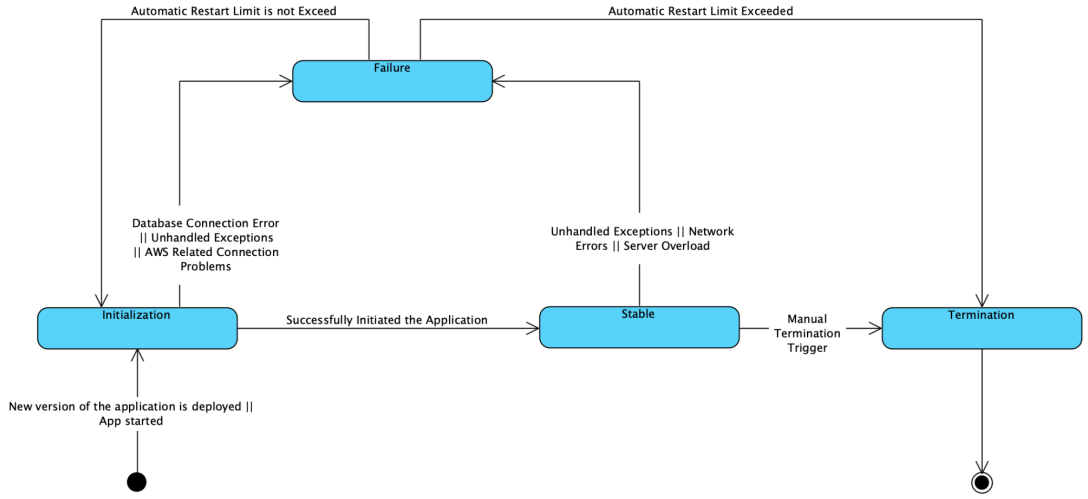

Fig. 3: Boundary conditions diagram

2.7.1 Initialization

The app can be started by either triggering the GitHub Action which automatically deploys the app to the AWS Elastic Beanstalk instance and starts it or it can directly be run from the Elastic Beanstalk console. The application requires Redis, MongoBD and AWS S3 instances to work properly; that is, the maintainer of the application needs to make sure that an AWS Elasticache instance is running, MongoDB instance is deployed and S3 bucket is ready to use. Also, they all need to be configured such that their network security properties allow them to be accessed by the Elastic Beanstalk instance where the application is deployed. Moreover, the maintainer needs to make sure that the frontend part of the application is deployed on AWS Amplify and is authorized to access the backend. The initialization steps list can be seen below:

- Redis Elasticache instance is created/started.
- MongoDB instance is created/started.
- S3 instance is created/started.
- App initialization is triggered through Elastic Beanstalk or a new version of the app is deployed by GitHub actions.
- The frontend react application is deployed to AWS amplify and started.
- App is now ready and accessible.

2.7.2 Termination

The termination can happen either in a planned manner or due to unhandled exceptions. A maintainer can manually trigger termination for maintenance purposes or to handle security concerns. Also, an unhandled exception within the system can lead to its termination if the problem persists. That is, in case of such an event, the application will try to restart itself until either the issue is resolved or automatic restart limit is exceeded. In addition to that, in case of any unplanned termination, both AWS services and MongoDB have built-in features to prevent data loss by securing the last available data. The termination can occur due to following reasons:

- Manual trigger, due to maintenance or security concerns.
- Upon planned termination, the logged in users will be notified that the system will shutdown in a certain amount of minutes and they need to save their unsaved work.
- Upon planned termination, the login functionality will be disabled for users that are trying to log in to the system and they'll be presented with a screen saying that they should try again later.
- An unhandled exception which causes the system to crash altogether.
- Network errors from AWS side which might render the system inaccessible.
- Connection problems between components of the system. Such as the backend service not being able to access the database for security reasons or misconfiguration.

