Bilkent University

CS-319

Object-Oriented Software Engineering

Spring 2022/2023

DESIGN REPORT ITERATION 1

Group “BOSGII”

Instructor: Eray Tüzün

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

05/05/2023

Member Name	Student ID
Göktuğ Yılmaz	21903048
Berkay İnceişçi	21802088
Oğuz Kuyucu	21902683
Safa Eren Kuday	21902416
İlayda Zehra Yılmaz	22001769
İsmail Emre Deniz	21901913

1. INTRODUCTION

Our project aims to develop a web application dedicated to the evaluation process of internships, specifically for students and internship report graders affiliated with the Bilkent Engineering Faculty. The primary objective of the project is to automate the manual processes that currently exist in the system. This website allows students and graders to expect a smoother experience, saving time and reducing errors.

1.1 Purpose of the system

The purpose of the system is to enhance the overall experience of all parties involved in the process of handing in and grading the internship reports, including the department secretary, instructors/graders, and students. By automating and simplifying the tedious work typically handled by the department secretary, the system aims to streamline the entire process, saving time and reducing errors. Additionally, the platform will provide grader and student-friendly interfaces, allowing them to easily manage their tasks and stay informed throughout the process. Ultimately, the goal is to create a more efficient and user-friendly internship management system.

1.2 Design goals

1.2.1 Usability

To ensure a positive user experience, our web application should prioritize usability in order to fully meet the expectations that come with an entirely new system. This means designing the web page with a clear and intuitive layout, consistent design elements, and minimal as possible while being a functional interface. To enhance user experience, we will ensure that all functionalities in our web application can be accessed with a maximum of 6 clicks. Also, the average active time it takes for users to complete simple tasks and find specific information within the app should not take more than 10 minutes in average. For more complex tasks like answering feedback comments average time should be less than 20. For readability, font sizes of headings should be more than 18 and for other lines it should be more than 12. Moreover, only the actions that are specific to the user type will be displayed, further reducing any unnecessary elements on the interface. For example, the students are only allowed to see their internship and report information, the evaluators are only allowed to see their list of students and their work, etc. This approach will simplify the user experience, enabling users to easily find and access the functionalities they need without encountering any difficulties.

1.2.2 Maintainability

Maintainability is crucial for the effective functioning of our web application. As with many web applications, ours will be a complex system that is likely to undergo numerous changes throughout its lifecycle. These changes can include the addition of new features, bug fixes, updates to existing functionality, and enhancements to performance. To ensure effective maintenance, we should focus on four key attributes:

Well-organized code: The codebase should be structured and organized in a logical manner, making it easier to understand, modify, and maintain. A well-organized codebase reduces the likelihood of introducing bugs and facilitates efficient collaboration among developers.

Clear documentation: Comprehensive and up-to-date documentation is essential for maintaining a web application effectively. It should include detailed information about the system architecture, code structure, APIs, and any other relevant aspects. Clear documentation enables developers to understand the application's intricacies and make informed modifications or fixes.

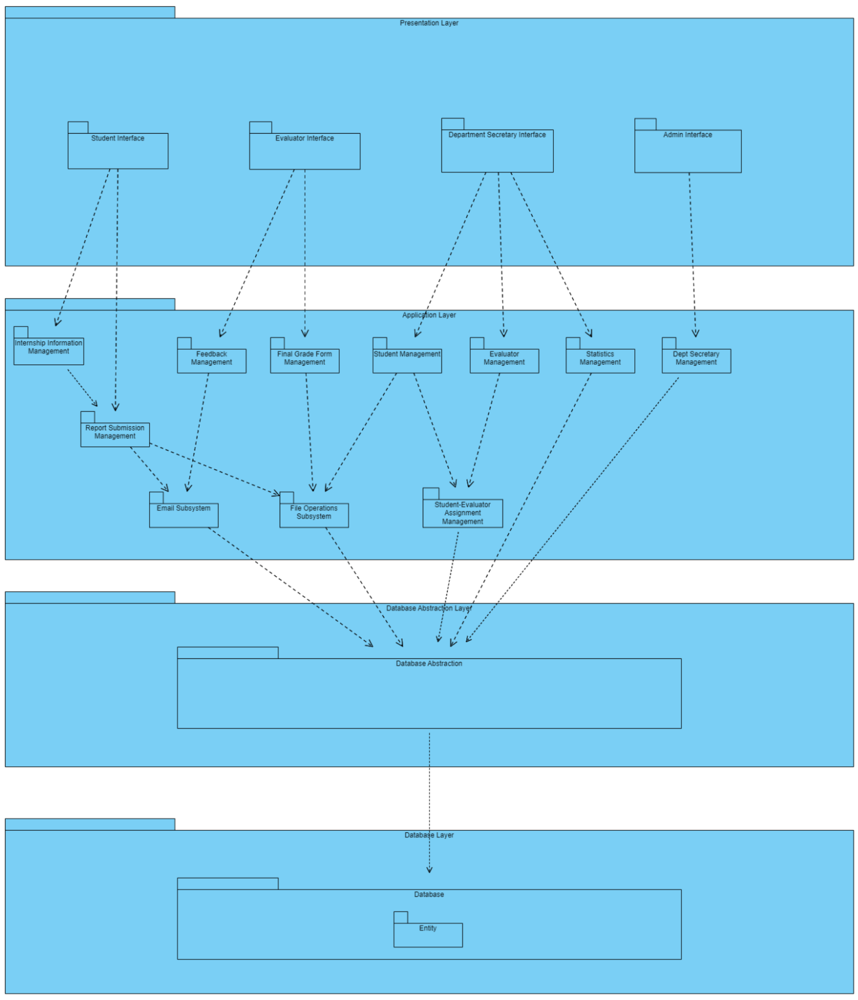
Standardized development practices: Consistent and standardized development practices promote maintainability. This includes using version control systems, following coding conventions, conducting code reviews, and employing automated testing. Standardized practices enhance code quality, reduce errors, and streamline the maintenance process.

