Bilkent University

Department of Computer Engineering

Internship Report Management System

Project short-name: Bilport

Design Report

Arda İynem, Atika Zeynep Evmez, Ege Çenberci, Yağız Özkarahan, Zeynep Naz Sevim

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

Design report
May 5, 2023

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Object-Oriented Software Engineering course CS319.

1. Introduction

1.1. Purpose of The System

Bilport is a web application that aims to automate the internship report management process for Bilkent University's Computer Engineering Department. The current system used to evaluate the student's internship reports could be more convenient and efficient. Currently, the report evaluation process leaves a lot of meaningless file-sorting work to the department secretary. Moreover, the department secretary has to be the middleman while delivering the evaluators' feedback to the students, as the process' different parts are handled on various services and web applications. Bilport aims to provide a portal to manage all file and feedback exchanges through a single location. Bilport will relieve the department secretary of its needlessly time-consuming tasks by automating file handling and providing a direct feedback channel between students and evaluators.

1.2. Design Goals

1.2.1. Usability

Usability plays a key role as this project’s main motivation, because the need for this new system arises from the needless complexity of the previously existing internship report management system. The project aims to reduce the complexity of the report management process as much as possible, and make it easy and straightforward to use. Therefore, usability and general ease of use is a key design goal for our system. For this goal, the key aspect to consider is to design a user interface that is easy, straightforward and self-evident to use without the need of any guide or help, meanwhile covering all the functionalities of the system. With this in mind, the user interface will be designed with components such as sidebars, tabs, collapsible panels etc. to avoid an overly cluttered page. Of course while aiming to avoid overly cluttered pages, it is also important to avoid dividing the functionalities between different components in an unnecessary way, which would make the layout needlessly complicated and make the user get lost between all the menus. For this, the most commonly used key functionalities will be easily and quickly available to find in the website. A key functionality should be ideally reachable in at most 3 mouse clicks.

1.2.2. Maintainability

The internship report management system that is designed through this project is aimed to be used for many semesters in the foreseeable future. This means that there might be (and most likely will be) changes in the internship evaluation process of CS299 and CS399 courses in the future. Therefore, the developed system must be able to accommodate these changes. The system must be designed in a way that is easily adaptable and maintainable for any modifications, additions, deletions, bug fixes and performance improvements that may be implemented in the future, without needing an entire overhaul or rewrite of the source code. Also the possibility of the internship management system to be extended to other engineering departments in the future should be taken into consideration.

Making the system overly dependent on the CS department must be avoided for this reason. The modularity, adaptability and reusability advantages gained by React and Spring technologies will be used for the goal of maintainability through the project. Specifications of these technologies are further explained in the report.