2.7.3 Failure

Failure of backend service or the dependencies of it (Elasticache, MongoDB, S3) will be automatically handled by restarting the environment or using the backup copy of the service that failed. For instance, when a MongoDB instance fails, it immediately assigns a slave database node as the master and continues where it left off with the last available data. If a failure cannot be resolved by simply restarting the environments or using backups and if the number of automatic retries are exceeded, the developers will be notified by AWS and GitHub indicating that the environment is not healthy and needs careful inspection. In any way, the application is designed to prevent data loss as much as possible, thus when the application is running in a healthy way again, it'll continue where it left off. The cases in which the app comes to a failure state can be seen below:

- Unhandled exceptions which result in ungraceful shutdown.
- Memory overload on the server, which forces it to restart itself to clear memory.
- A budget overrun on AWS, which eventually kills the server if no further action is taken to avoid unnecessary costs.
- MongoDB Cloud related failures, which prevent the app from accessing the database.
- Network failures, which restrict the access to any other service that is running outside the backend EC2 machine.
- AWS hardware related problems that would lead to the termination.

3 Low Level Design

3.1 Object Design Trade-offs

Usability v. Functionality

The Internship Management System is designed with simplicity in mind, considering that many users may not be familiar with it. The goal is to ensure that new users can easily understand the system's processes and how to use it. To achieve this, the system focuses on implementing only the most essential functionalities, such as uploading reports, providing feedback, and viewing submissions. This decision prioritizes usability and clarity for users.

As a result of emphasizing simplicity, the system may have fewer functionalities than initially expected. For example, although internship supervisors can complete company evaluation forms through a one-time link, this feature was not implemented to avoid creating confusion and complexity for secretaries who would need to manage multiple sources of the same information. Instead, the system concentrates on implementing a select few features that are necessary, while avoiding unnecessary functionalities that may introduce ambiguity without contributing significantly to the overall usability.

Rapid Development v. Automation

The Internship Management System's primary goal is to automate the application evaluation process. Therefore, we aim to automate the evaluation and communication process completely. The system will provide an automated email system and a feedback system to minimize the emailing process between students and faculty members. Also, it will provide an automated grading system to decrease the workload of department secretaries to mark and publish every student's final state of the report. The system must have many complicated methods and subsystems behind the scenes to make automation processes possible. Therefore, aiming for automation will decrease the development speed of the system.

Scalability v. Rapid Development

In the Internship Management System, we designed the system to support all departments in the engineering faculty, which means having a high number of active users, 1600, as explained in the Information Security & Privacy section. Because of this high user number, we decided to store file data in a cloud system, AWS S3, to reduce the burden on our database system. In addition, we decided to keep the evaluation form questions in our database and provide a semi-dynamic system to support all departments' needs, and handle the changes in the questions easily. Also, we decided to use Redis as a cache for authorization and frequently accessed resources, such as announcements, to decrease the burden in the database. All these factors will reduce our development speed but increase the system's scalability.

Security v. Performance

Security is a crucial aspect of the internship management system. Given the sensitive nature of the information stored, such as internship details, user data, and company information, the system prioritizes security over performance. It incorporates multiple features to safeguard user data and prevent any potential data breaches.

To begin with, the system stores user passwords in a hashed format within the database. This ensures that even in the event of a data leak, unauthorized individuals cannot access user accounts. However, this hashing process and the subsequent comparison of hashed passwords with plain text passwords during login requests impose a performance overhead on the system.

Additionally, the system's backend service operates in a stateless manner, meaning it does not retain session information about users. Consequently, for every operation, the system needs to verify the credibility of the request. To achieve this, users are assigned JSON Web Tokens (JWTs) that accompany each request made to the server. These tokens serve as identifiers for users and expire after one hour, requiring users to log in again. Each request is checked using Redis to determine if the associated token has been blacklisted. Tokens are blacklisted in scenarios where there are security concerns regarding the user's account or during logout. This blacklisting process is necessary because JWTs do not expire automatically. While this credibility verification process introduces some performance degradation, it significantly contributes to maintaining the system's security.

3.2 Final Object Design

High resolution version of the final object design diagram can be seen here.

