The University Team Project Platform

Software Design Specification

2021.05.06.

Introduction to Software Engineering 41

TEAM 14 (Hello Team Project)

Team Leader Jaehyun Ju
Team Member Min Jang
Team Member Heesoo Jung
Team Member Byeongsu Woo
Team Member Seongwook Lim

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Preface

This chapter contains the readership information, readership, scope, objective of this document and the document structure of this Software Design Document for Hello Team Project.

1.1. Readership

This Software Design Document is divided into 10 sections with various subsections. The structure of the Software Design Document can be found as listed below, in the Document Structure subsection of this SDD. In this document, Team 14 is the main reader. Additionally, professors, TAs, and team members in the Introduction to Software Engineering class can be the main readers.

1.2. Scope

This Design Specification is to be used by Software Engineering and Software Quality Engineering as a definition of the design to be used to implement the University online team project platform.

1.3. Objective

The primary purpose of this Software Design Document is to provide a description of the technical design aspects for our university online team project platform, Hello Team Project. This document describes the software architecture and software design decisions for the implementation of Hello Team Project. It also provides an architectural overview of the system to depict different aspects of the system. It further specifies the structure and design of some of the modules discussed in the SRS document and in addition, displays some of the use cases that have been transformed into sequential and activity diagrams, including the class diagrams which show how the programming team would implement the specific module. The intended audience of this document is, but not limited to, the stakeholders, developers, designers, and software testers of the Hello Team Project university online team project platform.

1.4. Document Structure

- **1. Preface**: this chapter describes readership, scope of this document, object of this

system, and structure of this document.

- **2. Introduction**: this chapter describes several tools used for this document, several diagrams used in this document and the references, and object of this project.
- **3. Overall System Architecture**: this chapter describes the overall architecture of the system using context diagram, sequence diagram, and use case diagram.
- **4. System Architecture - Frontend**: this chapter describes architecture of the frontend system using class diagram and sequence diagram.
- **5. System Architecture - Backend**: this chapter describes architecture of the backend system using class diagram and sequence diagram.
- **6. Protocol Design**: this chapter describes design of several protocols used for communication of client and server.
- **7. Database Design**: this chapter describes database design using several ER diagrams and SQL DDL.
- **8. Testing Plan**: this chapter describes the testing plan for our system.
- **9. Development Plan**: this chapter describes which tools to use to develop the system, constraints, assumption, and dependencies for developing this system.
- **10. Supporting Information**: this chapter describes the baseline of this document and history of this document.

2. Introduction

The project is to develop and design a web platform used for the introduction which makes students find team members easier, for the communication which makes students do a team project online easier, and for the rating which reduces free riders. The system should provide the information management place for individual students, information searching place for students who find team members, team making request function for students who want to make a team, team management place for professors who want to manage the teams in their lecture, communication place for professors and students to communicate with team members and professor during the project, and rating place for students who want to evaluate their team members after the team project. This design document presents the designs used or

intended to be used in implementing the project. The designs described, follow the requirements specified in the Software Requirements Specifications document prepared earlier for the project.

2.1. Objectives

In this chapter, we describe the various tools and diagrams which we have applied to this project in the design phase.

2.2. Applied Diagrams

2.2.1. UML

UML is an acronym that stands for Unified Modeling Language. Simply put, UML is a modern approach to modeling and documenting software. In fact, it's one of the most popular business process modeling techniques. UML is a common language for business analysts, software architects and developers used to describe, specify, design, and document existing or new business processes, structure and behavior of artifacts of software systems. UML can be applied to diverse application domains (e.g., banking, finance, internet, aerospace, healthcare, etc.) It can be used with all major object and component software development methods and for various implementation platforms (e.g., J2EE, .NET). It is based on diagrammatic representations of software components. As the old proverb says: "a picture is worth a thousand words". By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

2.2.2. Use case Diagram

A cornerstone part of the system is the functional requirements that the system fulfills. Use Case diagrams are used to analyze the system's high-level requirements. These requirements are expressed through different use cases. We notice three main components of this UML diagram: Functional requirements – represented as use cases; a verb describing an action. Actors – they interact with the system; an actor can be a human being, an organization or an internal or external application. Relationships between actors and use cases – represented using straight arrows.

2.2.3. Sequence Diagram

Sequence diagrams are probably the most important UML diagrams among not only the computer science community but also as design-level models for business application development. Lately, they have become popular in depicting business processes, because of their visually self-explanatory nature. As the name suggests, sequence diagrams describe the sequence of messages and interactions that happen between actors and objects. Actors or objects can be active only when needed or when another object wants to communicate with them. All communication is represented in a chronological manner.

2.2.4. Class Diagram

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. Since classes are the building block of objects, class diagrams are the building blocks of UML. The various components in a class diagram can represent the classes that will actually be programmed, the main objects, or the interactions between classes and objects. The class shape itself consists of a rectangle with three rows. The top row contains the name of the class, the middle row contains the attributes of the class, and the bottom section expresses the methods or operations that the class may use. Classes and subclasses are grouped together to show the static relationship between each object.

2.2.5. Context Diagram

The system context diagram (also known as a level 0 DFD) is the highest level in a data flow diagram and contains only one process, representing the entire system, which establishes the context and boundaries of the system to be modeled. It identifies the flows of information between the system and external entities (i.e. actors). A context diagram is typically included in a requirements document. It must be read by all project stakeholders and thus should be written in plain language, so the stakeholders can understand items. The objective of the system context diagram is to focus attention on external factors and events that should be considered in developing a complete set of systems requirements and constraints. A system context diagram is often used early in a project to determine the scope under investigation. Thus, within the document. A system context diagram represents all external entities that may

interact with a system. The entire software system is shown as a single process. Such a diagram pictures the system at the center, with no details of its interior structure, surrounded by all its External entities, interacting systems, and environments.

2.2.6. Entity Relationship Diagram

An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties. By defining the entities, their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases. Entity Relationship diagrams are used to sketch out the design of a database.

2.3. Applied Tools

2.3.1. Microsoft PowerPoint

This is a tool that supports drawing text and figures. It is convenient to draw diagrams using various shapes. In addition, it is easy to edit because it works as full word-processor formatting (which is a tool for working with documents), graphic shapes with attached text for drawing diagrams and tables.

2.3.2. Lucid

This is a web-based drawing diagram tool that used to draw the class diagram in this document.

2.4. Project Scope

The Hello Team Project online team project platform is established to reduce the inconvenient situations when team projects are done online such as finding team member problems, team management problems, communication problems, and free riding team member problems. The system is based on a relational database. We will have a database server handling the information of students, professors, and courses in the university. Furthermore, it will handle the team information in the courses. Above all, we hope to provide a comfortable user experience with the helpful online team project platform that provides the finding team member space, the communication for team project space, and the

commenting team member space.

2.5. References

The user of this SDD may need the following documents for reference:

- Team 1, 2020 Spring. Software Design Document, SKKU.
- Appleton, Brad. A Software Design Specification Template. N.d.
- P. Burke, K. Martin, D. Longtin. Software Design Specification for a One Runway Airport/Air Traffic Controller Simulation <https://www.academia.edu/29811493/SOFTWARE_DESIGN_SPECIFICATION_FOR_A_ONE_RUNWAY_AIRPORT_AIR_TRAFFIC_CONTROLLER_SIMULATION>.

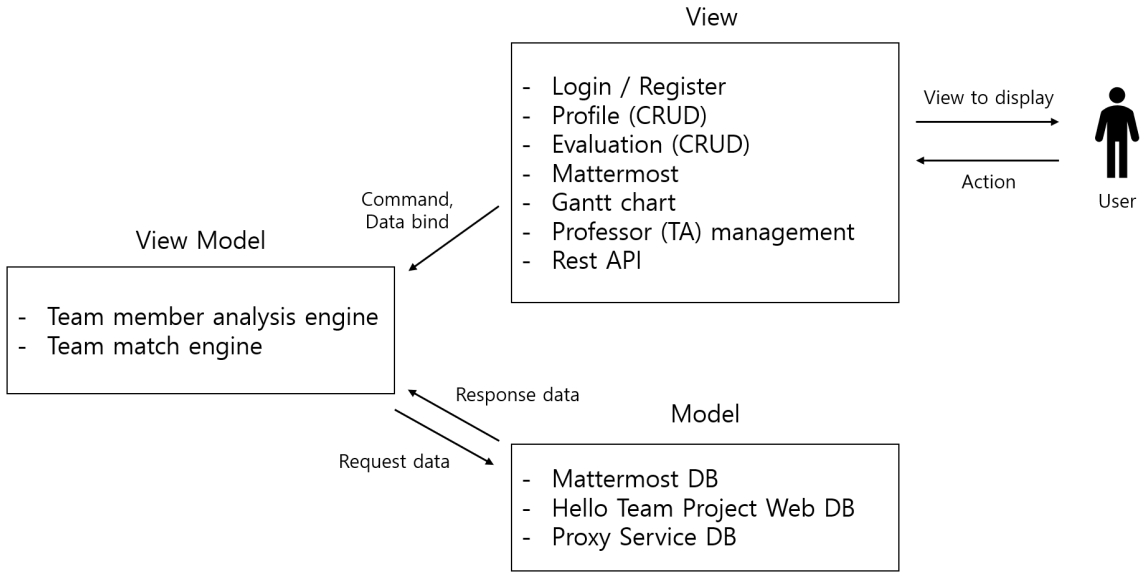
3. System Architecture – Overall

3.1. Objectives

Here in this chapter, we describe and show the organization of the system, ranging from the frontend design to the backend design of the application for the project.