Scalability and extensibility: Designing the web application with scalability and extensibility in mind ensures that it can accommodate future changes without significant rework. By adopting a modular architecture and utilizing scalable technologies, such as cloud computing, the application can easily adapt to evolving requirements and handle increased user loads.

By prioritizing these attributes, we can ensure that our web application remains maintainable throughout its lifecycle, allowing for efficient updates, bug fixes, and enhancements. This, in turn, enables the application to function effectively and meet the evolving needs of its users.

2. HIGH-LEVEL SOFTWARE ARCHITECTURE

2.1 Subsystem decomposition



Presentation Layer
Student Interface
Evaluator Interface
Department Secretary Interface
Admin Interface
Application Layer
Internship Information Management
Feedback Management
Final Grade Form Management
Student Management
Evaluator Management
Statistics Management
Dept Secretary Management
Report Submission Management
Email Subsystem
File Operations Subsystem
Student-Evaluator Assignment Management
Database Abstraction Layer
Database Abstraction
Database Layer
Database
Entity

Figure 1: Subsystem Decomposition Diagram

Our internship report management system is divided into five different layers. The first layer is the Presentation layer, which is responsible for providing an interface for each type of user in the system - students, evaluators, department secretaries, and admin. Each user will have their own specific interface that they can use to access the system's features. Moving down the diagram, we have the Authentication layer, which is responsible for ensuring that only authorized users can access the system. This layer includes an authentication service and Spring Security, which work together to provide a secure login process. The third layer is the Application layer, which contains the core logic of the system. This includes classes which are responsible for handling the business logic of the system. The fourth layer is the Database Abstraction Layer, which provides an abstraction over the underlying database. This layer allows the system to communicate with the database without affecting the application code. Finally, we have the Database Layer, which is responsible for storing and managing the data required by the system. By dividing our system into these different layers, we can ensure that each component is modular and can be managed and maintained independently.

2.2. Hardware Software Mapping

For the hardware configuration, the database server and web server are going to provide the necessary hardware resources, such as CPU, memory and disk space for the database. Web server uses them to run the application logic of the web application and the database uses them to store and retrieve data for the web application. Amazon Web Services (AWS) is chosen for an appropriate web server and it includes virtual machines (EC2 instances). For the database server Amazon RDS is going to be used, as well.

In addition to the hardware configuration mentioned above, our web app has specific hardware requirements to ensure optimal performance. Initially, we expect 400 students and 50 instructors/secretaries to use our system. As the other engineering departments start to use the system, the user base expands to 1800 (450 x 4). The hardware should be able to accommodate the increased load and maintain the similar level of performance. So, the hardware setup should be capable of handling their concurrent requests and providing a good user experience. We know that Amazon Web Services (AWS) offers Elastic Load Balancing (ELB) and Auto Scaling features that will help us adjust capacity as needed. Amazon RDS, being a managed database service, can help in scaling the database resources vertically or horizontally to handle the increased workload. Also, AWS provides options to choose the appropriate instance types with the required CPU and memory specifications. As a result, our hardware configuration are enough to handle the initial user load of 450 users and it can scale up to accommodate 1800 users without compromising much about performance and user experience.

Now, let’s look at node responsibilities and communication between nodes. Amazon Web Services is responsible for handling incoming requests from users and serving the necessary web pages and information to the users. It is also responsible for the application logic of the web application, so on the user side, it is going to process user input, interpret them and generate responses accordingly, and on the database side it is going to retrieve necessary data and store new or updated information. Amazon RDS is responsible for storing and retrieving data for the web application. It is going to listen to requests from the web server and send data asked accordingly. Also, it is responsible for storing data sent by AWS. Since we use MySQL, this communication is done using MySQL.

Frontend, the client-side of the application, is built using React, a JavaScript library for building user interfaces. The React application runs on the client-side, in the web browser, and communicates with the server-side using HTTP protocol. The server-side of the application is built using Java and the Spring Boot framework, which provides a set of tools for building web applications. The Spring Boot runs on EC2 instances and listens for incoming HTTP requests from the client-side of the application. The Spring Boot application communicates with the database server hosted on Amazon RDS using MySQL. The database server stores application data and responds to requests from the Spring Boot application.