2. High-level software architecture

2.1. Subsystem Decomposition

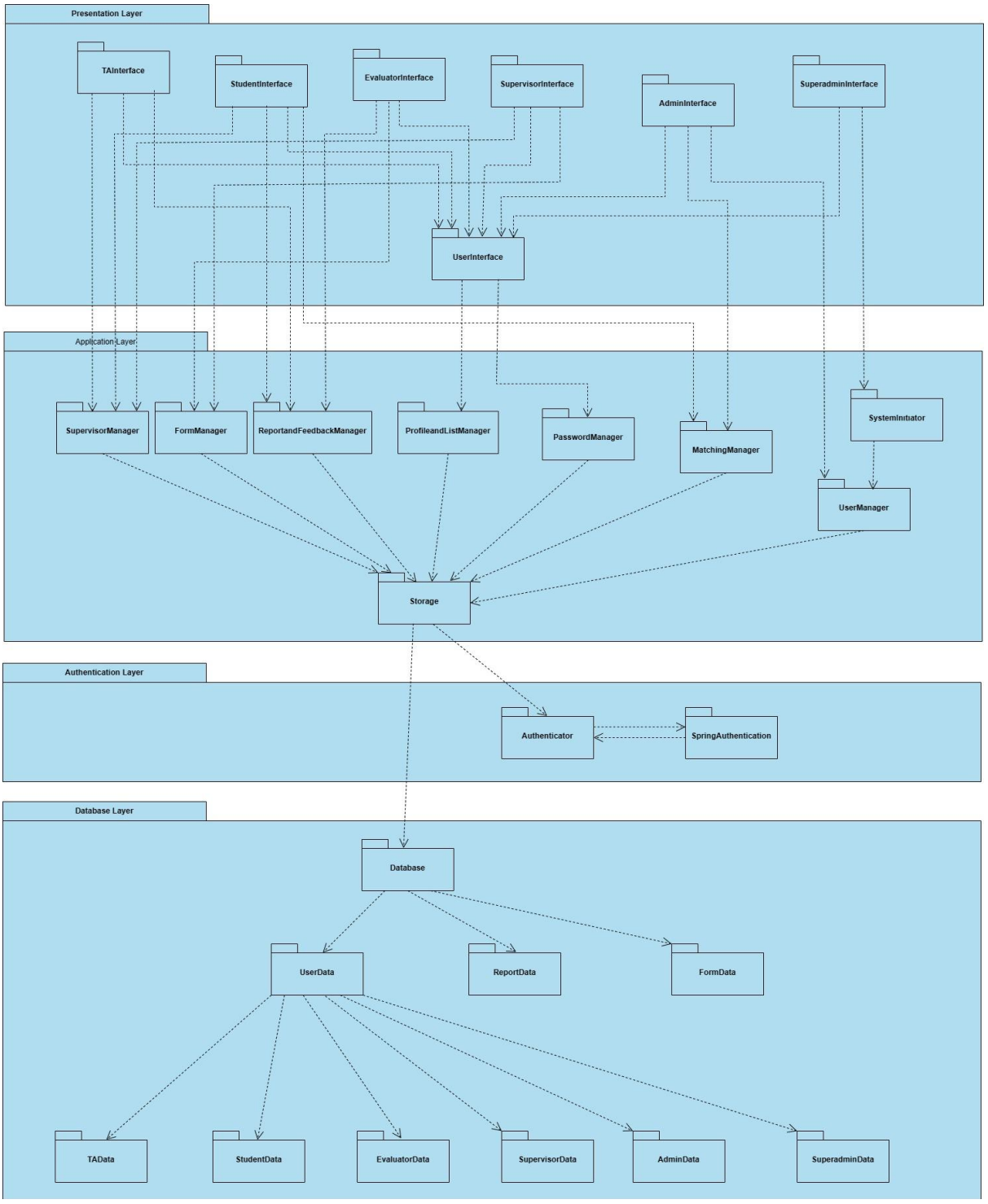

Figure 1: System Decomposition Diagram
(https://drive.google.com/file/d/1BwxlEUQR2aHJe5ErZ_UpaTselj5gALnU/view?usp=sharing)

2.1.1 Presentation Layer

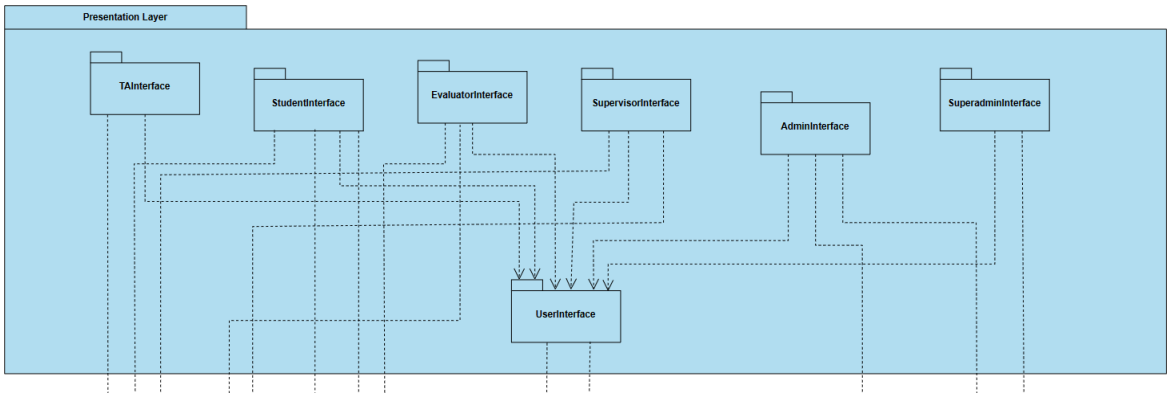

Figure 2: Presentation Layer Diagram

- This layer indicates the user interfaces of the application. End users can only interact with this part of the application. This layer includes boundary objects.
- For the dependencies that each user has (such as the PasswordManager package), all of the user interfaces are generalized in the UserInterface package.
- Each package is unique to the assigned user type.