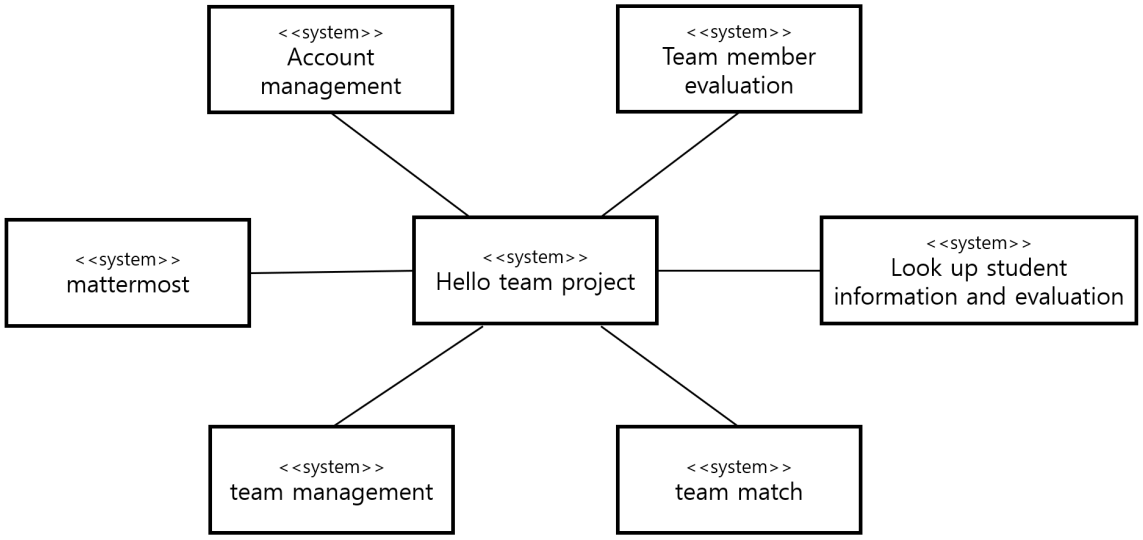
3.2. System Organization

This service is designed by applying the client - server model, the frontend web page is responsible for all interactions with users, and the front-end system communicates with the back-end system through HTTP communication based on JSON. The back-end system distributes design specification requests from the front-end to the controller, obtains the required object information from the database, processes it from the database, and delivers it to the JSON format. The back-end receives student review from the review writing requests every time. If a student wants to block it, the student can do it. When the professor or TA wants to start a team project and make a team for the students who don't have their team, the fair team matching system will use a special algorithm for fast matching. After team making is confirmed, the team channel will be made at mattermost server. Students can do their team project in that space.



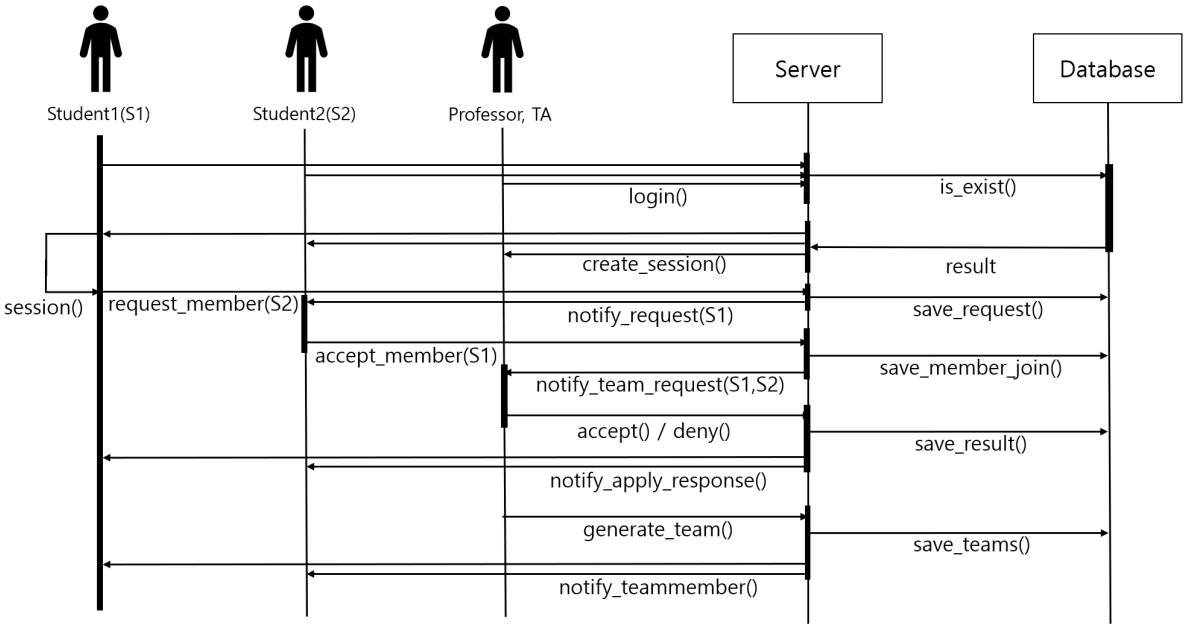
[Figure 1] Overall system architecture

3.2.1. Context Diagram



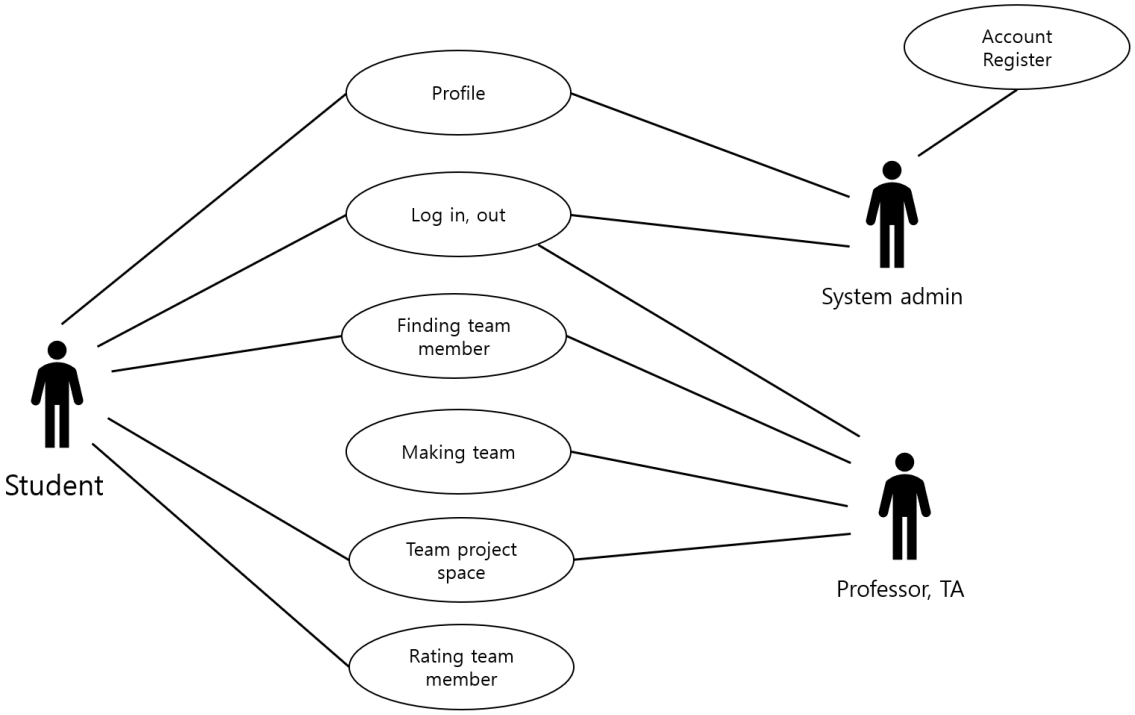
[Figure 2] Overall context diagram

3.2.2. Sequence Diagram



[Figure 3] Overall sequence diagram

3.2.3. Use Case Diagram



[Figure 4] Use case diagram

4. System Architecture – Frontend

4.1. Objectives

This chapter describes the structure, attributes and function of the frontend system and describes the relation of each component.