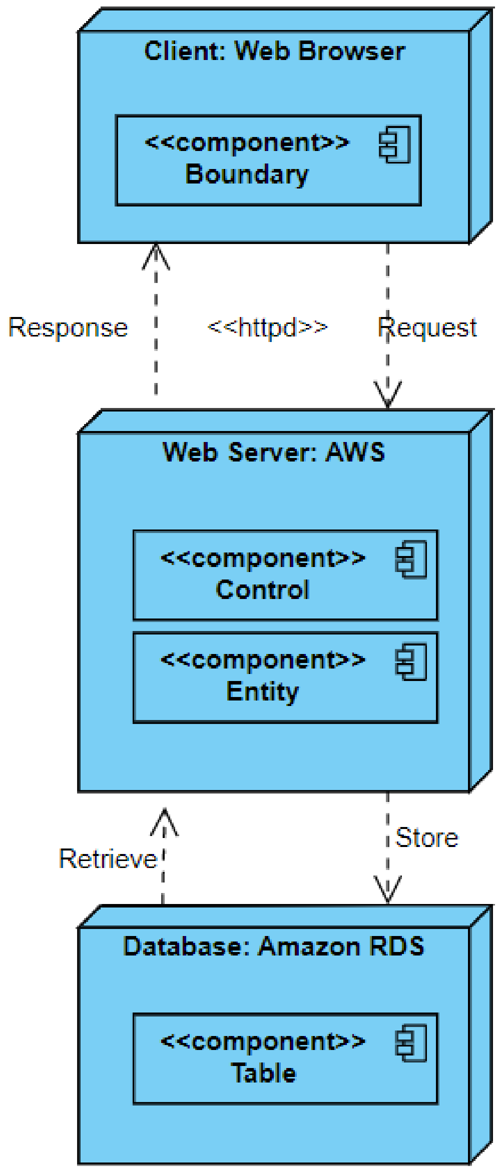


Figure 2: Deployment Diagram

2.3. Persistent Data Management

Persistent data management is the process of storing, retrieving, and managing data in a way that ensures its long-term availability and persistence. Our system has to be able to be restored in case it crashes we shut it down in a controlled manner. Therefore, we need persistent data management for our long-living objects. Above all, we have to maintain the data that is required for throughout all system executions rather than a single execution, such as entity objects and required info to execute the system.

We are using MySQL as a database language for our web application, and our database design involves storing the data related to entity objects. This ensures that the necessary information is stored in the database. These persistent data will be students, instructors, internships, submissions, versions, comments, reports, grades. This enables our system to be consistent because we keep the information in the database that is necessary for persistence of the system. That's why our system is able to bring this data every time required and to restore itself in case of a shutdown.

Additionally, MySQL provides mechanisms for data backup, replication, and recovery. These mechanisms can help ensure that the data is not lost due to hardware failure, software bugs, or other unexpected events. Additionally, MySQL provides security features to ensure that the data is protected from unauthorized access and malicious attacks.

Also, since it is a relational database, it is highly reliable due to its structured and organized approach to storing and managing data. It ensures data integrity through constraints and enforces consistency through transactions. With standardized querying capabilities, relational databases offer consistent and accurate data retrieval. These features collectively make relational databases a reliable choice for storing and managing data.

2.4. Access Control and Security

Our Internship Manager web-based application, BIM, contains important information such as mails, IDs, passwords, and reports of the students. So securing the application and data has great significance. Therefore, each user has different authorization and verification levels, which will be handled by Spring Security. Authorization achieved by attaining roles for users. Each of the users has certain roles assigned to them. These roles are student, instructor, teaching assistant, secretary, and admin.

All users will be able to see just their user interfaces. For example, users whose role is a student cannot see the secretary home page. Same for the secretary. It will be handled on the client side. In addition, external users will not be able to access the system as all users will be created and granted access by the system, which increases the security of the system and provides admin to control access of users.

On the server-side, we will implement a system that assigns a user's role upon registration to the system. Later, when a user attempts to log in, their user type information is accessed from the database and sent to the client-side. Furthermore, each user can only make specific requests to the server-side, as we restrict certain functionalities to certain user types. For instance, a student cannot grade the internship report, a secretary cannot provide feedback, and the admin cannot view or change the passwords of other users, which are hashed to keep user information safe.

Access Control Matrix

	Student	Instructor	TA	Secretary	Admin
Login	x	x	x	x	x
Change password	x	x	x	x	x
Upload report	x				
See grade	x				
View feedback report	x				
View status of feedback report	x				
Generate final PDF		x			
View assigned student list		x	x		
Download student internship report		x	x	x	
Drop Selected Student		x			
Give feedback		x	x		
Grade report		x			
View statistic				x	
Upload company evaluation form				x	

View student list				x	
View submissions of student		x		x	
Assign student to grader				x	
Initialize evaluators				x	
Initialize students				x	
View final PDF				x	
Reassign student the grader				x	
Initialize secretaries					x
close secretaries					x
view secretaries					x

Figure 3:Access Control Matrix

2.5 Boundary Conditions

Here is an overview of our system's boundary conditions. Our goal is to have few failures overall. We want to create a recovery mechanism for system failures that would return us to the state in which failure occurred.