2.1.2 Application Layer

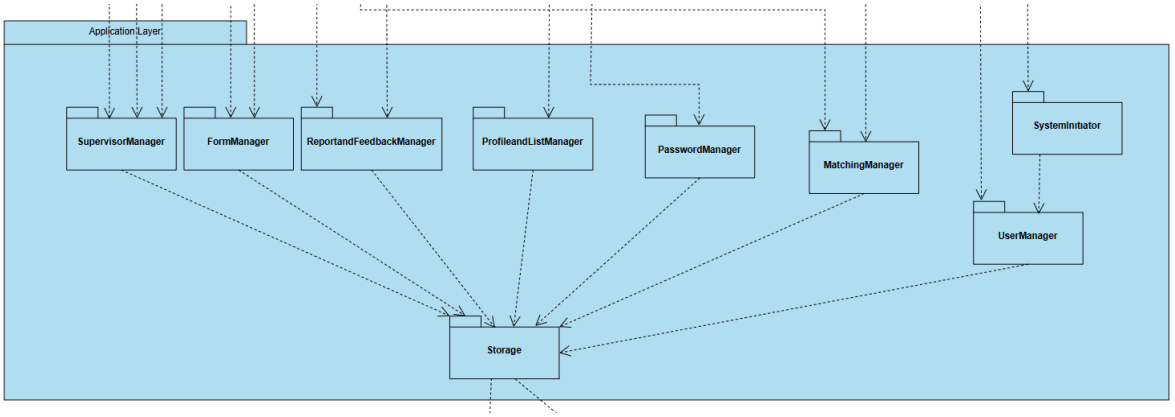

Figure 3: Application Layer Diagram

- This layer includes controller objects. This layer is mostly about the functionalities of the application.
- The objects in this layer can get or set data (information from entity objects) or manipulate this data. Also they can retrieve new results via related computations on this data.
- **SupervisorManager:** This package is about creating a supervisor user or retrieving her/his data. Students' TAs' and supervisors' interfaces depend on this package. TAs' interfaces also depend because as we explained in the analysis report, TA has to check the signature of the supervisor to make sure that the signature is the same with the signature in the internship acceptance letter which is also signed by the supervisor
- **FormManager:** This package is about storing data which is retrieved by the "Confidential Summer Training Evaluation Form" which is filled by the supervisor online. Also, it should retrieve this data when the evaluator needs it.

- **ReportandFeedbackManagement:** All functionalities about reports and feedback.
- **ProfileandListManager:** Functionalities about viewing a user profile or a user list.
- **PasswordManager:** Responsible for password change functionality.
- **MatchingManager:** Responsible for functionalities about matching students and evaluators, or retrieving this data if it exists.
- **UserManager:** Functionalities about users' accounts, such as creation of a new student user or deleting an evaluator's user account, and etc.
- **SystemInitiator:** Functionalities about composing the system from zero. It has more functionalities than the UserManager package and also users who can reach SystemInıtıator are different from the users who can reach only UserManager.

2.1.3 Authentication Layer

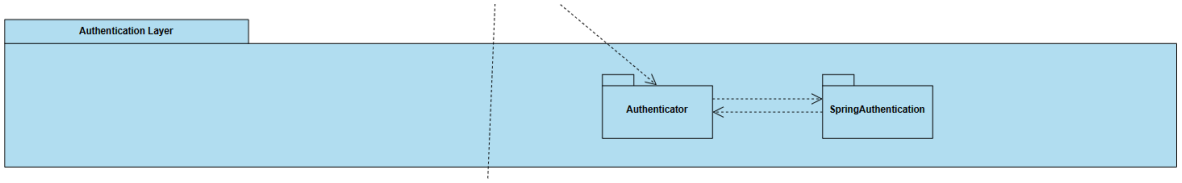

Figure 4: Authentication Layer Diagram

- This layer includes controller objects about authentication functionalities.

2.1.4 Database Layer

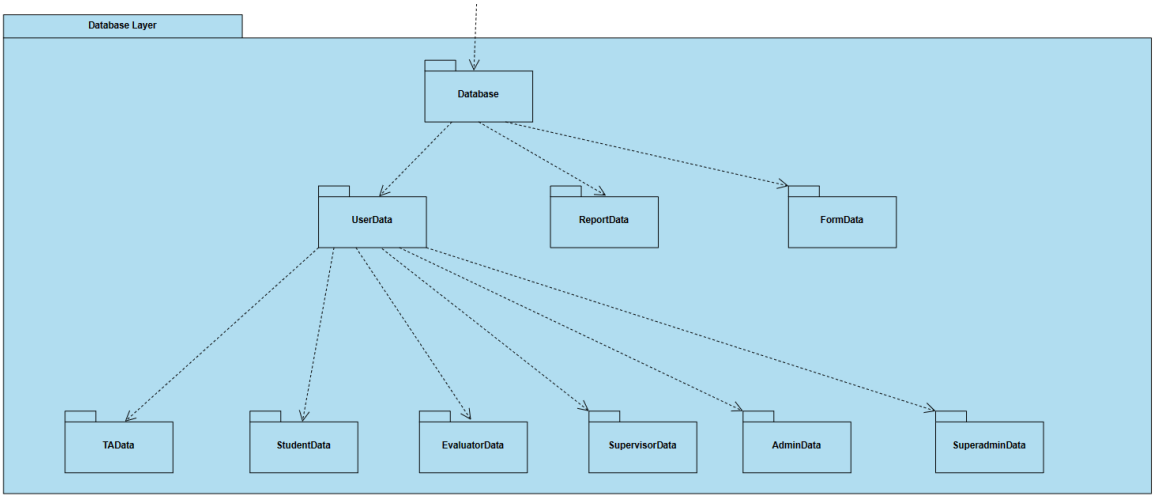

Figure 5: Database Layer Diagram

- This layer includes a database system. Database system is an external system which includes entity objects.