4.2. Subcomponents

4.2.1. Log-In

The Log-In class deals with user account information, which is obtained by a school database. After student login, the server can give the user profile page.

4.2.1.1. **Attributes**

These are the attributes that “student_from_school” object has.

- **Account_id**: id of the user (email address)
- **Password**: password of the user
- **Email**: email address of the user
- **Student Number:** student number of the user
- **Name**: name of the user

And these are the attributes that “Profile” object has.

- **Student Number:** student number of the user
- **Comment**: the table of the comments
- **Major**: major of the student
- **Rating**: rating of the student
- **project**: the list of finished student’s project
- **Available language**: the language that student can speak
- **User_class_list**: the list of class that the user attends

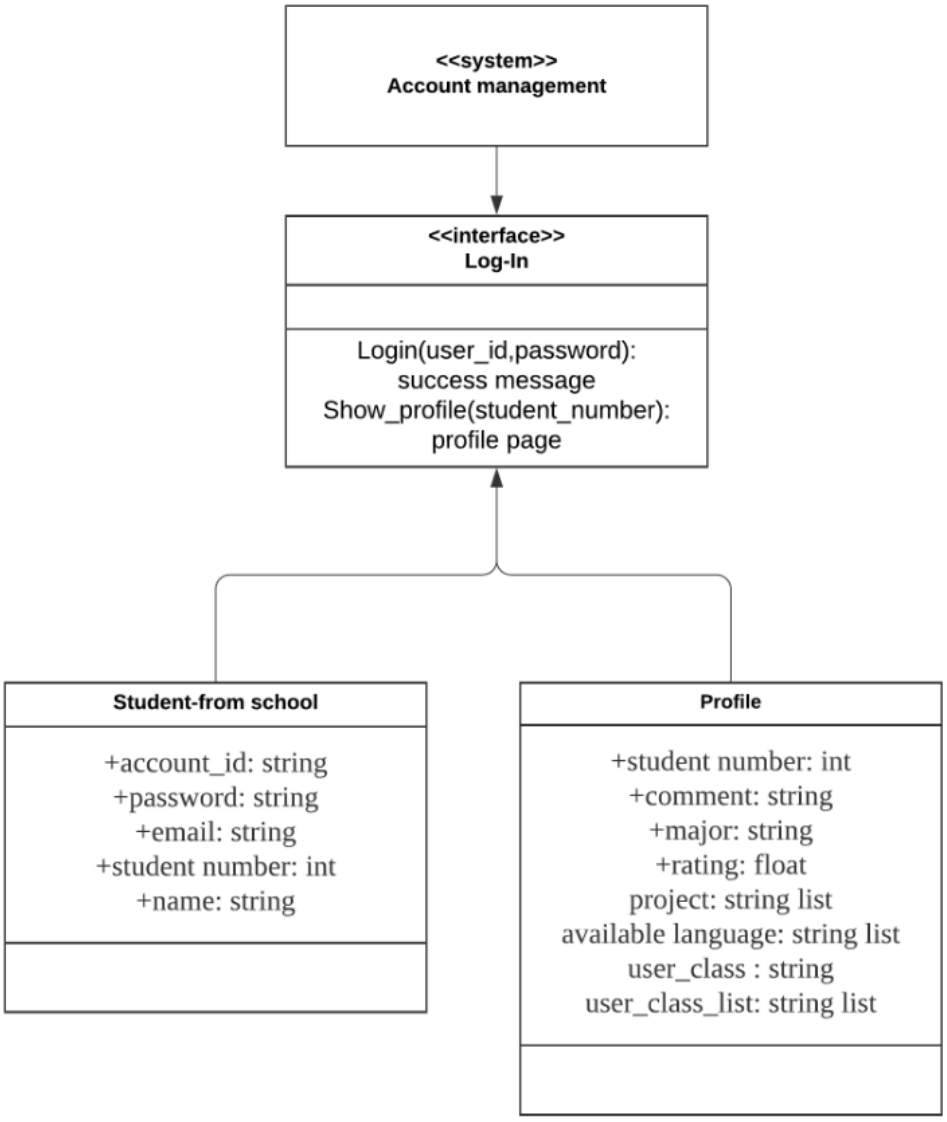
- **User_class**: the class that is chosen by user, for searching team member

4.2.1.2. **Methods**

These are the methods that the interface has.

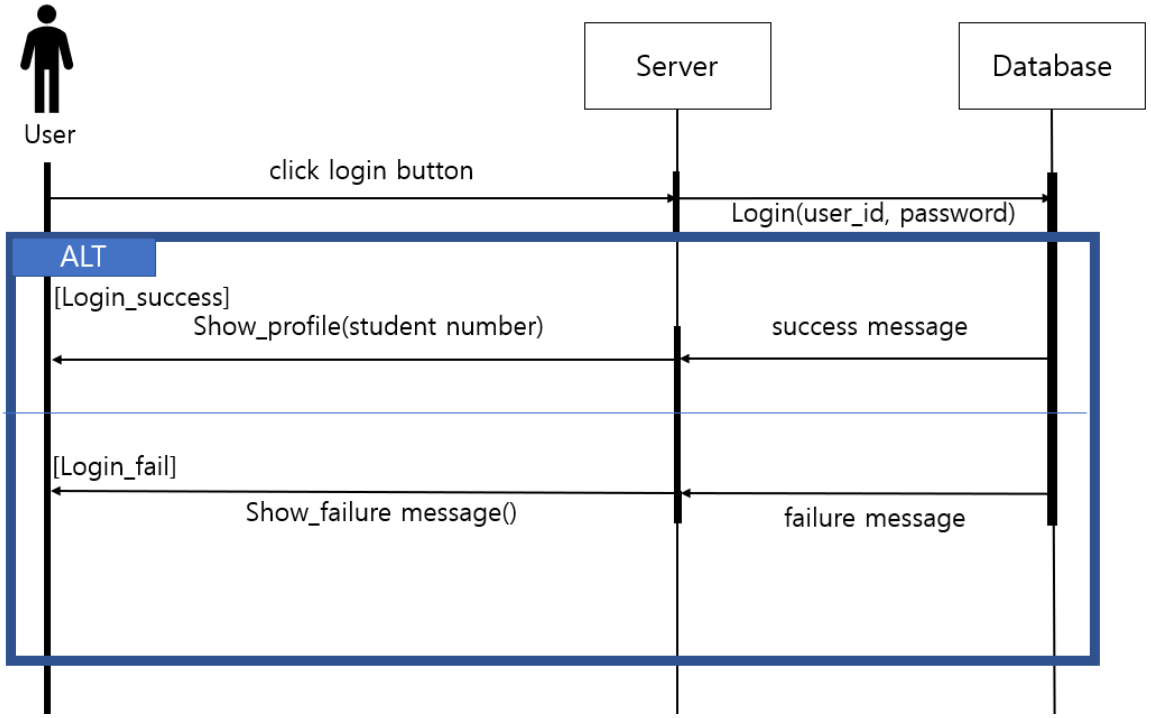
- Login(user_id, password) - try to login with input id and password.
- Show_profile(student_number) - get the profile page of student

4.2.1.3. **Class Diagram**



[Figure 5] Class diagram – Login

4.2.1.4. **Sequence Diagram**



[Figure 6] Sequence diagram – Login

4.2.2. Lookup student info

This takes part of showing the list of students, and showing the specific information about one particular student.

4.2.2.1. **Attribute**

These are the attributes that “profile” object has.

- **Student Number:** student number of the user
- **Comment**: the table of the comments
- **Major**: major of the student
- **Rating**: rating of the student
- **project**: project that the student has done
- **Available language**: the language that student can speak

- **User_class_list**: the list of class that the user attends
- **User_class**: the class that is chosen by user, for searching team member

And these are attributes that “comment & rating” object has.