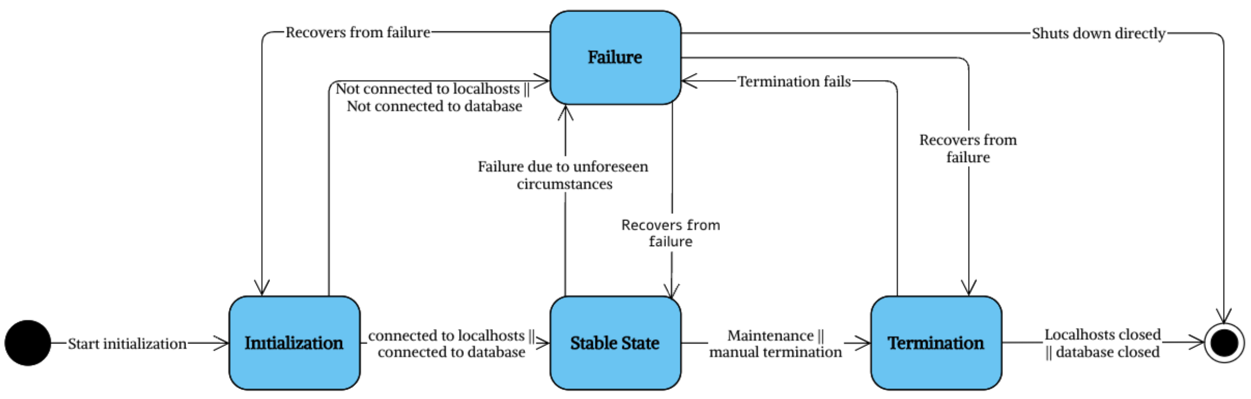

Recovers from failure
Failure
Shuts down directly
Not connected to localhosts || Not connected to database
Termination fails
Recovers from failure
Failure due to unforeseen circumstances
Recovers from failure
Start initialization
Initialization
connected to localhosts || connected to database
Stable State
Maintenance || manual termination
Termination
Localhosts closed || database closed

Figure 4: Boundary Conditions

2.5.1 Initialization

For development purposes, the services provided by the development tools in the local host will be used. The initialization process of the Bilkent Internship Management, BIM, application is going to start with running the MySQL Workbench program to start the MySQL database. Then, the com.bosgii.internshipmanagement file will be executed in the source file to make the backend accessible, enabling the REST API to function. The command "npm start" will be used to launch the React project for the front-end web application. Check the followings to start the application.

- ☐ Opening MySQL Workbench application
- ☐ Running the Spring Boot (backend) Java source code
- ☐ Establishing MySQL database connection
- ☐ Running React (frontend) source code - shows browser screen of web application
- ☐ Checking the communication between the backend and frontend (REST API)

2.5.2. Termination

Termination of the project can be initiated by the admin because of maintenance or may occur due to unforeseen circumstances. In both cases, data loss is an undesirable situation for the application, and it must be addressed during the design phase. To ensure proper termination and avoid any failure, a specific order of steps should be followed.

If the admin decides to shut down the system, the reverse order of initialization should be used to correctly terminate the application. First, terminate the React application by typing "ctrl + c" in the terminal. Next, stop the BIM InternshipManagementApplication.java program. Finally, stop the database by using MySQL Workbench.

Following these steps will ensure that the project is correctly terminated, minimizing the risk of data loss or other issues that may arise during the termination process. Check the followings to terminate.

- ☐ Terminating React Application

- [ ] Stopping running InternshipManagementApplication.java file
- [ ] Backuping the database
- [ ] Closing the database connection
- [ ] Closing MySQL Workbench

2.5.3 Failure

If a software problem or unexpected event occurs, the user will be directed to the login screen to prevent the system's database from becoming corrupted by invalid data or other issues. Implementing this strategy can significantly enhance the system's security and reliability. The project's objective is to minimize the occurrence of unexpected events that could potentially degrade usability and user experience.

Thanks to the use of a three-layered architecture, failures are less likely to occur in multiple layers at the same time. In most scenarios, the application can be recovered directly to a stable state.

3. LOW-LEVEL DESIGN

3.1. Object Design Trade-Offs

Rapid Development vs. Functionality

In our internship management system, there is the necessity to develop our system in a limited time but also there is the necessity to add a lot of functionalities for different users (admin, students, instructors, etc.) who need some functionalities for internship reports (submitting, commenting, replying, etc.). In this case, developing these many functionalities in a limited time is hard so we have to choose if we should spend more time and add more functionalities or spend less time and add not all of the functionalities. In this case, the functionality of the internship system is much more important than spending less time to create rapid development because adding all of the functionalities is necessary for the internship management system to work correctly. Thus, we should create a slower development and spend more time writing the program to add more functionality.

Functionality vs. Usability

The internship management system has many attributes that might lower the usability of the system when they all are implemented. For example, the system needs to enable students to generate internship reports, reply to instructor comments and add revised reports as well as letting the students see their old revisions. If they are taking two internship classes, they should be able to take action in both of them. This is just some functionalities for student users, and it is already very complex. Adding these functionalities and adding many others to the other user roles increases the complexity of the system which lowers the usability of the system. However, we chose that the users should be able to perform complex functionalities even if it reduces usability.