2.2. Hardware/Software Mapping

For the backend of the application, Java version 17 will be used as the main programming language. To implement extended back-end logic such as database manipulation, frontend communication and security/authorization features, Spring Boot version 3.0.6 will be used as the framework. The general structure of the backend will be

designed and organized in accordance to Spring Boot interfaces and features. We saw Spring Boot as a fitting choice to use as our framework mainly because:

- The framework's advantages in eliminating boilerplate code with its many annotations, which increases time-efficiency and helps to produce a cleaner code
- The framework provides a rich variety of plugins which greatly help to handle database management and security functionalities, which could be difficult and time consuming otherwise.

As for the database system of the application, we will use MongoDB. MongoDB employs a NoSQL database, which is useful for our application as it (especially with its Spring Boot plugins) greatly reduces the complex details of database design. Throughout this project, the database will be deployed on Amazon Web Services (AWS) with MongoDB's free tier. The free tier provides 512 MB memory and shared RAM. That much will be sufficient for our application, but the database can be later deployed on a priced tier, if the need for a larger memory or better performance arises in the future.

For the frontend of the application, JavaScript will be used, mainly with the React framework. The React was chosen as the main framework because:

- It allows us to write the frontend code in a much more organized way with React's component-based structure, which greatly reduces reused code, incorporates the HTML part directly into the JavaScript logic and allows extended modularity compared to standalone JavaScript.
- Its component-based structure mimics object orientation, which is a good addition in the context of our course.

To create the user interface, HTML/CSS with Bootstrap library will be used.

Since the project is exclusively designed as a web application, no detailed hardware components are needed. The app is expected to run on any hardware that can support modern (i.e. released after at least 2010 and forwards) operating systems and browsers. No functionality related compatibility issues are expected with such systems, and usability related issues will be minimized (for example, a very old browser version which does not support background gradients in the webpage will just display a chosen default color instead of a white page).

2.3. Persistent Data Management

To persist the data, MongoDB will be used. MongoDB is a document-oriented database, which means data is stored in a JSON-like document called BSON document. This enables a faster and more flexible data storage environment for complex and user-specific content such as reports, feedbacks, evaluation forms. MongoDB is a scalable database and this can help to improve the performance of the system in case other departments start to use the application. Additionally, MongoDB is a cloud-based database and this offers a convenient set up and straightforward database management for us, the developers. In our internship management system, MongoDB will be used to store all types of users (administrator, evaluator, student, supervisor, super admin, TA), reports uploaded by the students, feedback given by the TAs and evaluators and supervisor evaluation forms.

2.4. Access Control and Security

Spring Security is a framework for Java including pre-built mechanisms for authentication and authorization process while providing protection against common attacks.

Spring Security architecture basically consists of a chain of servlet filters that filters the HTTP requests. The filter chain provides the authentication and authorization functionalities. The built-in filters can be replaced and customized with user-defined filters then can be placed in the desired position in the filter chain.

For the Bilport Internship Report Management Application, a custom filter is implemented for the login process which checks the header of the request. The filter chain continues if the HTTP request satisfies the conditions of this custom filter which will be mentioned later in detail. Only after passing all the filters in the chain, the HTTP request can access the control layer of the application.