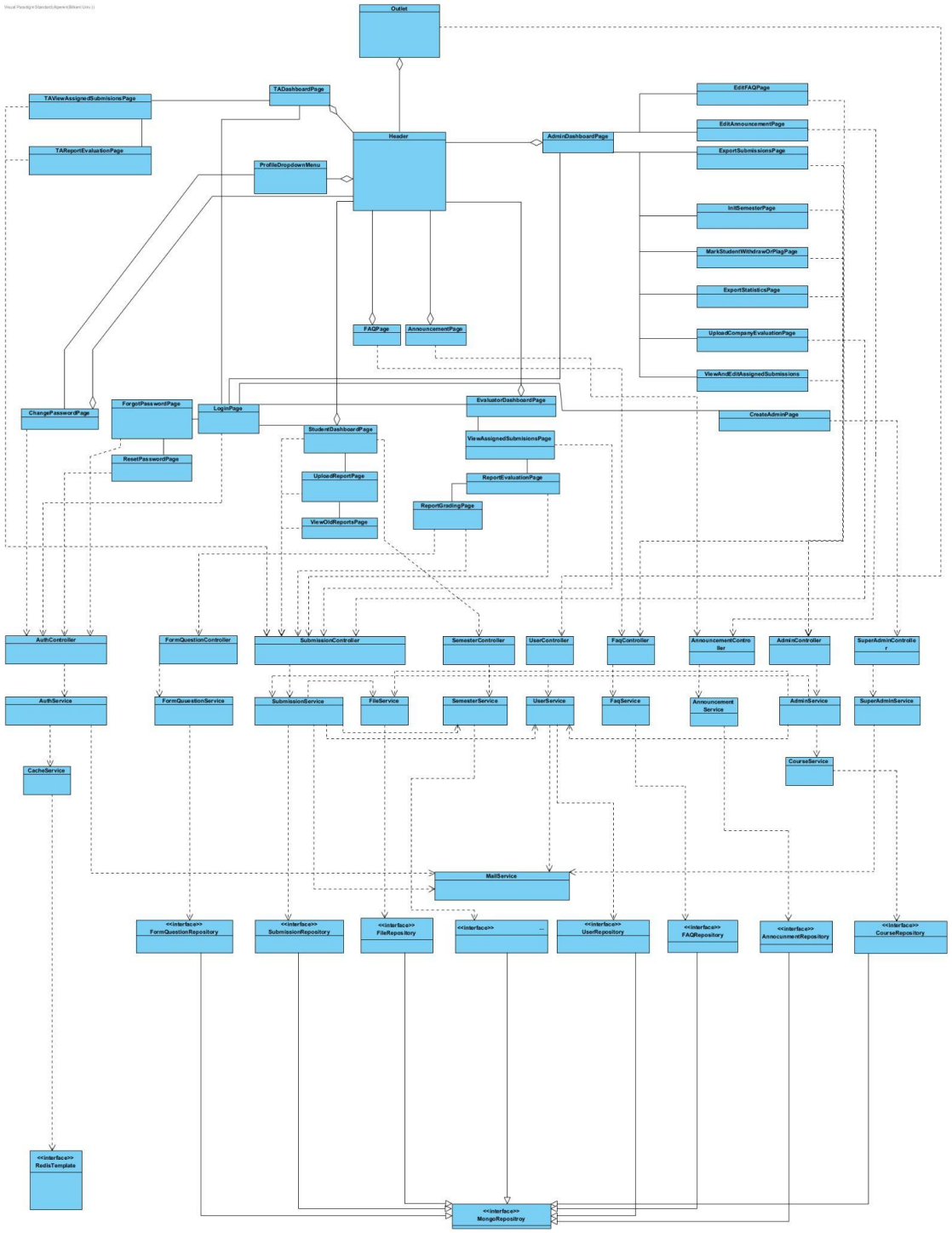

Fig. 4: Final object design diagram[3]

[3] Outlet in this diagram refers to an empty page, on which every other page is rendered.

3.3 Layers

3.3.1 User Interface Management Layer

High resolution version of the user interface management layer diagram can be seen here.

