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

First Iteration

May 4, 2023

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

Contents

Project Design Report

Internship Management System

1 Introduction

This report initially 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 mentioning 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 both the students and the academic staff taking part in the internship report evaluation process. In the current system, the internship report evaluation process is managed inefficiently - through three different platforms: Moodle, Google Drive and email. The current system also requires lots of by-hand operations. The aims of this web-based project are to offer a new automated solution to deal with internship reports, eliminate unnecessary manual operations and combine each minor process into one application, therefore 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 will make the system more user-friendly and robust in many aspects.

1.2.1 Information Security & Privacy

The application needs to store lots of confidential information about the 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 [2].
- The passwords of users will be hashed using Bcrypt, and the hashed version will be saved into the database.
- 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 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.

1.2.3 Usability

The internship management system's user interface provides the users with all of the operations that they need to do within the system in a user-friendly, unique, and clear manner. The users will be able to conduct the operations that they are authorized to easily, that is any action should be reached by at most 5 clicks. Since the internship report evaluation cycle could become complex, enabling users to operate without any difficulties within the system is vital. Hence, the user interface of the system is designed with the purpose that the majority of the users will not have any difficulty navigating through the system. By this means, the users will be able to find their desired operation through their dashboards customized for each user type. Operations that are similar or are conducted in the same scope will be displayed together and mostly done simply by clicking a button. The colors of the reports will indicate their status. Red if rejected or requires change, orange if waiting for review, and green if approved. In addition, responsive design approaches will be used so that the system should easily be used via any device regardless of their screen sizes.