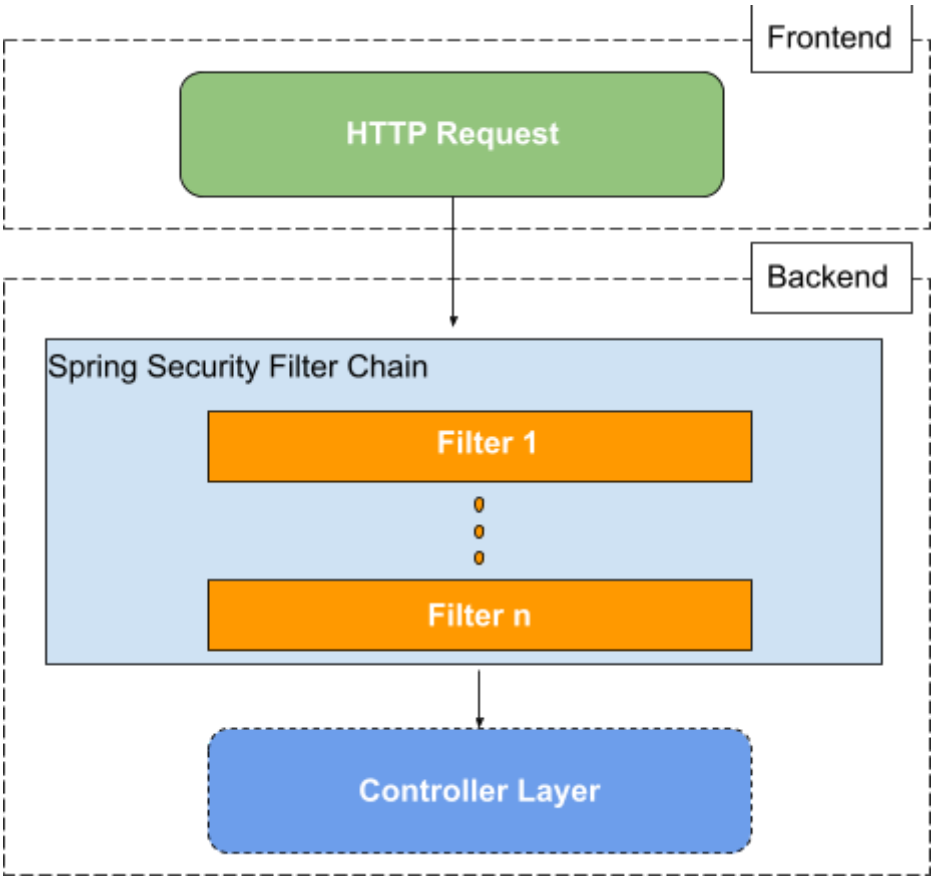

Figure 6: Bilport's Custom Layer

Besides the Spring Security, authentication and authorization process involves another mechanism for the stateless token based architecture. Namely, JWT (JSON Web Token) is utilized which encrypts the username and expiration date of the token with a specific key set in the backend and HS512 crypting algorithm.

The custom filter implemented determines whether the user tries to log in or tries to access an endpoint depending on the request's "Authentication" header. This filter accepts only either request with Authentication header with a value starting with Basic prefix followed

by Base64 encoded valid login data or requests with Authentication header with a value starting with Bearer prefix followed by a valid JWT token data.

If user is not logged in -no JWT refresh token cookie exists in browser cookies- and submits the login form, username and password is encoded with Base64 encoding then an HTTP request is sent to backend with an “Authentication” header which has the encoded username and password data with a “Basic” keyword in front of the data. The request passes through the filter chain if the login data is valid and user exists in the database, then a refresh token (created token stored in the database) and access token is created where refresh token is sent as a cookie and access token is sent in the response body with the role of the user to the frontend.

If user is logged in before -has a refresh token cookie in the browser and has JWT access token in the request’s Authentication header- the access token is checked in the filter whether it is valid or not then an HTTP response with error is returned if it is not valid or user is authorized to reach called endpoint, otherwise the corresponding HTTP response with data is returned.

Refresh tokens are just generated UUIDs mapped to logged in users to check whether a user is logged in to the system. These tokens are created in the database and sent to the user's browser as a cookie whenever a user is logged in, and similarly they are deleted both from the browser’s cookies and database when the user signs out. Refresh tokens are used for creating new JWT access tokens as access tokens lifetime is very short compared to refresh tokens. Whenever a request is denied due to invalid access token, frontend sends another request to “refresh” API endpoint which gets refresh token from the cookies of request then if the refresh token is a valid token in the database returns a newly created access token for the user which will be used for subsequent accesses to API endpoints.

	Student	TA	Evaluator	Supervisor	Admin	Superadmin
Sign in	X	X	X	X	X	X
Sign out	X	X	X	X	X	X
Change password	X	X	X	X	X	X
Assign students to evaluators					X	X
Import Student, Evaluator and TA Data						X
Export Student, Evaluator and TA Data						X
Initialize the System						X
Enroll/Delete Students					X	
Enroll/Delete TAs					X	
Enroll/Delete Evaluators					X	

Enroll/Delete Admins					X	X
Pre-Evaluate Report		X				
View Report	X	X	X			
Evaluate Report			X			
Give Feedback			X			
Submit Report	X					
Update Report	X					
Enter Supervisor Data	X					
Submit Training Evaluation Form				X		
View List of all Students		X	X		X	
View Students' Statuses		X	X		X	
View Assigned Student List			X			
View Student's Profile	X	X	X		X	
View Student's Report List	X	X	X		X	
View Student's Information	X	X	X		X	
View Student's Status	X	X	X		X	
Search Student		X	X		X	
View List of TAs		X	X		X	
View List of Evaluators		X	X		X	
View TA Profile		X	X		X	
View Evaluator Profile		X	X		X	
View Student List of TA/Evaluator		X	X		X	

Table 1: Users and their accessible methods

2.5. Boundary Conditions

2.5.1 Initialization

The system will be initialized by the super-admin user. For its initialization, several crucial information containing documents are required. The system will initialize all of its different users. Therefore, the users' data must be fed into it at initialization time. Microsoft Excel files containing all of the students', TAs', and evaluators' information must be prepared beforehand and given to the system. The Microsoft Excel document must be formatted such that the first two columns contain the name and surname of the user. If a user has multiple

names, they must be placed into the first column. The third and final column must contain the e-mail addresses of the users. The system will create the profiles of its users from the given Excel files, initialize their profiles, and send their automatically generated passwords via e-mail to each user. As the super-admin user will oversee the initialization process, they will handle any errors and exceptions if the operation fails. Suppose it is later discovered that a single user's initialization was unsuccessful. In that case, the super-admin will create a profile and assign it proper authorization.