- **Rating:** rating that is given with a particular comment. not overall rating.
- **Comment**: comment string
- **Comment_ID**: id of comment
- **student num_to**: number of the student that is evaluated
- **student num_from**: number of the student that evaluates
- **hide/open**: the flag that the comment is hidden
- **report**: the flag that the comment is reported

And these are attributes that “Course” object has.

- **class:** class code
- **professor name**: name of the professor
- **classroom**: name of the classroom
- **time**: time of the class
- **student_number:** the list of student id of the students taking the course
- **group made:** the list of flag whether each student has team or not
- **group list:** the list of group in the course

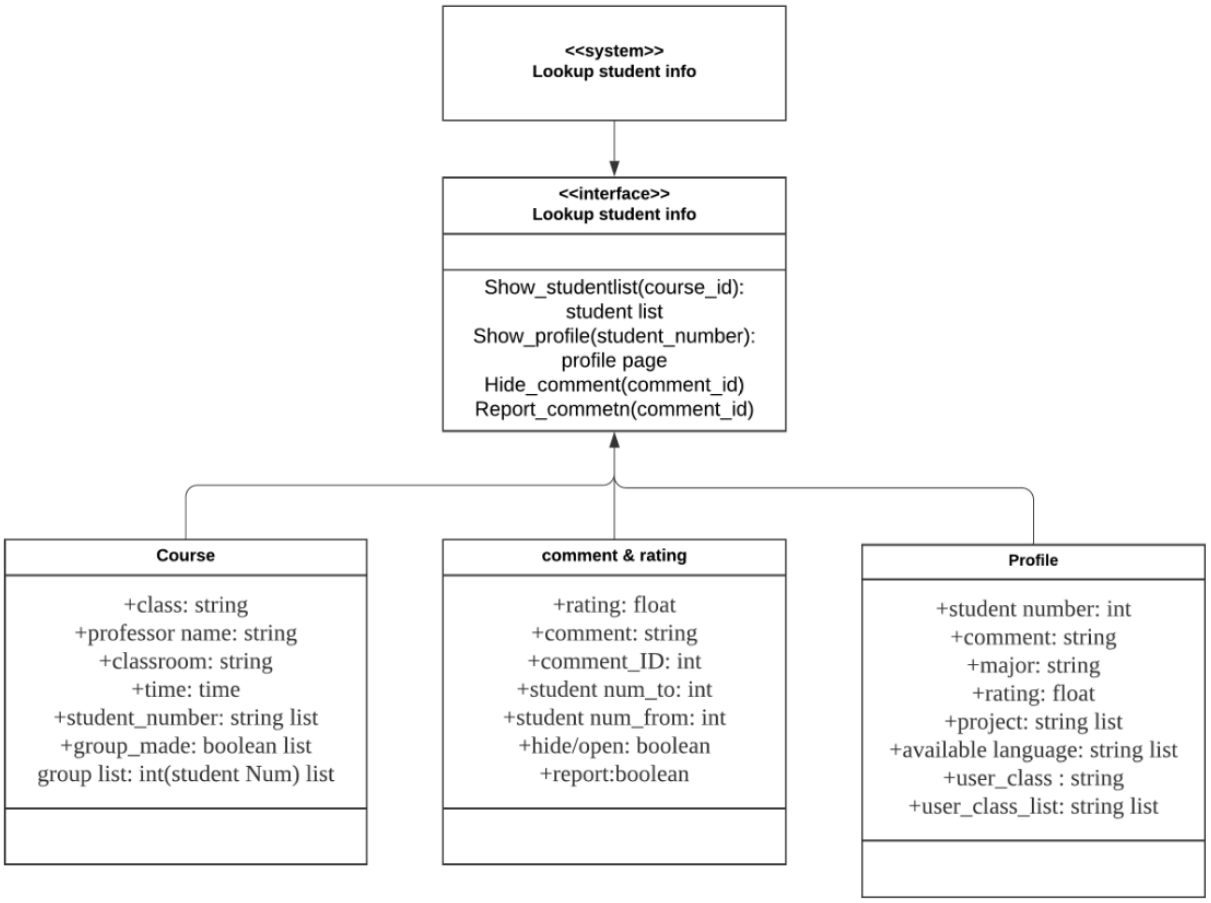
4.2.2.2. **Methods**

These are the methods that the interface has.

- Show_studentlist(course_id) - show the student list of the course
- Show_profile(student number) - show the profile of selected student

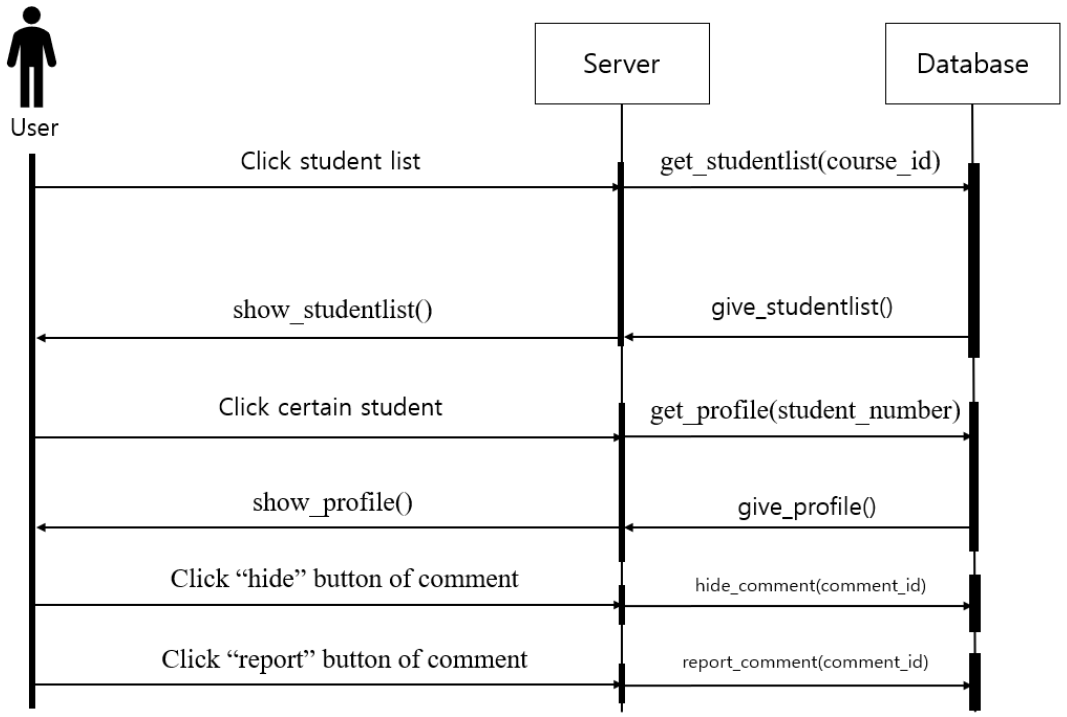
- Hide_comment(comment_id) - hide the inappropriate comment. This function can only be called by the owner of the profile page.
- Report_comment(comment_id) - report the inappropriate comment. This function can only be called by the owner of the profile page.

4.2.2.3. **Class Diagram**



[Figure 7] Class diagram – Lookup student info

4.2.2.4. **Sequence Diagram**



[Figure 8] Sequence diagram – Lookup student info

4.2.3. Course selection

The user has to select a course before looking up the student list. The interface changes the currently selected course of the user.

4.2.3.1. **Attributes**

These are the attributes that “Profile” object has.

- **Student Number:** student number of the user
- **Comment**: the table of the comments
- **Major**: major of the student
- **Rating**: rating of the student
- **project**: the list of finished student’s project
- **Available language**: the language that student can speak
- **User_class**: the class that is chosen by user, for searching team members

- **User_class_list**: the list of class that the user attends

And these are attributes that “Course” object has.