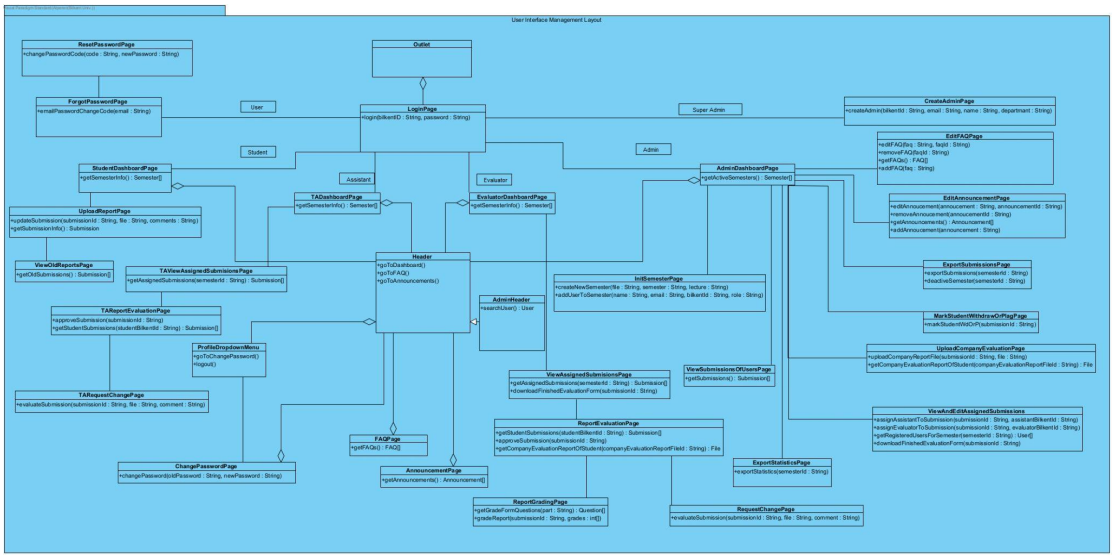

Fig. 5: User Interface Management Layer[4]

[4] Some methods are repeated in some pages. Such as getAssignedSubmissions(semesterId: String) in TAViewAssignedSubmissionsPage and ViewAssignedSubmissionsPage. This stems from the fact that we designed the backend service in a simple way so that we do not have to write separate methods for every user for the same or similar actions. Thus, some pages use the same functions to get the required data.

3.3.2 Web Server Layer

High resolution version of the web server layer diagram can be seen here.