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

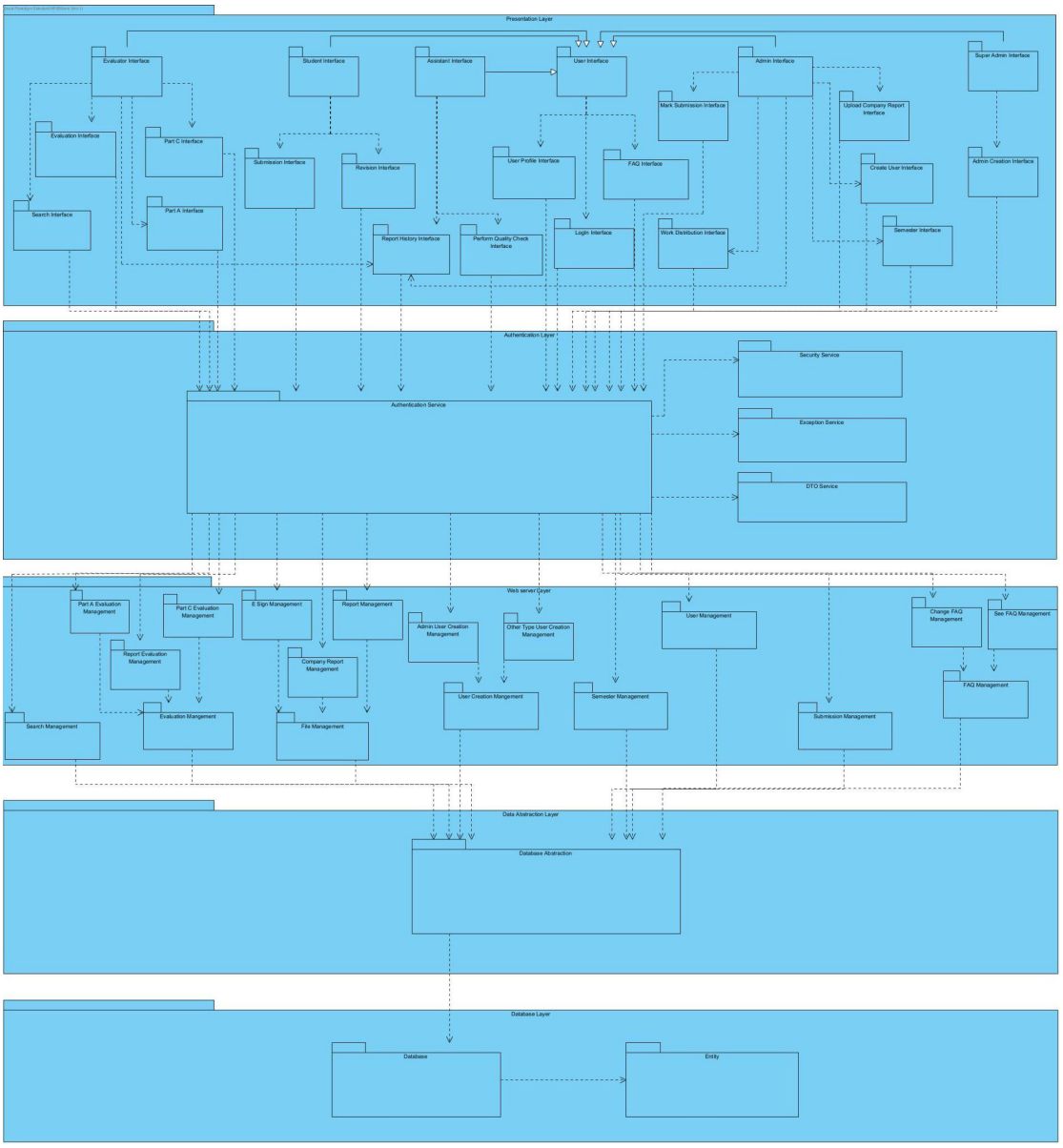

Fig. 1: Subsystem decomposition diagram

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 like 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 work 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.

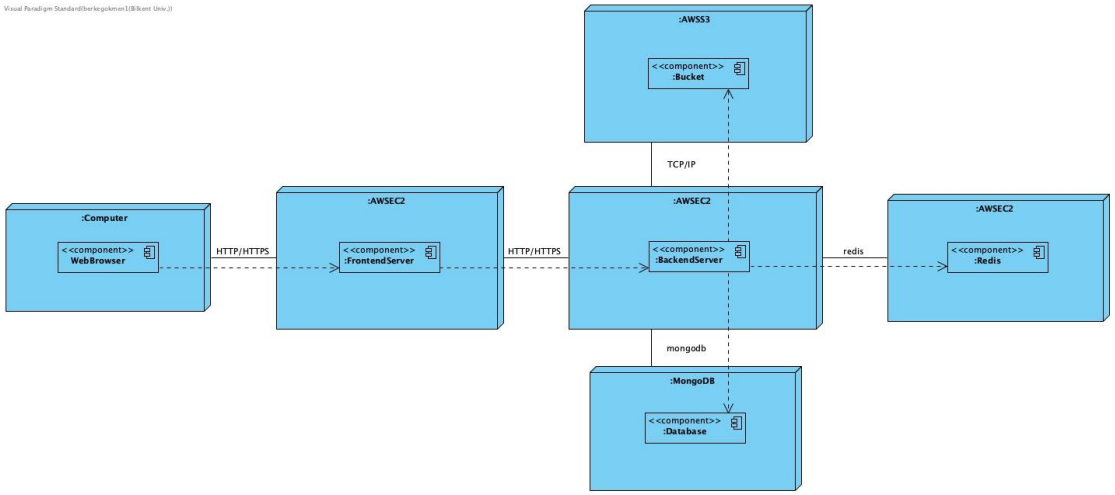

Fig. 2: Deployment diagram

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 MongoDB 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 [3]. 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. Our MongoDB instance is currently running on a M0 cluster [4], which has 500MB of storage. Moreover, the instance can be easily accessed, viewed and edited via MongoDB Compass [5], 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 [6]. 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 is currently managed by AWS Elasticache [7] and runs on a T2 Micro instance which has 1GB of memory and 3.3 GHz Intel Xeon Scalable processor [8].

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*.

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.

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

2.5 Persistent Data Management

MongoDB will be used as the main database to store all the information regarding the Internship Management System. Since MongoDB is a NoSQL database and allows for unstructured and structured data, we can directly map our entity objects to the database documents [9]. Additionally, Amazon S3 (Simple Storage Service) will be used as object storage, specifically to store documents such as student internship reports.

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 [10], that is it does not hold user state, rather it distributes JWT tokens [11] which identify the user. The state of the user is persisted in the frontend application using Zustand [12], a library to manage global state in React applications.