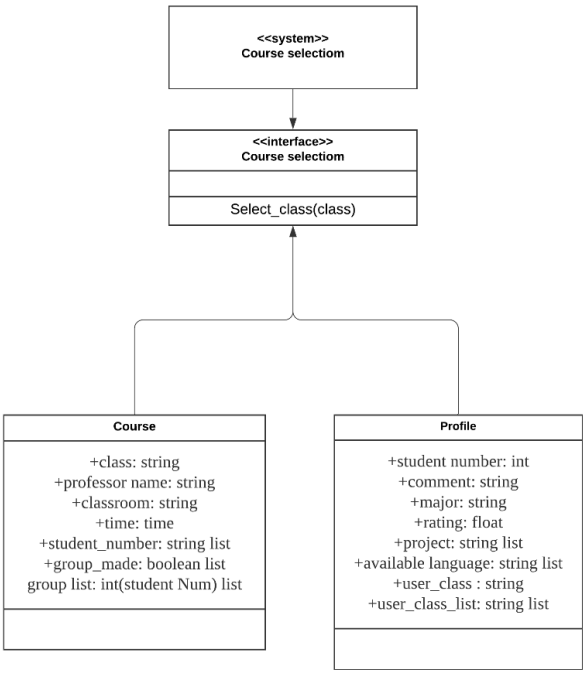
- **class:** class code
- **professor name**: name of the professor
- **classroom**: name of the classroom
- **time**: time of the class
- **student_number:** the list of student id of the students taking the course
- **group made:** the list of flag whether each student has team or not
- **group list:** the list of group in the course

4.2.3.2. **Methods**

These are the methods that Course select class has.

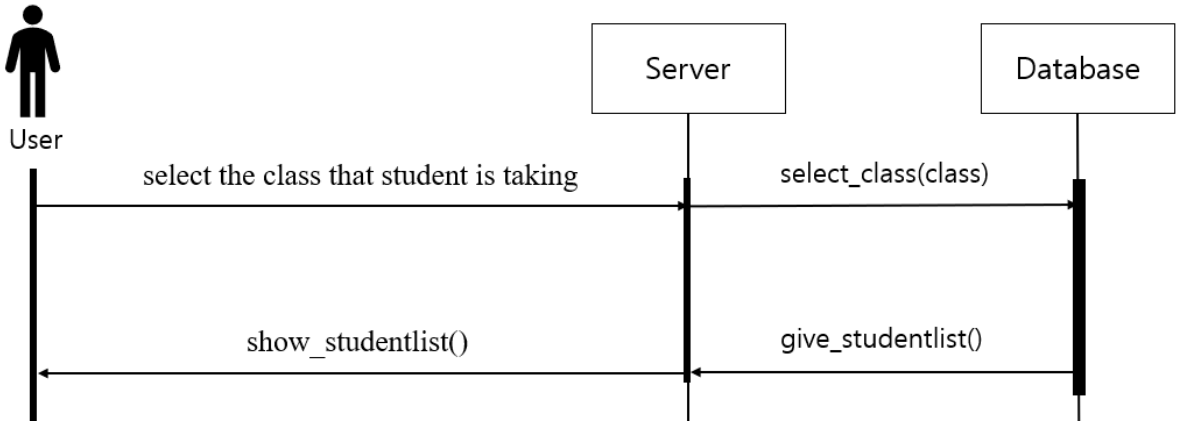
- select_class(class)

4.2.3.3. **Class Diagram**



[Figure 9] Class diagram – Course selection

4.2.3.4. **Sequence Diagram**



[Figure 10] Sequence diagram – Course selection

4.2.4. Team building

The team building class makes team requests and manages their acceptance. If the team is made, all team members should agree with making a team. Also, the professor can deny the team request, although the members want it.

4.2.4.1. **Attributes**

These are attributes that “Course” object has.

- **class:** class code
- **professor name**: name of the professor
- **classroom**: name of the classroom
- **time**: time of the class
- **student_number:** the list of student id of the students taking the course
- **group made:** the list of flag whether each student has team or not
- **group list:** the list of group in the course

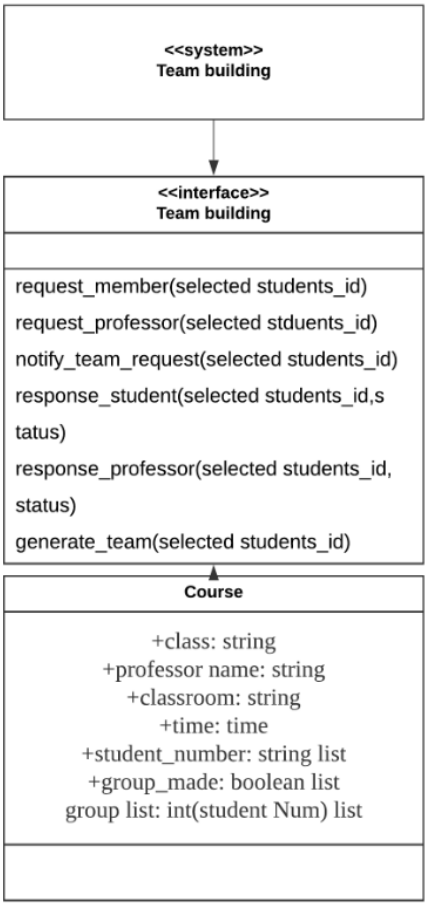
4.2.4.2. **Methods**

These are the methods that team building class has

- request_member(selected students_id) - send requests to selected students.

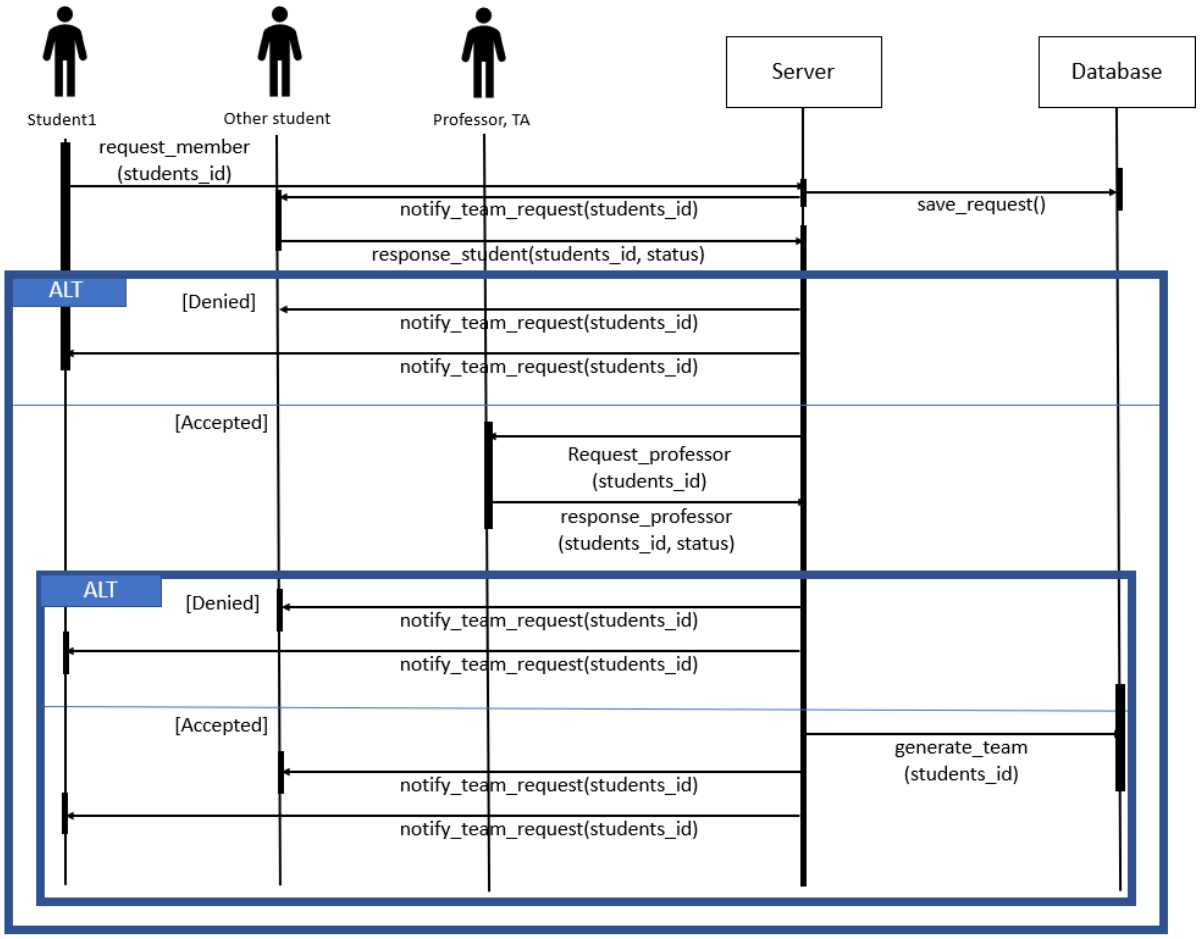
- response_student(selected students_id, status) - send response about the team building request
- notify_team_request(selected students_id, status) - notify the result of team building
- request_professor(selected students_id) - send team building request to professor
- response_professor(selected students_id, status) - send response about the team building request
- generate_team(selected students_id) - generate the team of the selected students

4.2.4.3. **Class Diagram**



[Figure 11] Class diagram – Team building

4.2.4.4. **Sequence Diagram**



[Figure 12] Sequence diagram –Team building

4.2.5. Team management

The team should be shown to the professor with the average rating. And auto team-making is available for the professor.