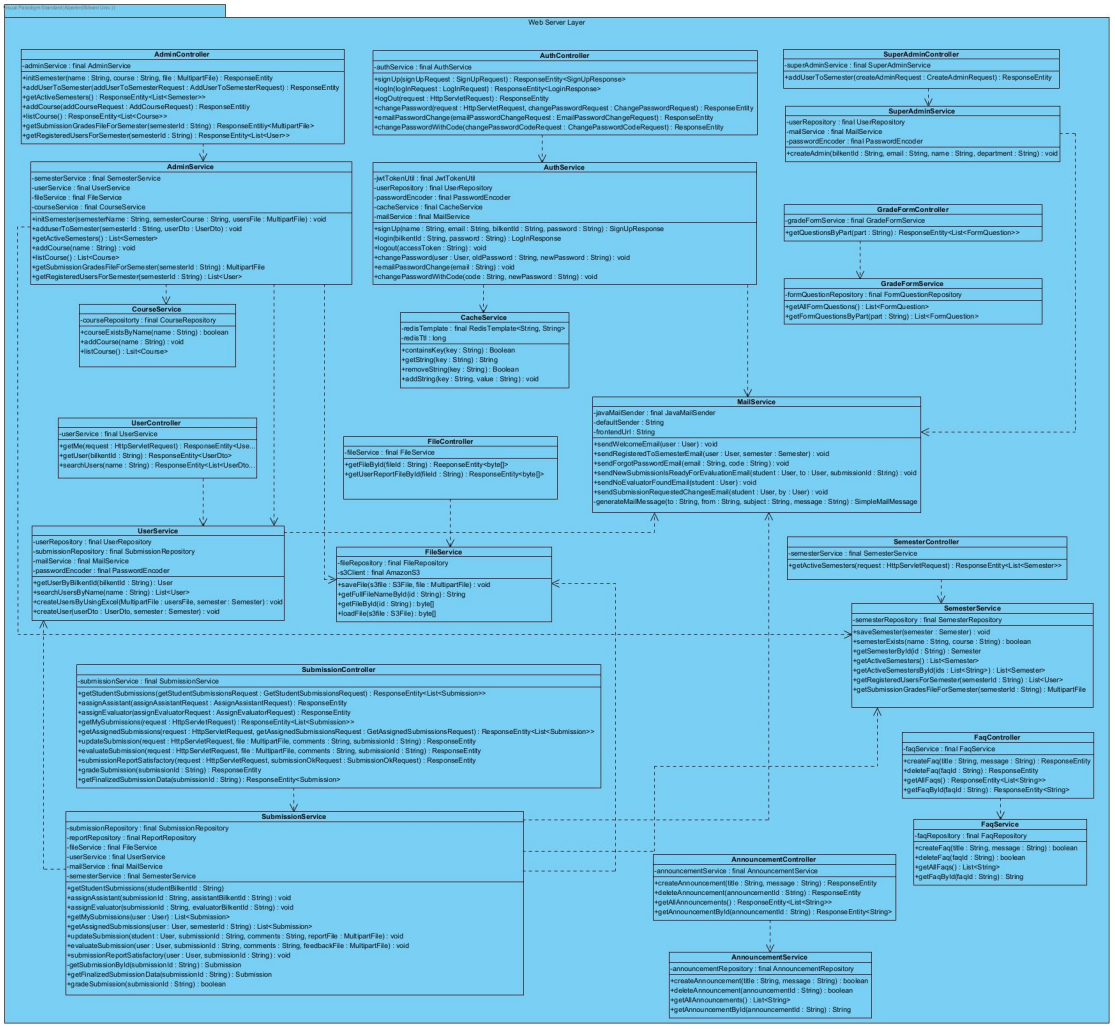

Fig. 6: Web server layer diagram[5]

[5] Here, IDs are kept as string since MongoDB internally represents IDs as ObjectIds which consists of letters and numbers [16].In the Spring environment, these ObjectIds are used as strings since they cannot be represented as just numbers. Whenever a query is made to the database, they are automatically converted to ObjectIds by MongoDB.

3.3.3 Data Management Layer

High resolution version of the data management layer diagram can be seen here.