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*) 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 [13].

*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

	Super Admin	Admin (Secretary)	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 Evaluation Report for Student Submission		**X**			

	Super Admin	Admin (Secretary)	Evaluator (Faculty Member)	Assistant (TA)	Student
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**			
Create/Delete Form Question		**X**			
Download Any File in the System		**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.

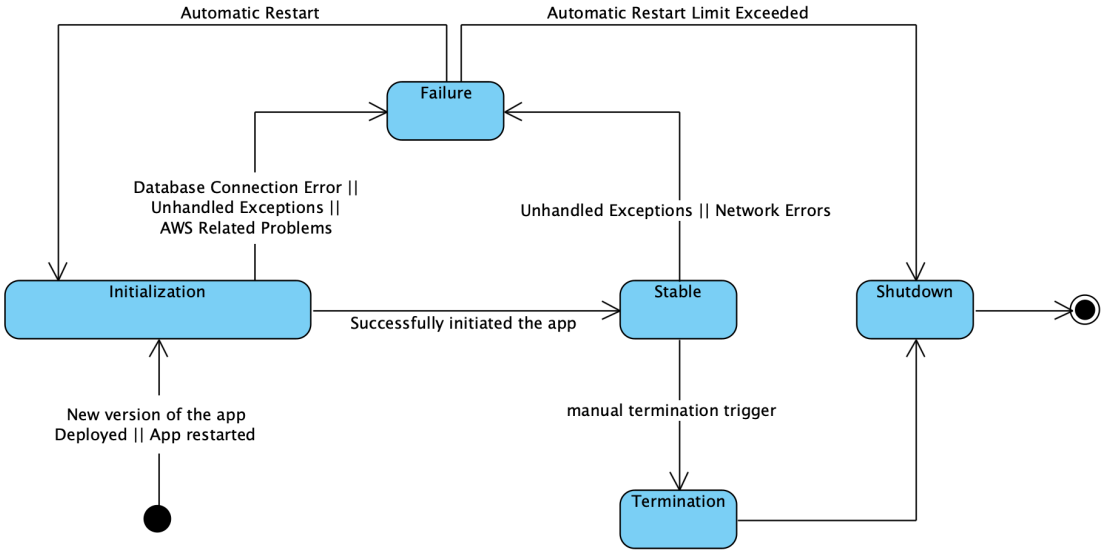

Fig. 3: Boundary conditions diagram

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. However, these types of terminations are handled by the backend, which either exits gracefully or shows a meaningful message to the user. 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.
- The application will shutdown after termination.
- 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.

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 of the app.
- 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 machines hardware related problems that would lead to the termination of the app.

3 Low Level Design

3.1 Object Design Trade-offs

Usability v. Functionality

In the Internship Management System, many of the users will be students who would be not familiar with the system. Therefore, IMS needs to be simple enough for new users to not have confusions about process or the usage. Also, faculty members will only use the system for simple actions like downloading a report or grading it, therefore there is no need to have many functionalities. Since main communication between faculty members and students would be provided by feedback and announcements, the system didn't require further systems. Only users that would use the system continuously will be admin users, who are department secretaries, and we will provide decent functionality for them to manage the general process.

Rapid Development v. Automation

In the Internship Management System, the main goal is to automate the application evaluation process. Therefore, we aim to completely automate the evaluation and communication process. 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 application. For these automation processes, we designed a more complicated system than what it would require for a more manual system, and therefore, this will decrease the development speed of our system.

Scalability v. Rapid Development

In the Internship Management System, we designed the system to cover up all departments in engineering faculty, and therefore to have a high number of active users. Because of this high user number, we decided to store file data in a cloud system, AWS S3, to reduce the burden in our database system. Also, we decided to keep the evaluation form questions in our database and provide a semi-dynamic system to be able to provide for any department which has different evaluation forms. Also, we decided to use Redis as cache for authorization to decrease burden in the database to be able to support a high number of users. All these factors will reduce our development speed but increase the scalability of the system.