4.2.5.1. **Attributes**

These are attributes that “Course” object has.

- **class:** class code
- **professor name**: name of the professor
- **classroom**: name of the classroom
- **time**: time of the class

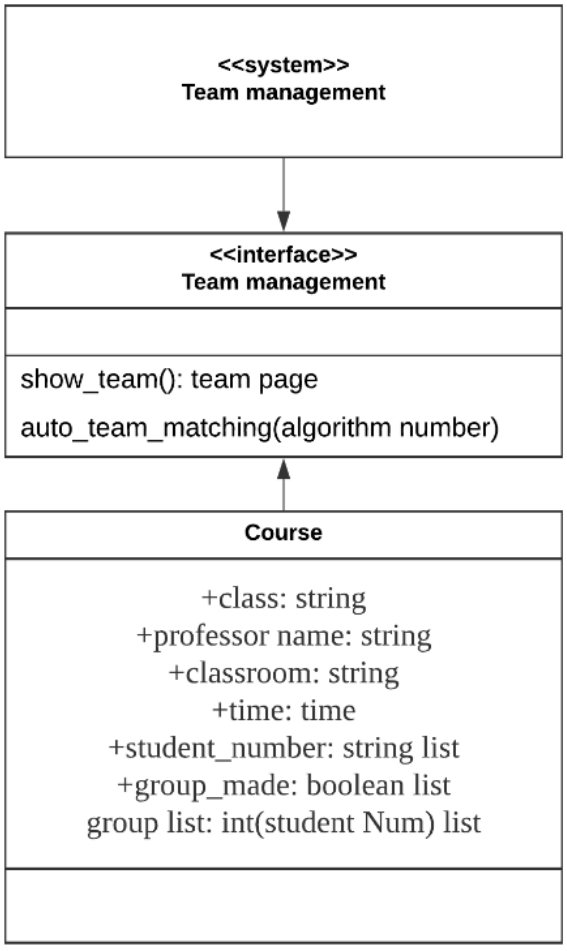
- **student_number:** the list of student id of the students taking the course
- **group made:** the list of flag whether each student has team or not
- **group list:** the list of group in the course

4.2.5.2. **Methods**

These are the methods that profile class has.

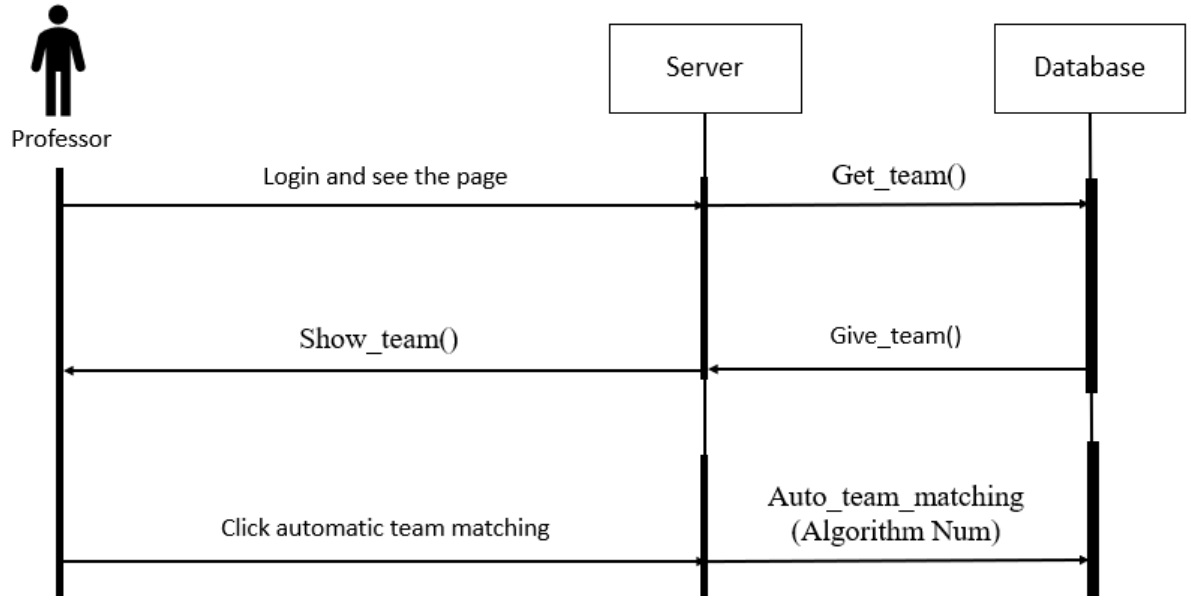
- Show_team() - show team information in the course
- Auto_team_matching(algorithm Number) - make team using selected algorithm with remain students

4.2.5.3. **Class Diagram**



[Figure 13] Class diagram – Team management

4.2.5.4. **Sequence Diagram**



[Figure 14] Sequence diagram – Team management

4.2.6. Evaluation

The evaluation class deals with the evaluation of team members. A user can evaluate the team members and can leave a nested comment.

4.2.6.1. **Attributes**

These are the attributes that “profile” object has.

- **Student Number:** student number of the user
- **Comment**: the table of the comments
- **Major**: major of the student
- **Rating**: rating of the student
- **project**: project that the student has done
- **Available language**: the language that student can speak
- **User_class**: the class that is chosen by user, for searching team members
- **User_class_list**: the list of class that the user attends

And these are attributes that “comment & rating” object has.

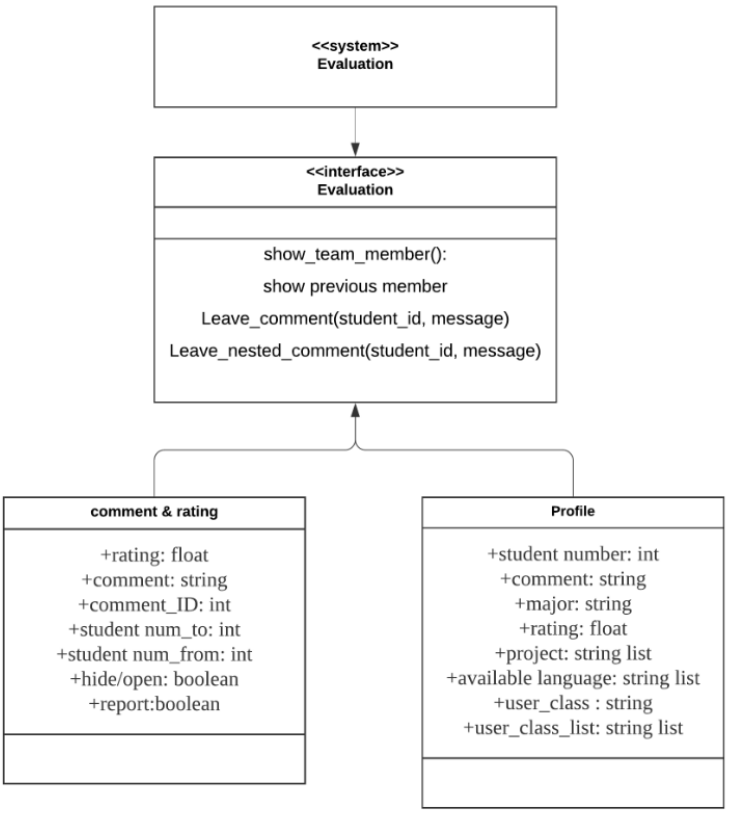
- **Rating:** rating that is given with a particular comment. not overall rating.
- **Comment**: comment string
- **Comment_ID**: id of comment
- **student num_to**: number of the student that is evaluated
- **student num_from**: number of the student that evaluates
- **hide/open**: the flag that the comment is hidden
- **report**: the flag that the comment is reported

4.2.6.2. **Methods**

These are the methods that profile class has.