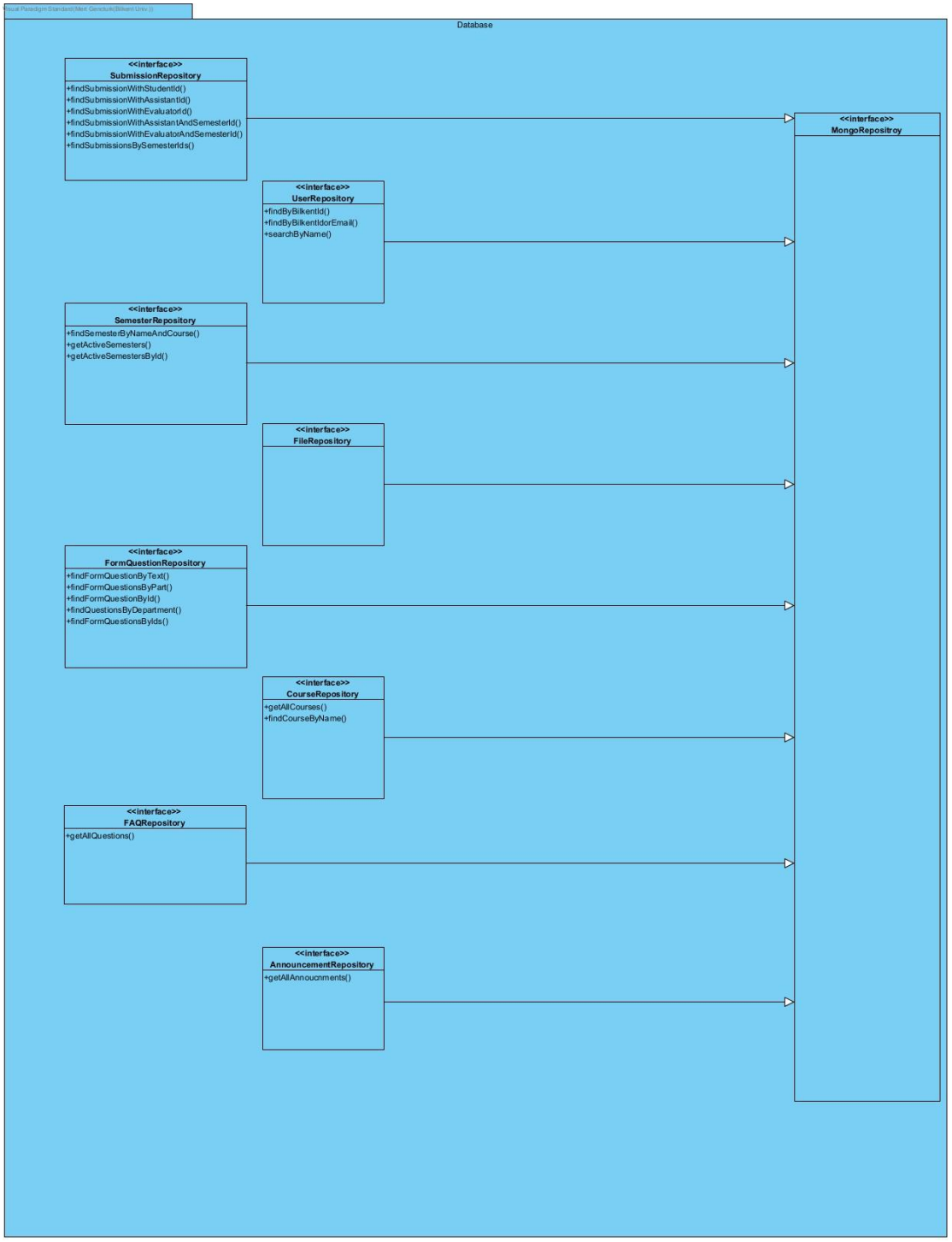

Fig. 7: Data Management Layer - Repository Diagram

Data Management Layer consists of two components, which are Repository and Entity components. Repository layer communicates with the database by sending or retrieving data using the related entities shown in the Entity diagram below. Note that the MongoRepository interface already comes with some built-in functions such as findById, save or delete.

High resolution version of the entity diagram can be seen here.