2.5.2 Termination

The super-admin user will terminate the system when the semester or the class is concluded. During termination, the semester's relevant data will be exported to files for documentation. Once the super-admin confirms the exported data is not corrupted, the termination operation will delete all the users except the super-admin and empty the database.

2.5.3 Failure

Bilport uses Spring to authenticate its users and transfer information between layers and MongoDB to store data; hence is a Spring and Mongo-based system. Spring and MongoDB have built-in exception-throwing, error message-displaying, and logging capabilities. In the case of a failure that the system can not recover from, or when the error persists after a system restart, the super-admin and the developers will be notified of the problem to troubleshoot.

3. Low-Level Design

3.1. Object Design Trade-Offs

- Maintainability & Performance

Bilport prioritizes maintainability over performance, as can be inferred from the use of Spring to manage most of the application's functionality. Using Spring tools improves the maintainability of the project in several ways. Spring provides a flexible architecture that allows easy addition or removal of functionalities. Hence, the system can be easily extended or modified without having to rewrite the entire application. As Spring provides several features that help to reduce coupling between different components in the system, it is easier to test and maintain the application. Moreover, Spring provides various tools for monitoring and debugging the application, which allows for identifying and fixing issues quickly. However, due to its heavyweight nature and the additional processing overhead that comes with its features and abstractions, and our team not being experienced Spring developers to optimize everything to its full extent, some sacrifices are made in the performance department.

- Usability & Functionality

At its core, Bilport is a document-handling system. Therefore, usability has higher precedence than functionality. The web application must not feel too crowded with rarely utilized features. The earlier ideas presented by the team were scratched to achieve simplicity, such as getting student, evaluator, and TA information from the SRS system. Such

functionalities require Bilport to be able to communicate with a completely different system, which complicates things and wastes more time and effort than the extra functionality would save.

3.2. Final Object Design

The current object design of Bilport can be seen in Figure 7. It is important to note that not all getter and setter methods are shown in the diagram for simplicity's sake. It should be assumed that all attributes have getter and setter methods.

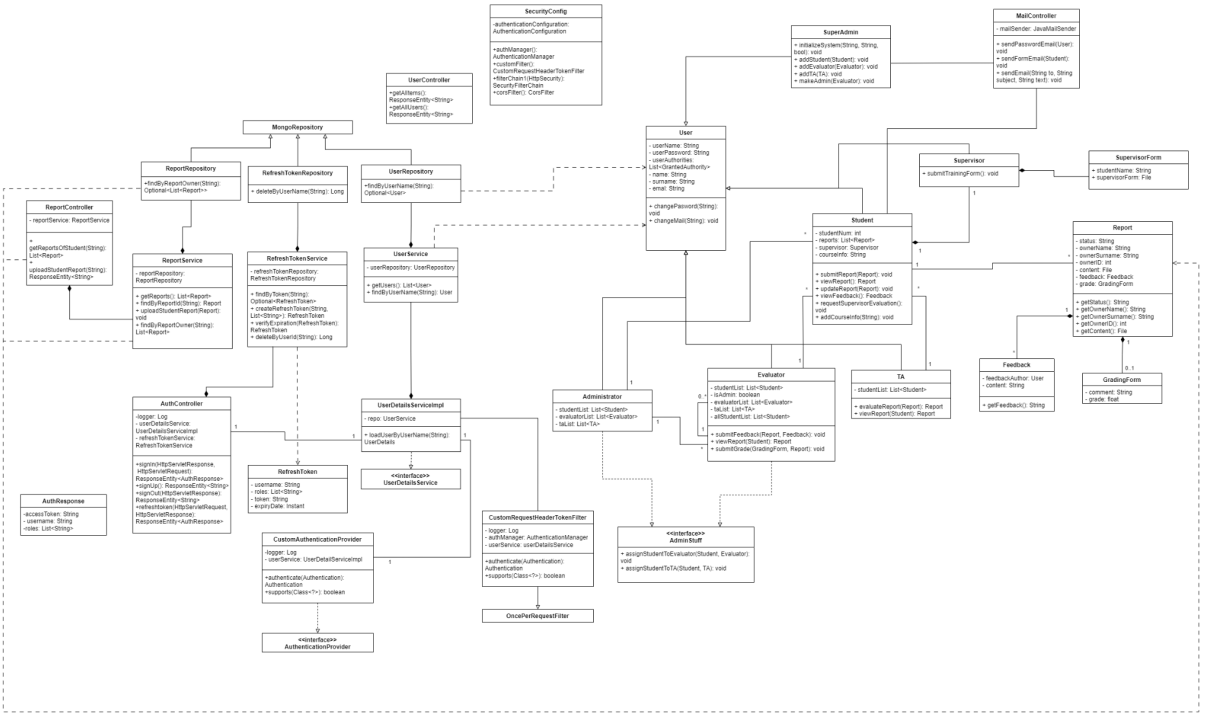

Figure 7: Object Design
(https://drive.google.com/file/d/1GyDaDeZeaE4k5zdCF9tD6qIQVhsqk4gw/view?usp=sharing)