Security v. Performance

In the internship management system, we use encrypted passwords and blacklisted access tokens to make our systems secure. That is, when the account of an user is created, their password is saved into the database in a hashed form, which allows it to protect data in case of any data leak. However, in every login request, this feature requires rehashing the password and comparing it with the hashed one in the database, thereby decreasing the performance of the system and increasing the response time slightly. Following that, in every request, the access token of the user is checked to see if it’s blacklisted. Blacklisting operation occurs when the user logs out of the system and it is needed since JWT tokens do not expire on their own. Since the token is checked in every request, it slight increases the response times of the service, however this is minimized by using Redis as the blacklist store to ensure that this operation takes minimum time.

3.2 Final Object Design

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

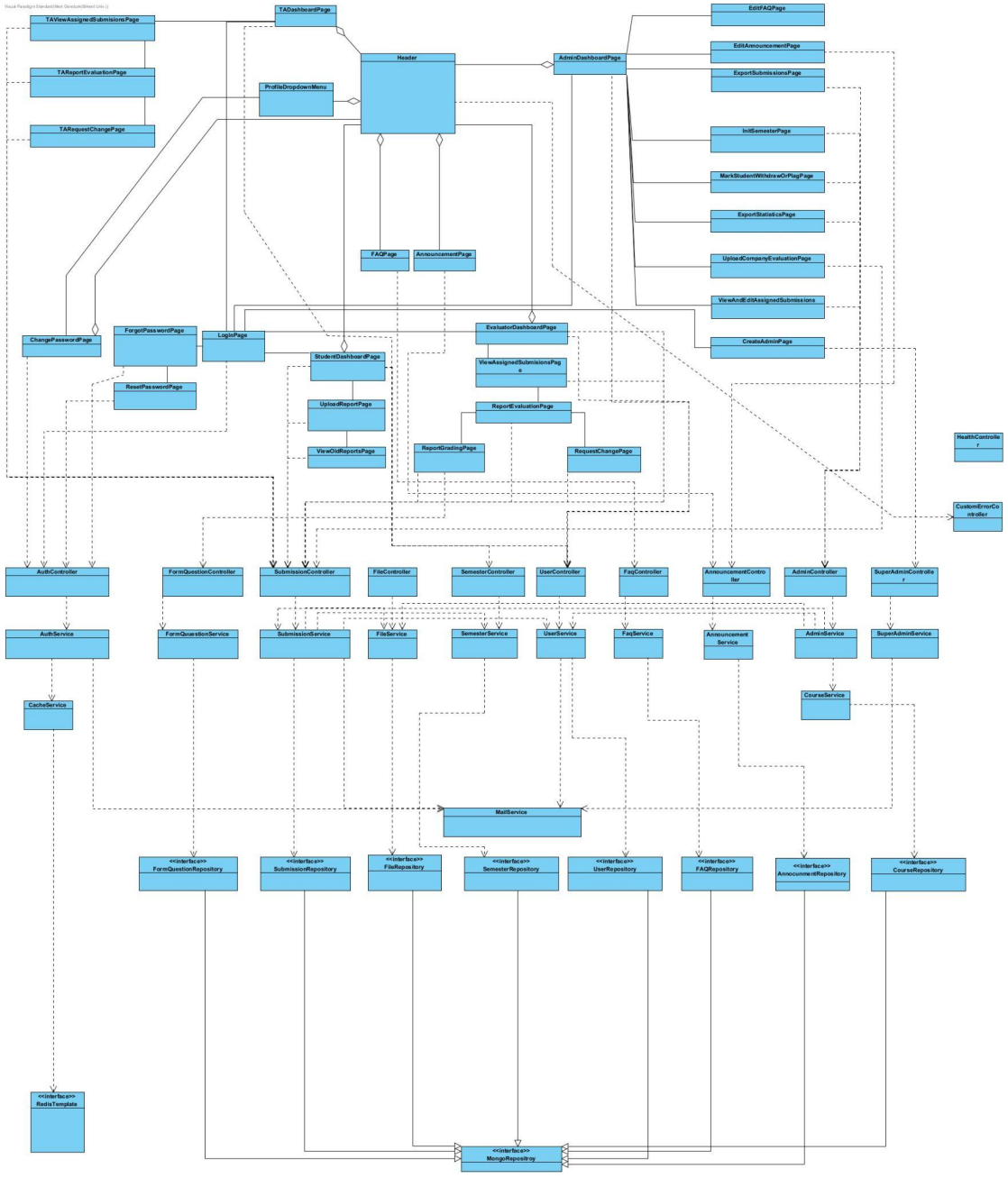

Fig. 4: Final object design diagram

3.3 Layers

3.3.1 User Interface Management Layer

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

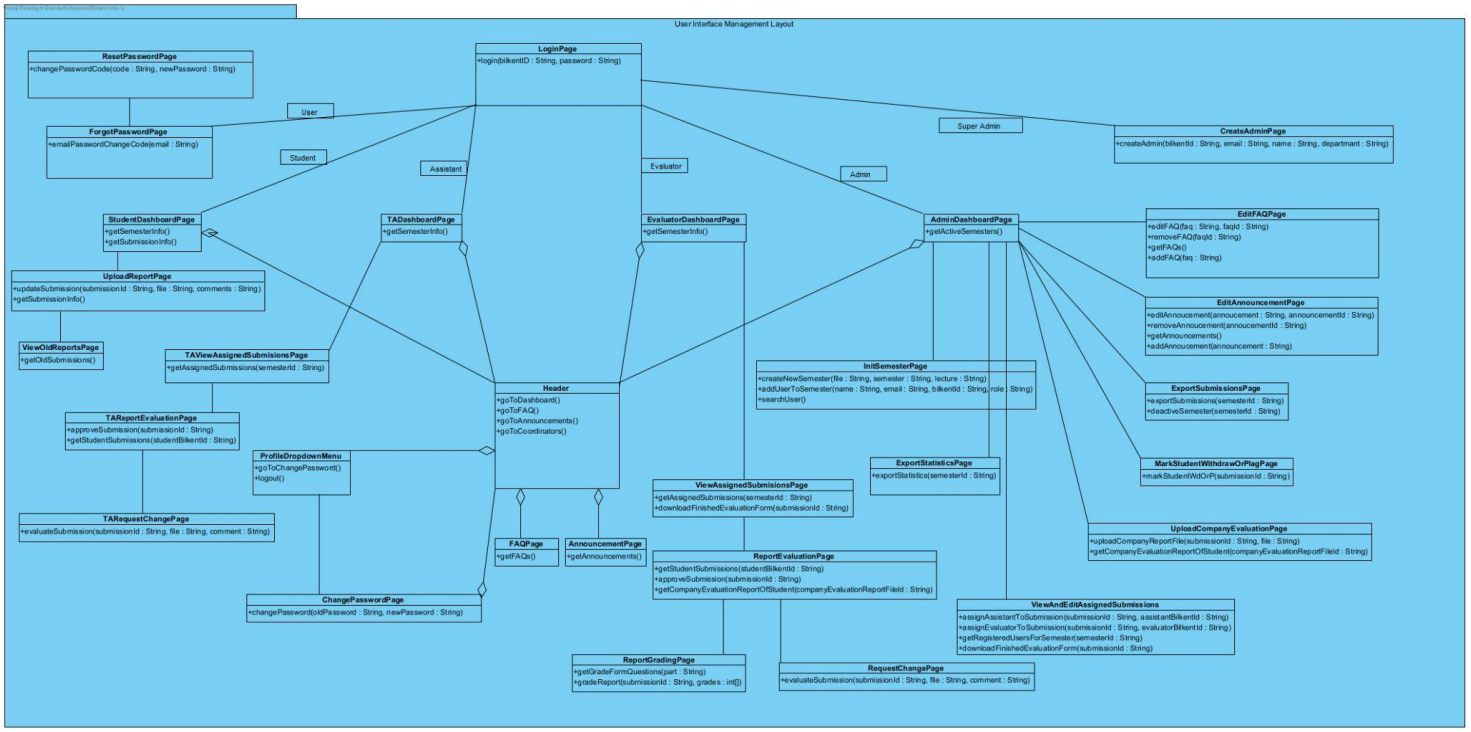