3.2 Final Object Design

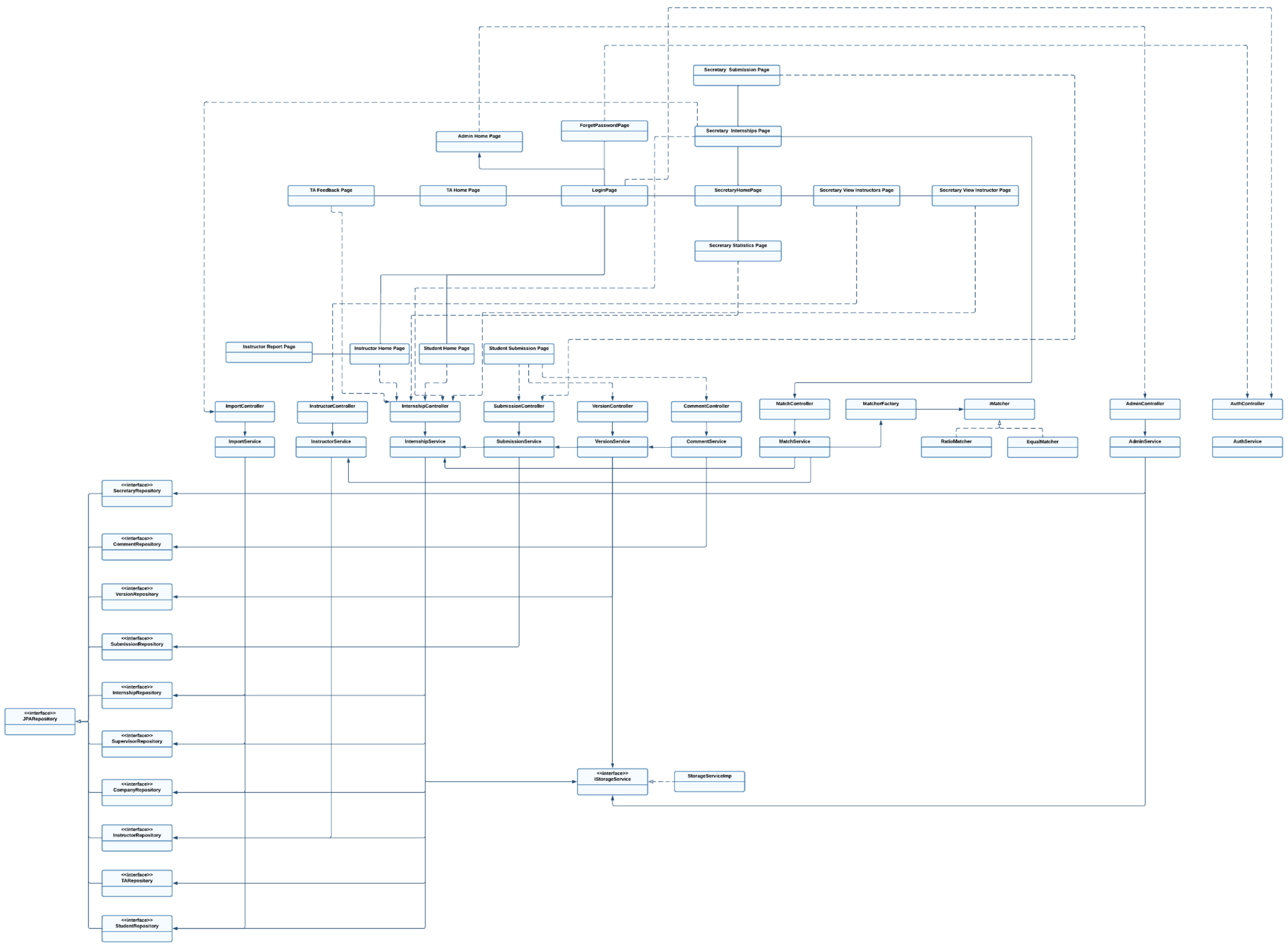


Figure 5: Final object design

Link: https://www.hizliresim.com/rpyo641