Several design patterns which are part of Spring's core architecture are applied with the use of the Spring Framework. One of these key design patterns is the Dependency Injection (DI) pattern, a form of Inversion of Control (IoC). DI allows objects to be created and wired together at runtime rather than being hard-coded into the application code. This makes it easier to manage dependencies between objects and allows for greater flexibility and modularity in the design.

Moreover, since MongoDB was used as the project's database, Spring Data MongoDB, a Spring Framework module that provides integration with MongoDB, was used to communicate between the application and the database. One of the key design patterns used by Spring Data MongoDB's methods is the Template design pattern. The Template design pattern is used as Spring Data MongoDB provides a standardized way to execute common MongoDB operations such as querying, updating, and deleting documents.

Therefore, Dependency Injection and Template design patterns have been used for this project.

3.3. Packages

Spring Framework: A Java framework used to build web applications.

Spring Boot: A framework built on top of the Spring Framework that simplifies the process of building and deploying Spring-based applications.

Spring Security: Used in the authentication and authorization processes of the system's users.

Spring Data MongoDB: Used in communication between the database and the backend.

MongoDB: Used as the project's database.

React: Used in the development of the frontend of the web application.

3.4. Class Interfaces

3.4.1 Entity Classes

User

A parent class for all different users of the system.

Attributes:

- String userName: user's username
- String userPassword: user's password
- List<GrantedAuthority> userAuthorities: user's access authorities, separates student from evaluator etc.
- String name: user's name, may include middle names
- String surname : user's surname
- String email: user's email address

Methods:

- void changePassword(String): takes new password as input and changes the user's password.
- void changeMail(String): takes a new email address as input and changes the user's email address.

SuperAdmin

The class with the highest authority, tasked with initialization and shutdown of the system. Inherits the User class.

Methods:

- void initializeSystem(String, String, bool): takes two file names as input, one containing student information and one containing the evaluator information. The boolean represents the existence of TA information in the evaluator file. Reads the files accordingly to the parameters and initializes the system.
- void addStudent(Student): manually adds a Student object into the system.

- void addEvaluator(Evaluator): manually adds an Evaluator object into the system.
- void addTA(TA): manually adds a TA object into the system.
- void makeAdmin(Evaluator): manually grants an Evaluator object admin privileges.

Administrator

Administrator is a class with authority to assign students to TAs and evaluators. Inherits the User class and implements the AdminStuff interface.
Attributes:

- List<Student> studentList: list that holds all the students that take the course.
- List<Evaluator> evaluatorList: list that holds all the evaluators for the course.
- List<TA> taList: list that holds all the TAs for the course.

Methods:

- void assignStudentToEvaluator(Student, Evaluator): takes a student object and Evaluator object to assign the Student to the given Evaluator. Updates the studentList of Evaluator.
- void assignStudentToTA(Student, TA): takes a student object and TA object to assign the Student to the given TA. Updates the studentList of TA.

Evaluator

The Evaluator class represents the graders of the student reports. It inherits the User class and implements the AdminStuff interface.
Attributes:

- List<Student> studentList: list that holds the students that the evaluator is responsible for.
- boolean isAdmin: admin status of the evaluator object.
- List<Evaluator> evaluatorList: list that holds all the evaluators for the course.
- List<TA> taList: list that holds all the TAs for the course.
- List<Student> allStudentList: list that holds all the students of the system, empty if evaluator doesn't have admin privileges.

Method:

- void submitFeedback(Report, Feedback): method that submits Feedback object to the given Report object.
- Report viewReport(Student): returns the most recent Report of the Student.
- void assignStudentToEvaluator(Student, Evaluator): takes a student object and Evaluator object to assign the Student to the given Evaluator. Updates the studentList of Evaluator.
- void assignStudentToTA(Student, TA): takes a student object and TA object to assign the Student to the given TA. Updates the studentList of TA.
- void submitGrade(GradingForm, Report): updates the Report object's GradingForm to the one passed as an argument.

TA

The TA class is used to evaluate reports of students with less authority than the evaluators. It inherits the User class.
Attributes:

- List<Student> studentList: list that holds the student objects the TA is responsible for.