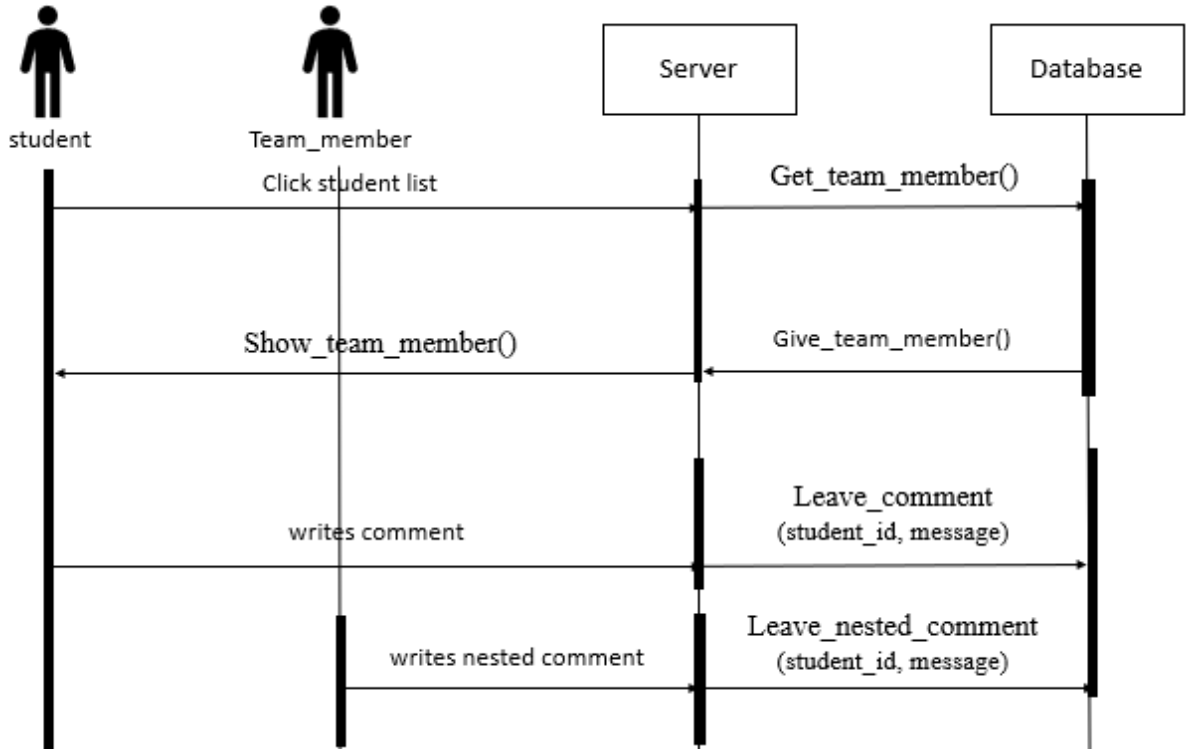
- Show_team_member()
- Leave_comment(student_id, message)
- Leave_nested_comment(student_id, message)

4.2.6.3. **Class Diagram**



[Figure 15] Class diagram –Evaluation

4.2.6.4. **Sequence Diagram**



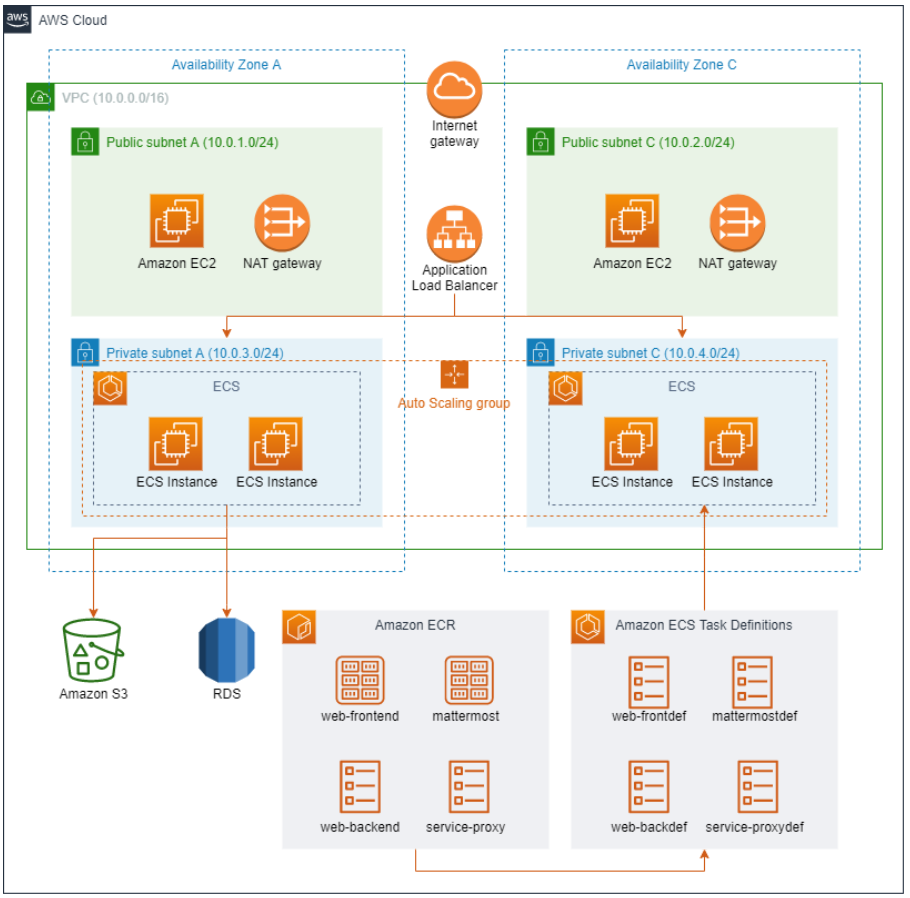
[Figure 16] Sequence diagram – Evaluation

5. System Architecture – Backend

5.1. Objectives

This chapter describes the structure of the back-end system including DB and API Cloud include.

5.2. Overall Architecture



[Figure 17] Overall architecture

The overall Cloud Architecture system is the same as Figure 17. It consists of at least two groups of Availability Zones, each containing a public subnet and a private subnet. Public subnet can communicate externally via the Internet gateway, private subnet can communicate externally and only between instances within the same VPC.

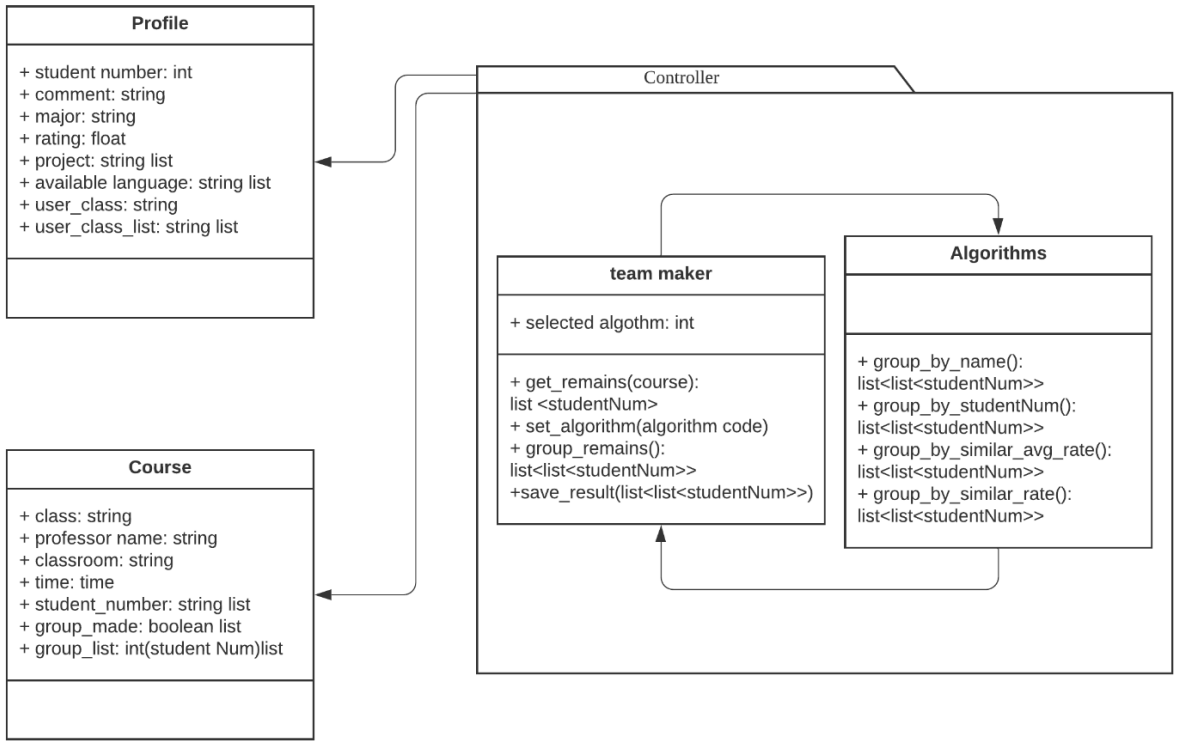
Private subnet consists of a docker container associated with the Hello Project as a microservice. Each container image is hosted on the Amazon ECR.