3.3. Packages

3.3.1 User Interface Diagram

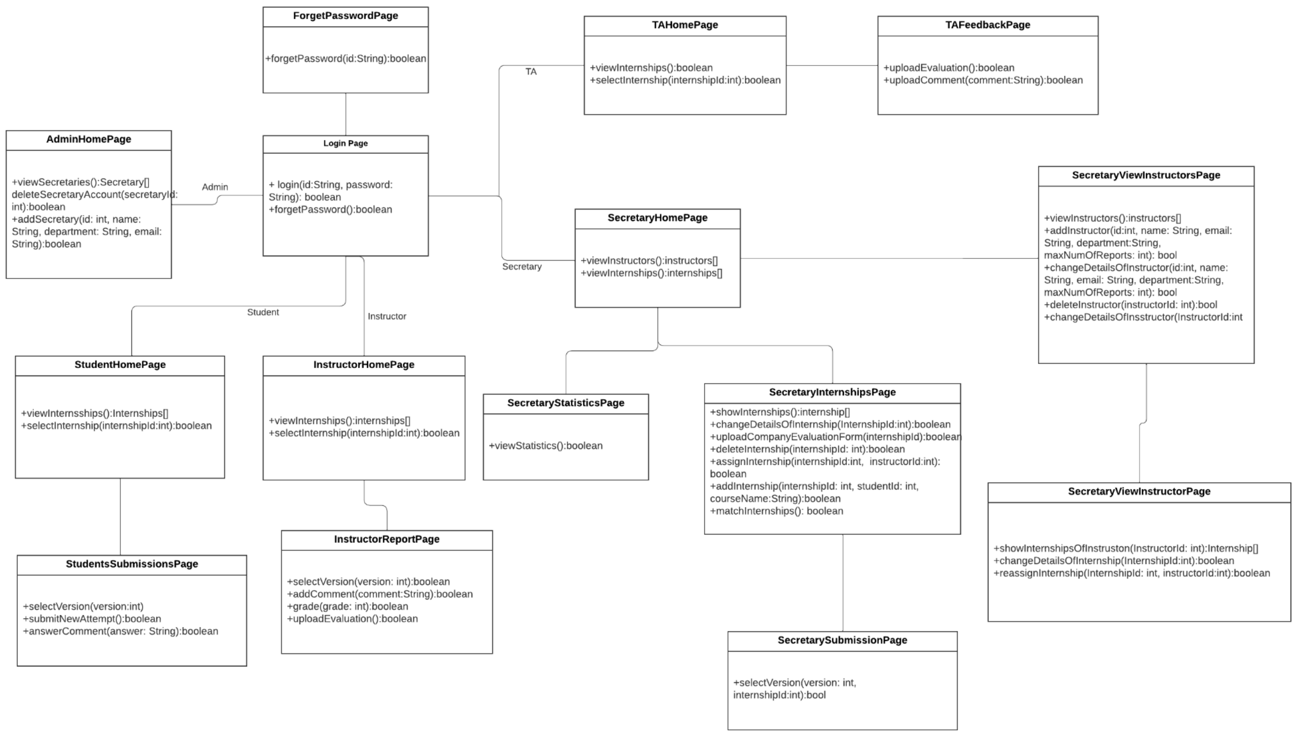
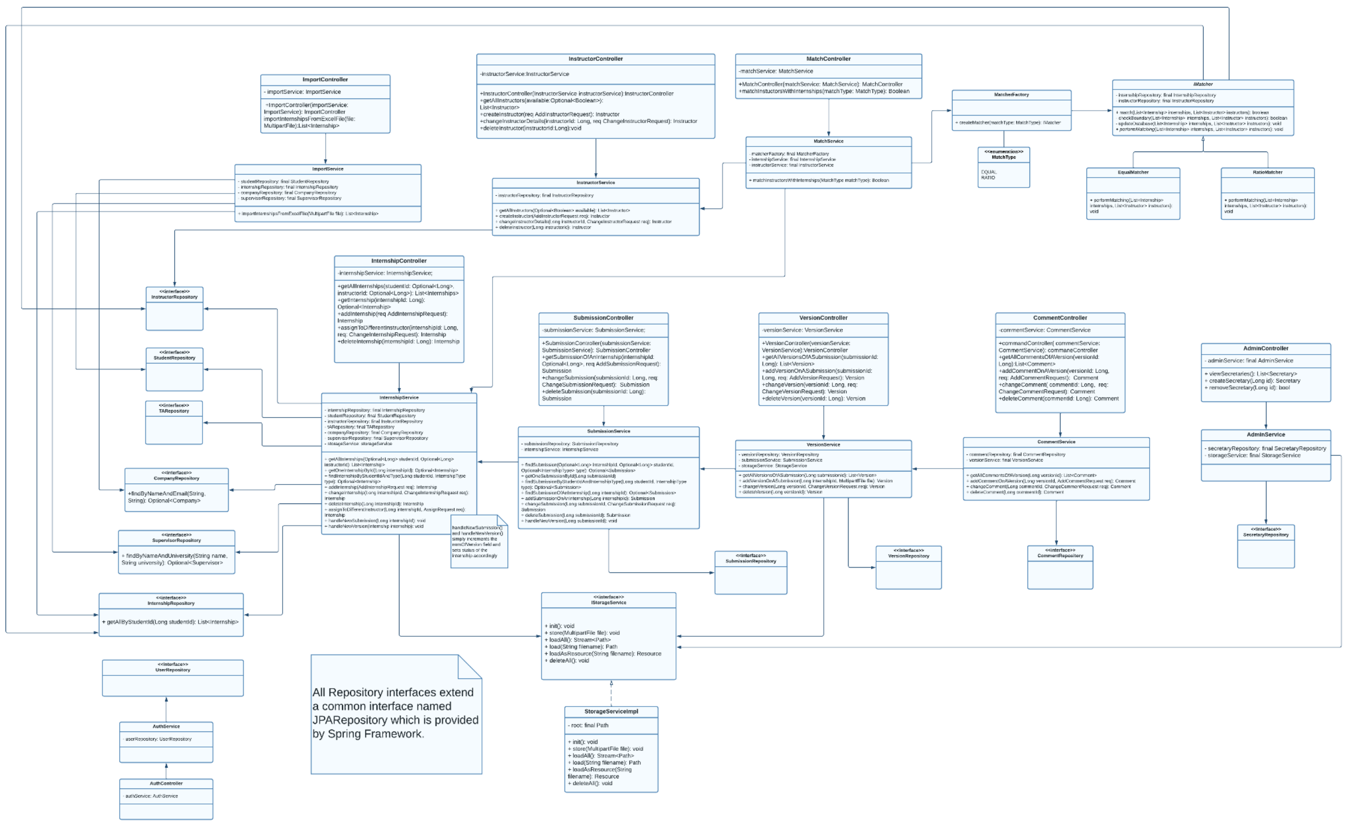