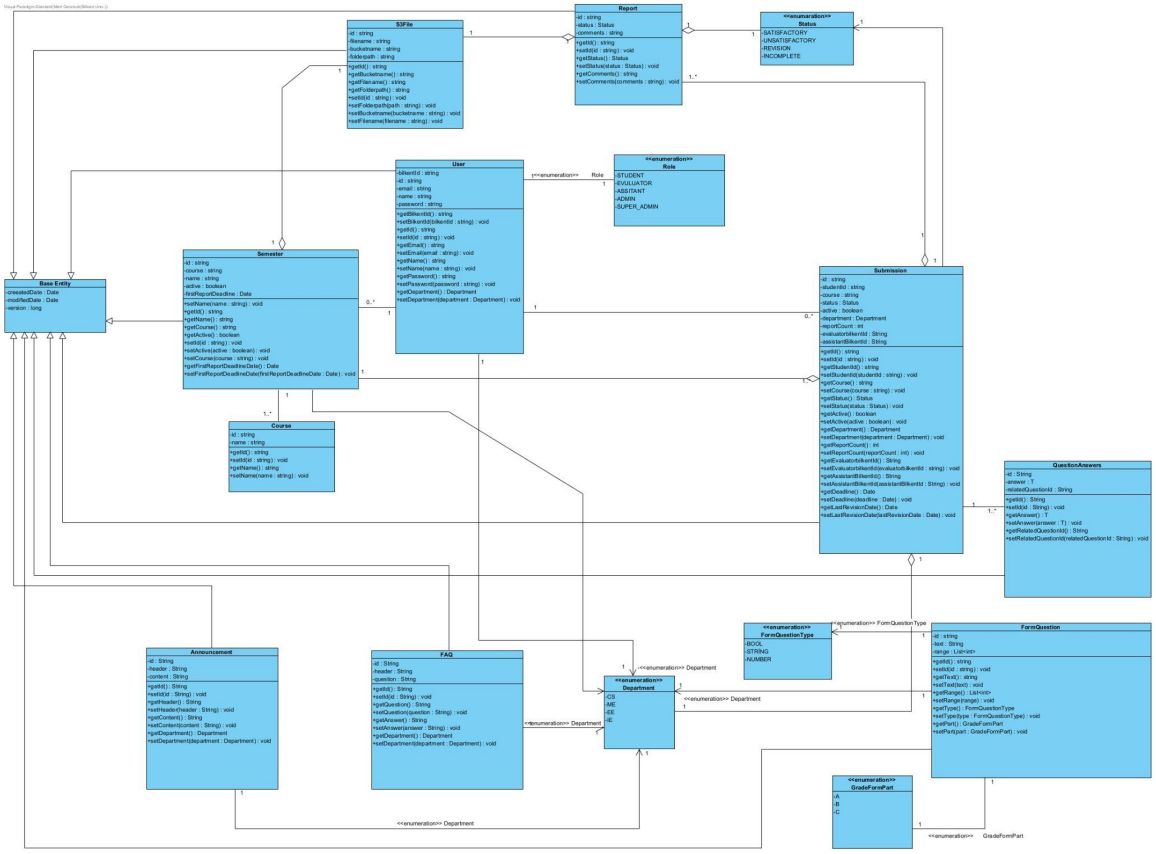

Fig. 8: Data Management Layer - Entity Diagram

3.4 Design Patterns

3.4.1 Builder Pattern

In the application's backend, we encounter various types of objects that often have optional parameters. Creating overloaded constructors for each object becomes cumbersome and impractical. To address this issue, we adopted the builder pattern for constructing all our objects. This pattern enables us to build objects step by step and leave certain fields empty when necessary.

To implement the builder pattern efficiently, we sought a ready-made solution rather than creating all the builder classes and methods from scratch. We discovered the Lombok package, which offers a powerful and elegant solution [17]. Lombok provides automatic builder construction for classes, along with autogenerated getters and setters. Instead of manually implementing each required method, Lombok simplifies the process by introducing annotations that can be applied to an entity class. These annotations generate the necessary builder methods, as well as getters and setters, automatically. By leveraging Lombok, we save time and effort in implementing the builder pattern in our application.

3.4.2 Facade Pattern

Given the complexity of the internship management application, we made the decision to adopt a layered architecture in order to separate and organize the different operations' logic. This separation occurs in two key areas: between the controller and service layers, and between the service and database layers.

In the controller and service layers, we have two distinct classes. The controller class handles incoming requests, parses them, and calls the appropriate service layer with the necessary information. It then takes the resulting data and returns it as a response. By implementing this separation, any changes made to the service layer will require minimal modifications to the controller layer. This allows for flexibility and ease of modification without disrupting the overall functionality of the application.

The same principle applies to the separation between the service and database layers. We utilize repository classes for accessing the database, which encapsulate the specific logic for querying the database. Notably, the service layers do not contain any database-related logic. This design approach allows for the modification of database queries without impacting the implementation of the service layer. In other words, the service layer remains unaffected by changes to the underlying database structure or query logic.

By adopting this layered architecture, we ensure a clear separation of concerns, enhance maintainability, and enable more flexible modifications to the different layers of the application.

4 Glossary

- **Semester:** Terminology that is used to explain a course in one semester. Such as CS299 in 2022-2023-fall semester. All the users that belong to a semester will have to submit a report and their reports are tracked according to the semester.
- **Evaluator:** Faculty members, professors that grade the reports.
- **Assistant:** TAs that pre-approve the reports.
- **Admin:** Department secretaries responsible for handling the reports, initializing a semester with a list of users.
- **Super Admin:** An account that the system provides. It's only responsibility is to create admin accounts.
- **Company Evaluation Report:** The report that is prepared by the company supervisors indicating the student’s overall performance in the internship.
- **Part A:** Refers to the workplace evaluation and the grades from the company supervisor.
- **Part B:** Refers to whether the report is satisfactory or not.
- **Part C:** Refers to the evaluation of the final version of the report. It includes evaluation of the work and evaluation of the report.

5 Improvement Summary

General

- Included footnotes instead of writing note blocks.
- Improved the grammar of the report.

1.1 Purpose of the System

- Provided more detail about the previous and to-be-implemented system.
- Corrected language usage.

1.2 Design Goals

- Specified the user creation method's stages.
- Specified the expected number of users & their confidential information.
- Explained BCrypt briefly & provided reference.
- Eliminated the third design goal, usability.

2.2 Subsystem Decomposition

- Removed unnecessary interfaces at the UI package.

2.3 Deployment Diagram

- Included explanations of how the components of the system interact.
- Changed FrontendServer to ReactApplication and BackendServer to SpringServer

2.4 Hardware/Software Mapping

- Explained how the component specification decisions are made using the expected number of students.

2.5 Persistent Data Management

- Explained why we have chosen MongoDB over traditional SQL databases.