Fig. 5: User Interface Management Layer

3.3.2 Web Server Layer

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

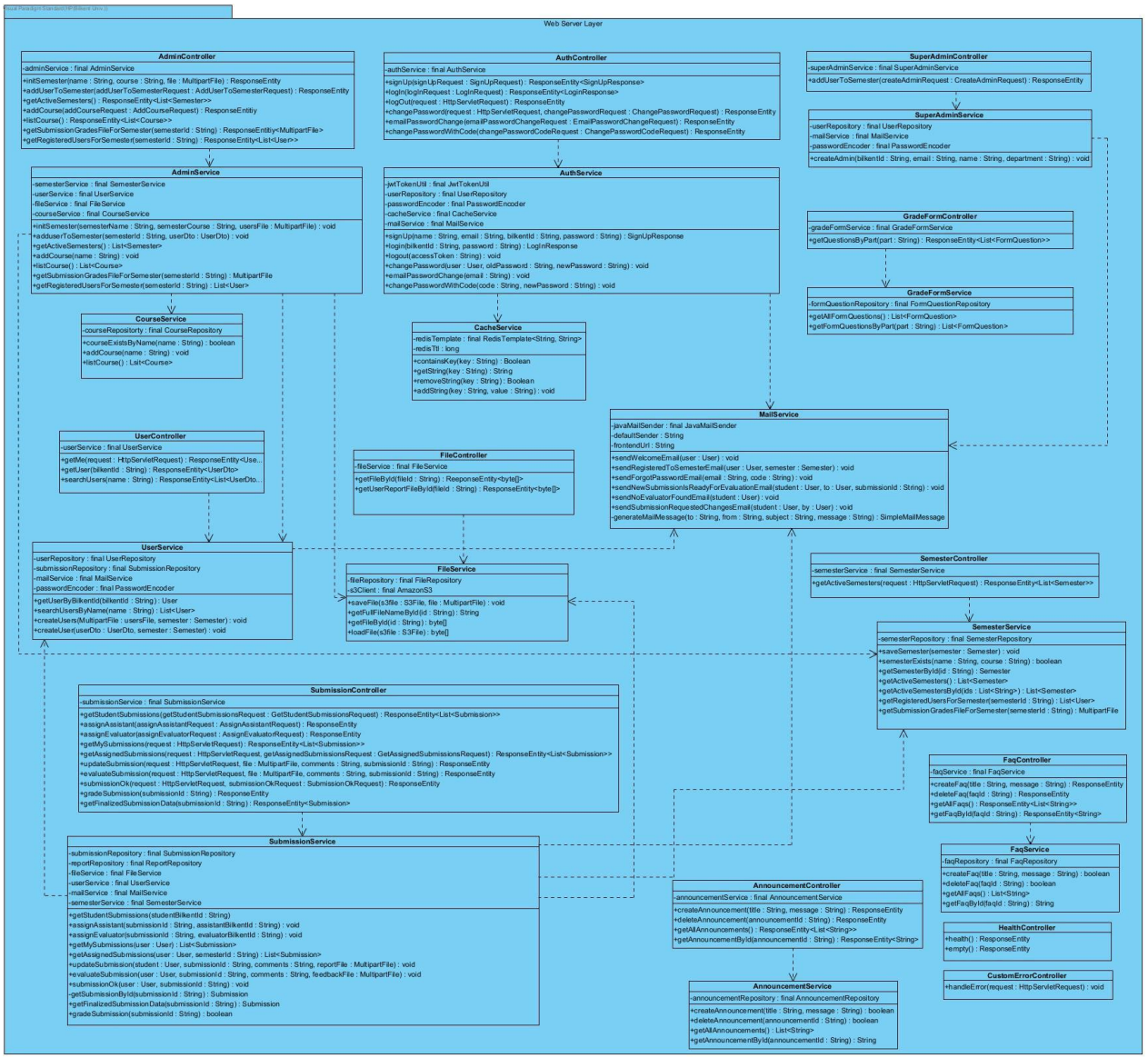

Fig. 6: Web server layer diagram

3.3.3 Data Management Layer

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

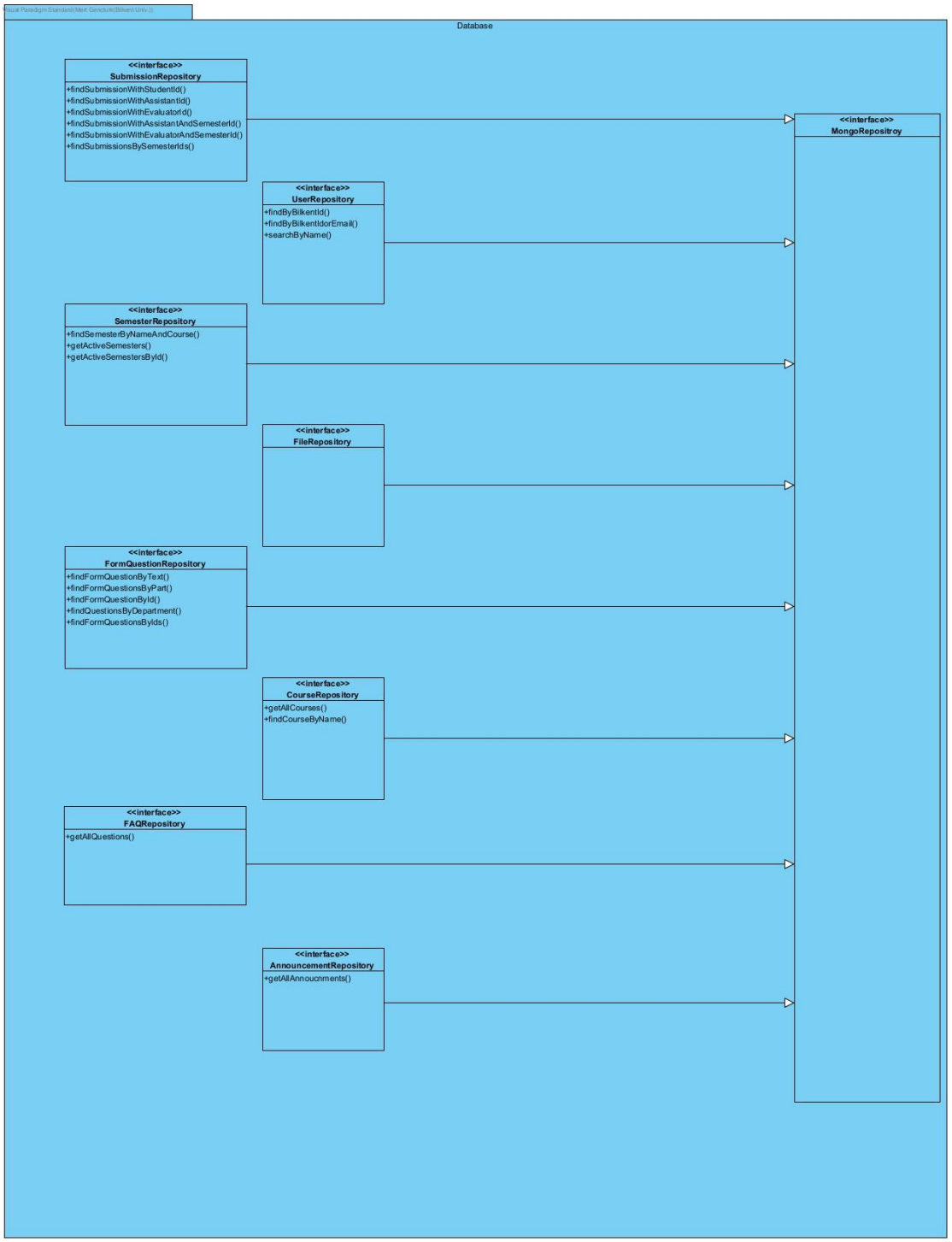

Fig. 7: Data Management Layer - Repository Diagram

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.