ForgetPasswordPage
+forgetPassword(id:String):boolean
TAHomePage
+viewInternships():boolean
+selectInternship(internshipId:int):boolean
TAFeedbackPage
+uploadEvaluation():boolean
+uploadComment(comment:String):boolean
TA
AdminHomePage
+viewSecretaries():Secretary[]
deleteSecretaryAccount(secretaryId: int):boolean
+addSecretary(id: int, name: String, department: String, email: String):boolean
Admin
Login Page
+ login(id:String, password: String): boolean
+forgetPassword():boolean
SecretaryViewInstructorsPage
+viewInstructors():instructors[]
+addInstructor(id:int, name: String, email: String, department:String, maxNumOfReports: int): bool
+changeDetailsOfInstructor(id:int, name: String, email: String, department:String, maxNumOfReports: int): bool
+deleteInstructor(instructorId: int):bool
+changeDetailsOfInsstructor(InstructorId:int
SecretaryHomePage
+viewInstructors():instructors[]
+viewInternships():internships[]
Secretary
Student
Instructor
StudentHomePage
+viewInternsships():Internships[]
+selectInternship(internshipId:int):boolean
InstructorHomePage
+viewInternships():internships[]
+selectInternship(internshipId:int):boolean
SecretaryStatisticsPage
+viewStatistics():boolean
SecretaryInternshipsPage
+showInternships():internship[]
+changeDetailsOfInternship(InternshipId:int):boolean
+uploadCompanyEvaluationForm(internshipId):boolean
+deleteInternship(internshipId: int):boolean
+assignInternship(internshipId:int, instructorId:int): boolean
+addInternship(internshipId: int, studentId: int, courseName:String):boolean
+matchInternships(): boolean
SecretaryViewInstructorPage
+showInternshipsOfInstruston(InstructorId: int):Internship[]
+changeDetailsOfInternship(InternshipId:int):boolean
+reassignInternship(InternshipId: int, instructorId:int):boolean
StudentsSubmissionsPage
+selectVersion(version:int)
+submitNewAttempt():boolean
+answerComment(answer: String):boolean
InstructorReportPage
+selectVersion(version: int):boolean
+addComment(comment:String):boolean
+grade(grade: int):boolean
+uploadEvaluation():boolean
SecretarySubmissionPage
+selectVersion(version: int, internshipId:int):bool

Figure 6:User Interface Diagram

Link : https://www.hizliresim.com/kfpwz08

3.3.2 Web Server Layer


ImportController
InstructorController
MatchController
MatcherFactory
Matcher
ImportService
InstructorService
MatchService
<<enumeration>> MatchType
EqualMatcher
RatioMatcher
InternshipController
<<interface>> InstructorRepository
<<interface>> StudentRepository
<<interface>> TARepository
InternshipService
SubmissionController
VersionController
CommentController
AdminController
SubmissionService
VersionService
CommentService
AdminService
<<interface>> CompanyRepository
<<interface>> SupervisorRepository
<<interface>> InternshipRepository
<<interface>> SubmissionRepository
<<interface>> VersionRepository
<<interface>> CommentRepository
<<interface>> SecretaryRepository
<<interface>> IStorageService
<<interface>> UserRepository
AuthService
AuthController
StorageServiceImpl

All Repository interfaces extend a common interface named JPARepository which is provided by Spring Framework.

Figure 7: Web Server Layer Diagram

Link: https://www.hizliresim.com/ck9ylrn

Request classes such as "AddRevisionRequest" or "UpdateInternshipRequests" are classes without methods, they are used to map the HTTP request messages to POJO's.