2.6 Access Control and Security

- Included table name.
- Included actor names; department chair and secretary.

2.7 Boundary Conditions

- Removed infinite loop between failure and initialization.
- Removed shutdown state since it caused unnecessary complication.

- Fixed inconsistencies in termination state and added more examples of cases in which the system transitions to termination.

3.1 Object Design Trade-offs

- Explained why we preferred one option over the other in more detail and discussed what are the implications of our decisions.

3.2 Final Object Design

- Removed unnecessary Controller objects.
- Re-organized the diagram to make it more clear and readable.
- Re-organized RequestChangePage relations.
- Fixed the relationship between some UI components.

3.3.1 User Interface Management Layer

- Added return types for some methods in UI Management Layer.
- Re-organized diagram to make it more readable.
- Fixed the relationship between some of the page classes.
- Added footnote for why we use repetition in some of the classes.

3.3.2 Web Server Layer

- Removed unnecessary Controller objects.
- Minor changes at the function names of the UserServices and SubmissionService.

3.3.3 Data Management Layer

- Re-organized the design of the data management entity diagram.
    - Added department information to the semester and course classes.
    - Added deadline information for semester and submission.
    - Revised the relationship between user and submission classes.
    - Revised FAQ and announcement classes, added department information.

3.4 Design Patterns

- Removed state design pattern since we decided to implement the logic using enums and differentiate the logic in the service layer.
- Explained how lombok works in builder pattern.

6 References

[1] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.

[2] "2022 Kontenjanlar, 2021 Taban Puanları ve Başarı Sırası," 2022 kontenjanlar, 2021 Taban puanlari ve Basari Sirasi, https://w3.bilkent.edu.tr/web/adaybilgi/2022_kontenjan.html [Accessed: 21-May-2023].

[3] "Using SSL/TLS to encrypt a connection to a DB instance." [Online]. Available: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.SSL.html. [Accessed: 04-May-2023].

[4] W. by: E. Paraschiv, "Password encoding with Spring," *Baeldung*, 26-Nov-2022. [Online]. Available: https://www.baeldung.com/spring-security-registration-password-encoding-bcrypt. [Accessed: 04-May-2023].

[5] Soni, D. (2022, December 12). *What is bulk insert in SQL?*. Scaler Topics. https://www.scaler.com/topics/bulk-insert-in-sql/

[6] "What is nosql? NoSQL databases explained," *MongoDB*. [Online]. Available: https://www.mongodb.com/nosql-explained. [Accessed: 04-May-2023].

[7] "Manage Clusters," *MongoDB*. [Online]. Available: https://www.mongodb.com/docs/atlas/manage-clusters/. [Accessed: 04-May-2023].

[8] "Mongodb Compass," *MongoDB*. [Online]. Available: https://www.mongodb.com/products/compass. [Accessed: 04-May-2023].

[9] "Redis benchmark," *Redis*. [Online]. Available: https://redis.io/docs/management/optimization/benchmarks/. [Accessed: 04-May-2023].

[10] "Amazon Elasticache - Amazon Web Services (AWS)." [Online]. Available: https://aws.amazon.com/elasticache/. [Accessed: 04-May-2023].

[11] "Amazon EC2," *Amazon AWS* . [Online]. Available: Amazon EC2. [Accessed: 04-May-2023].

[12] "Documents," *MongoDB*. [Online]. Available: https://www.mongodb.com/docs/manual/core/document/. [Accessed: 04-May-2023].

[13] "What is a rest api?," *Red Hat - We make open source technologies for the enterprise*. [Online]. Available: https://www.redhat.com/en/topics/api/what-is-a-rest-api. [Accessed: 04-May-2023].

[14] auth0.com, "Jwt.io," *JSON Web Tokens*. [Online]. Available: https://jwt.io/. [Accessed: 04-May-2023].

[15] "Zustand," *npm*. [Online]. Available: https://www.npmjs.com/package/zustand. [Accessed: 04-May-2023].

[16] "ObjectId," *MongoDB*. [Online]. Available:https://www.mongodb.com/docs/manual/reference/method/ObjectId/ [Accessed: 21-May-2023].

[17] *Project Lombok*. [Online]. Available: https://projectlombok.org/. [Accessed: 04-May-2023].