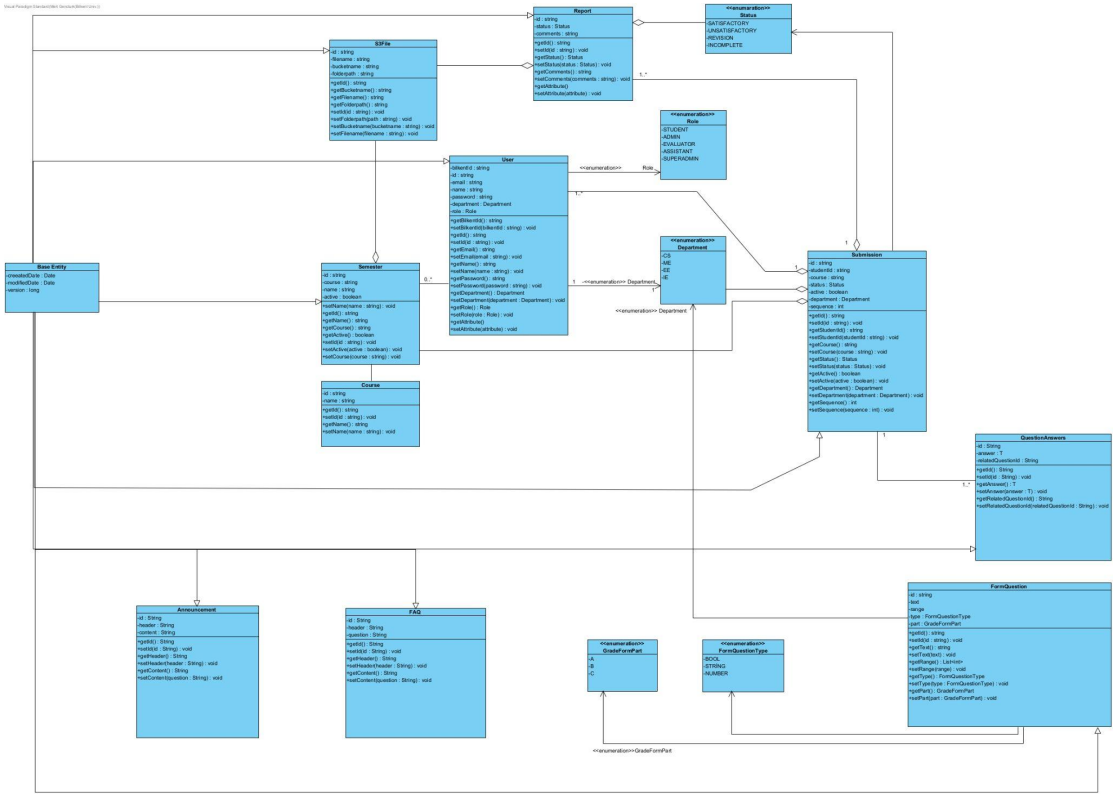

Fig. 8: Data Management Layer - Entity Diagram

3.4 Design Patterns

3.4.1 Builder Pattern

Throughout the application, in the backend, we create objects of all sorts of types. Since most of the objects have optional parameters, overloading the constructor for each of them is extremely cumbersome. That's why we decided to use the builder pattern in all of our object constructions. This pattern allows us to build the objects step by step and leave some fields empty as needed. In order to implement the builder pattern, we decided to use a real-world elegant solution instead of implementing all the builder classes from scratch. We've utilized the power of a package called Lombok [14] which provides automatic builder construction for classes as well as auto-implementing getters and setters.

3.4.2 State Pattern

In the internship management application, the reports of students, and the feedback reports of evaluators (or assistants) are extremely important. In each submission, we have a parameter called status, indicating the status of submission. Depending on what state the submission is in, some actions are restricted and some are allowed. For instance, when the status of a submission is graded, the student can no longer upload any reports or change anything related to their submission. Moreover, when the assistant approves the submission report to be passed onto the evaluator, the assistant can no longer change the status of the submission.

3.4.3 Facade Pattern

Since the internship management is a complex application, we decided to separate the logics of different operations and implement a more layered architecture. There are two places in which we separate the logic. One is between controller and service layers and other is between service and database layer. For the controllers and service layers, there are two classes. The controller handles the request, parses it and calls the service layer with the required information and then returns the result as response. That is, if the implementation of a service layer changes, there will be minimum modification required to the controller layer. The same principle goes for the database and service layer as well. For the database access we use repository classes which encapsulate the database query logic. There is absolutely zero database logic inside the service layers. Thus, it'll be possible to change/modify database queries without affecting the actual implementation of the service layer.

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.

5 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] "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].

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

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

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

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

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

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

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

[10] "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].

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

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

[13] 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].

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