Methods:

- Report evaluateReport(Report): takes a report object to evaluate and returns the evaluated Report.
- Report viewReport(Student): returns the latest submitted Report of the Student.

Student

The Student class holds student specific information and methods, inherits the User class.

Attributes:

- int studentNum: student number
- List<Report> reports: Report list that holds the report iterations submitted by the student.
- Supervisor supervisor: The supervisor that was responsible for the student during their internship.
- String courseInfo: String to represent which courses the student is taking.

Methods:

- void submitReport(Report): takes a Report object and submits it to the system.
- Report viewReport(): returns the latest submitted Report for viewing.
- void updateReport(Report): takes Report object as argument and updates the latest iteration of the Report, updates reports.
- Feedback viewFeedback(): returns the feedback given to the latest Report.
- void requestSupervisorEvaluation(): utilizes the MailController to send an automated email to the student's supervisor.
- void addCourseInfo(String): takes String input and updates Student object's courseInfo.

Supervisor

The Supervisor class is a trivial class that is used to submit supervisor's evaluation of the student's internship.

Methods:

- void submitTrainingForm(): submits the student's training evaluation form to the system as a SupervisorForm object.

SupervisorForm

The SupervisorForm class is a container class for the supervisor evaluation form.

Attributes:

- String studentName: holds the name of the student that is being evaluated.
- File supervisorForm: holds the evaluation form of the student.

Report

The report class holds some crucial information about the report, and the report itself.

Attributes:

- String status: status description of the report. Waiting to be evaluated, waiting for an update etc.
- String ownerName: the student's name that wrote the report, for easier access.

- String ownerSurname: the student's surname that wrote the report, for easier access.
- int ownerID: the student's Bilkent ID number that wrote the report, for easier access.
- File content: the submitted report file.
- Feedback feedback: Feedback object given to the Report object.
- GradingForm grade: GradeForm object given to the Report object.

GradingForm

Gradingform class holds the information about the final grade assigned to a Report object.
Attributes:

- String comment: holds the final comment given to the report, if any.
- float grade: holds the final grade assigned to the Report object.

Feedback

The Feedback class contains the information about the feedback given to a Report object.
Attributes:

- User feedbackAuthor: the user that has given the feedback to the Report object.
- String content: used to store the contents of the feedback comment.

3.4.2 Control Classes

AuthController

AuthController class provides endpoints for user authentication, sign-up, sign-out, and refreshing access tokens when they expire.
Attributes:

- Log logger: provides logging functionality to the controller.
- UserDetailsServiceImpl userDetailsService: used to load user specific data.
- RefreshTokenService refreshTokenService: used to manage refresh tokens, which is then used to request new access tokens without re-authentication.

Methods:

- ResponseEntity<AuthResponse> signIn(HttpServletResponse, HttpServletRequest): handles HTTP GET requests to the "/auth/signin" endpoint.
- ResponseEntity<String> signUp(): handles HTTP POST requests to the "/auth/signup" endpoint. When the endpoint is hit, it returns an HTTP response indicating that the user has been registered.
- ResponseEntity<String> signOut(HttpServletResponse): handles HTTP GET requests to the "/auth/signout" endpoint. Deletes the current refresh token associated with the user.
- ResponseEntity<AuthResponse> refreshtoken(HttpServletRequest, HttpServletResponse): handles HTTP GET requests to the "/auth/refresh" endpoint. When the endpoint is hit, the method looks for a refresh token in the request cookies and attempts to verify it using the RefreshTokenService.

ReportController

ReportController class handles HTTP requests related to reports.

Attributes:

- ReportService reportService: used to perform operations on reports.

Methods:

- List<Report> getReportsOfStudent(String): returns the list of reports belonging to the student with the name passed as an argument.
- ResponseEntity<String> uploadStudentReport(String, Report)

UserController

UserController class handles HTTP GET requests to the endpoints “/items” and “/users”.

Methods:

- ResponseEntity<String> getAllItems(): This method handles HTTP GET requests to the “/items” endpoint and returns a ResponseEntity object that contains a String response and an HTTP status code.
- ResponseEntity<String> getAllUsers(): This method handles HTTP GET requests to the “/users” endpoint and returns a ResponseEntity object that contains a String response and an HTTP status code.

MailController

MailController class is used to send automated mails to the system’s users for different scenarios.

Attributes:

- JavaMailSender mailSender: JavaMailSender object to help the mail sending methods.

Methods:

- void sendPasswordEmail(User): sends password reset email to the email address of the given User object.
- void sendFormEmail(Student): sends internship evaluation form request mail to the given Student object’s supervisor’s email address.
- void sendEmail(String to, String subject, String text): used to send email to the given email address with the details specified in the arguments.

4. Glossary

- TA: Teacher’s Assistant
- Evaluator: Instructors from Bilkent CS Department that are responsible for grading in the internship courses
- Supervisor: The company employee that oversees the internship process of the student

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.