3.3.3 Data Management Layer

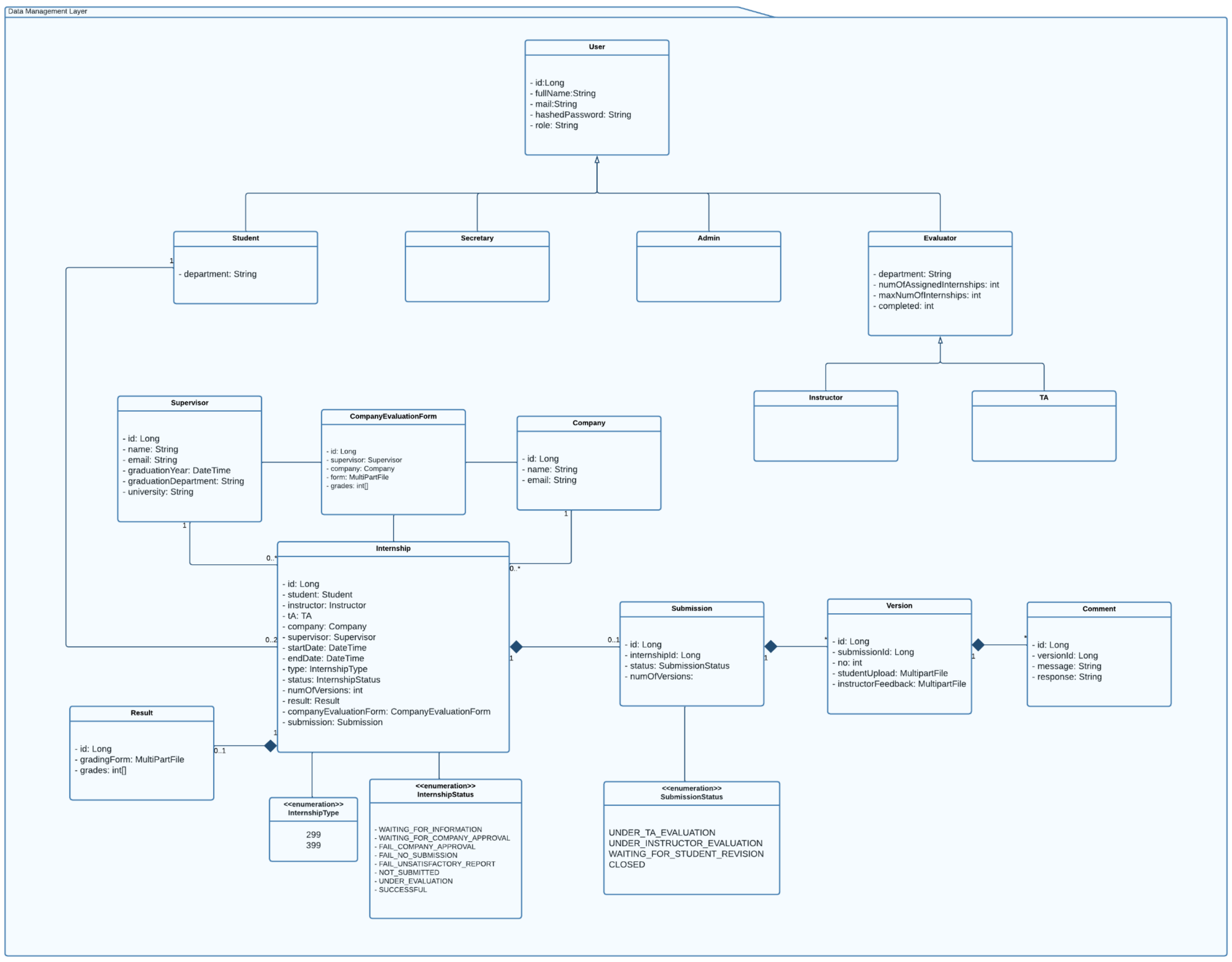


Figure 8: Data Management Layer Diagram

The associations between classes are achieved via foreign keys, which makes the size of JSON messages considerably smaller. For example, if the "Internship" class contained an instance of the "Submission" class, the JSON message would be nested and contain the submission file along with all the revisions, comments, etc. Instead, the id of the internship is added into the Submission class, as a foreign key. If a submission for a particular internship is desired, a search in the database according to the internship id needs to be performed.

3.4. Design Patterns

3.4.1. Facade

The first use of the Facade design pattern[1] in the backend of the application is to provide an abstraction to the database access. For this purpose, "Repository" interfaces that extend the "JPARepository"[2] interface provided by the Spring Boot are used.

Another appearance of the Facade design pattern is between controllers and the data access layer (repositories). Instead of performing boundary checks and applying business logic in the controller layer, a service layer for each controller is provided.

3.4.2. Strategy

In the application, a secretary should perform matching of the internships to the instructors. For now, our system offers two ways to achieve so; internships may be distributed equally or proportionally with the max number of internships that can be assignable to an instructor. Those different operations are implemented as strategies, namely EqualMatcher and RatioMatcher which inherit a common abstract base class named IMatcher. Implementing those algorithms as strategies allows the behavior of matching operation to change at runtime, depending on the input provided by the secretary (Secretary will be able to choose between equal and ratio on the frontend).

3.4.3. Factory

Above mentioned strategies are chosen with respect to the input provided by the secretary at runtime. Even though for now the system provides two matching algorithms, in the future there can be more. The client of the strategies should be unaware of the matching algorithm that is being used. Therefore, the determination of a matching algorithm that will be used for matching internships to the instructors is encapsulated in the MatcherFactory class, which provides a create() method that takes a MatchType as parameter and returns a IMatcher.

3.4.4 Template Method

Matching algorithms exhibit a very similar structure. They first perform a boundary check to determine whether the matching can be performed (number of internships should be less or equal than the total number of internships that can be accepted by the instructors). Then the algorithms perform matching, and lastly, they write the resultant assignments to the database. Those three steps and the order of the steps are encapsulated inside a template method match(), which is a method of IMatcher abstract class. Now the matching strategies only need to override performMatching() method, which actually performs the distribution of internships to instructors. The implementation for checkBoundary() and updateDatabase() are provided by the abstract base class IMatcher. In this way, whenever a new matching algorithm is going to be added, the new matching strategy class will not have to implement checkBoundary() and updateDatabase() methods again, it will simply need to override performMatching() method and implement its own matching algorithm. The matching system now is open to extension but close to modification.

3.4.5. Observer

To implement notification features in the application, Observer pattern[3] will be used. The students will be notified when an instructor requests a revision or finalizes the submission, and the instructors will be notified when students make a submission or upload a revised version. In this case, both Student and Instructor are going to be Subject and Observer at the same time.

4. IMPROVEMENT SUMMARY

According to the feedback, some alterations on both the textual descriptions and diagrams are performed. Instead of specifying top three design goals, top two design goals are selected. Some general statements are removed and application domain specific features are highlighted. The explanations on design goals are elaborated. Subsystem decomposition diagram is recreated. Some estimations about the system are presented in hardware software mapping part (2.2). Notation of the final object design is converted to UML standard. User interface diagram is revisited and the return types of the methods are changed. Links are provided for the original images. Design patterns section is enhanced, and some references are added accordingly.

5. REFERENCES

[1] E. Freeman et al., *Head First Design Patterns.* 2011, p. 235.

[2] "Interface jparepository," JpaRepository (Spring Data JPA Parent 3.0.0 API), 18-Nov-2022. [Online]. Available: https://docs.spring.io/spring-data/data-jpa/docs/current/api/org/springframework/data/jpa/reposit ory/JpaRepository.html. [Accessed: 28-Nov-2022].

[3] E. Freeman et al., *Head First Design Patterns.* 2011, p. 37.