Each container stores files and data in Amazon S3 and RDS, respectively.

When a request is received from outside the Hello Project, the request is divided equally between the two AZs in a round-robin manner through the Application Load Balancer.

5.3. Subcomponents

5.3.1. Team matching System



[Figure 18] Class diagram – Team matching system

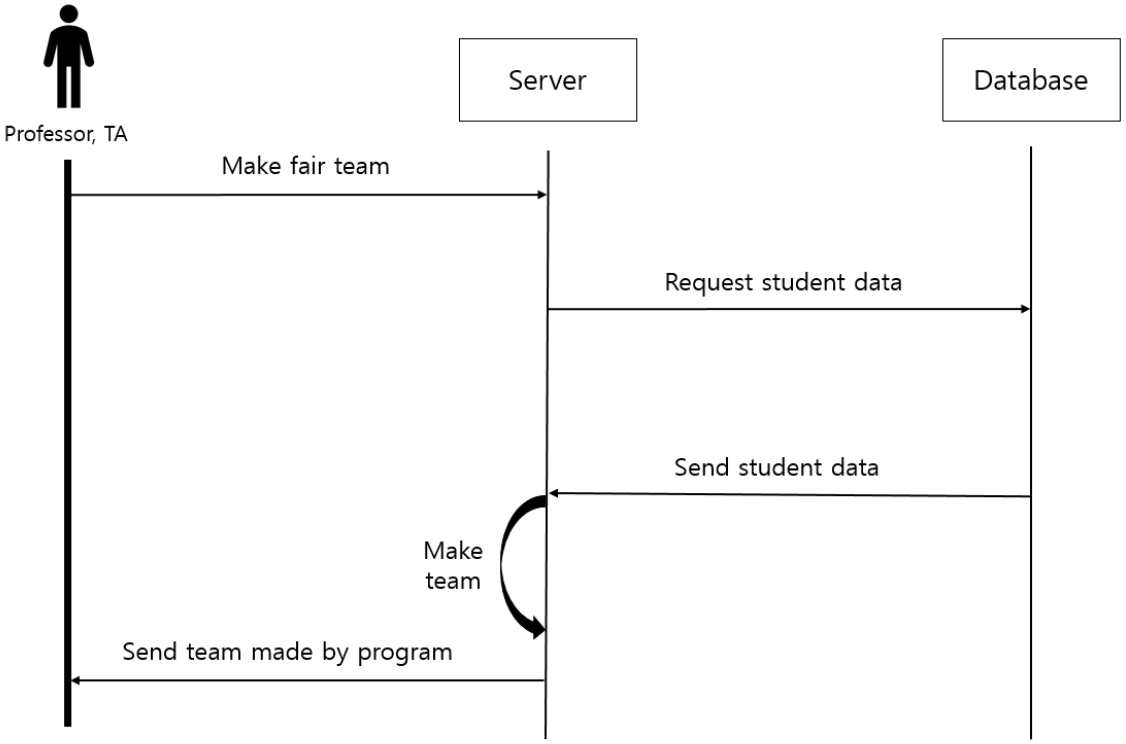
5.3.1.1. **Team maker**

Make the team automatically and save the information through the database. Various algorithms can be selected.

5.3.1.2. **Algorithms**

Saves various algorithms that the professor can select. Includes, group by student id, name and student rating.

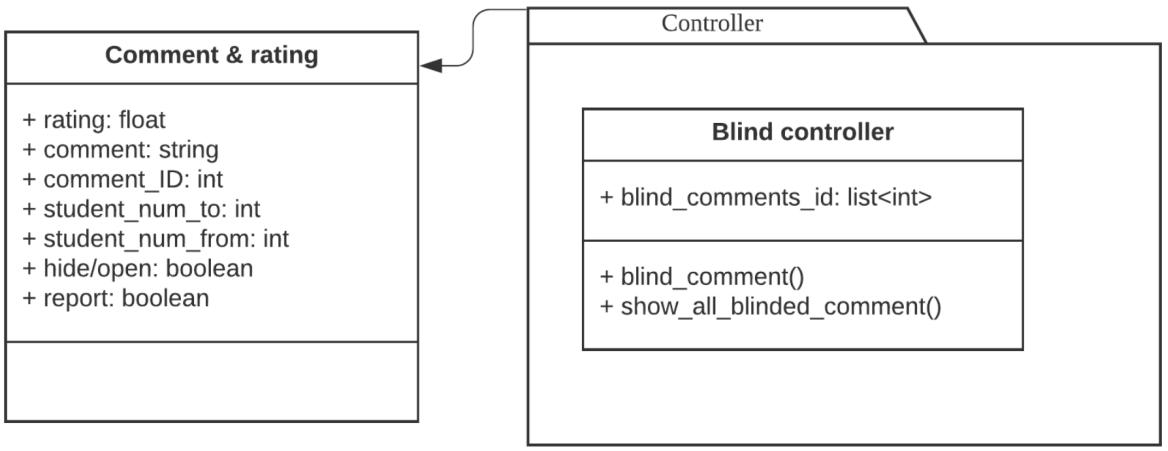
5.3.1.3. **Sequence diagram**



[Figure 19] Sequence diagram – Team matching system

5.3.2. Comment blind system

5.3.2.1. **Class Diagram**

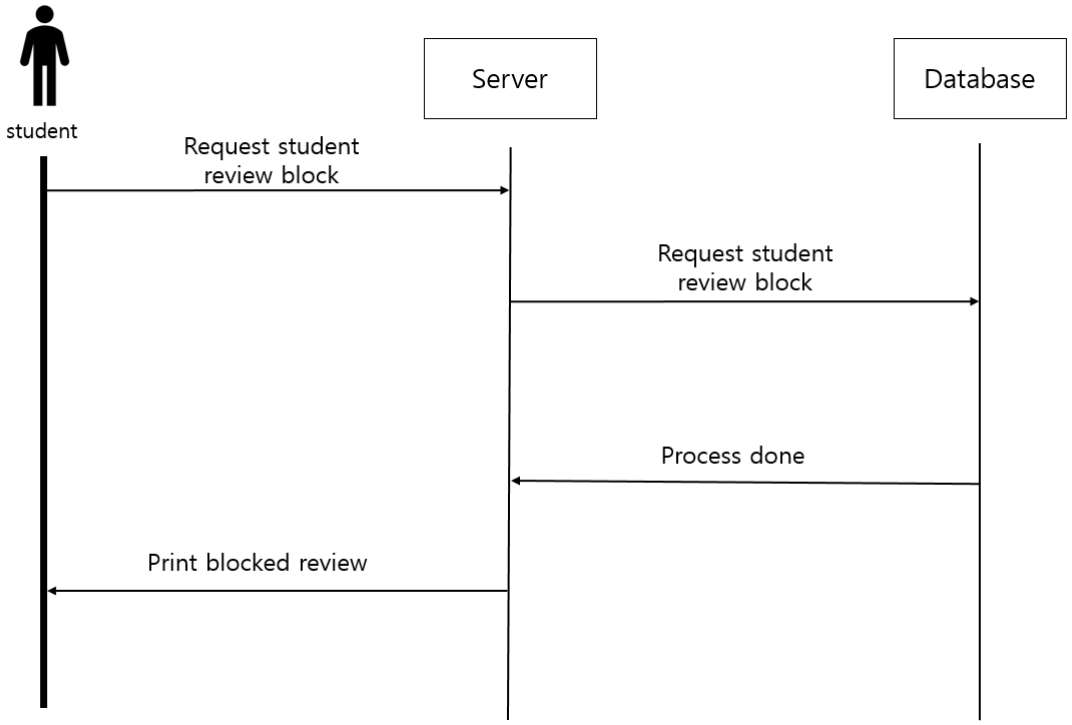


[Figure 20] Class diagram – Comment blind system

5.3.2.2. Blind Controller

If a student sends a blind request about their inappropriate comment, the server can blind the comment.

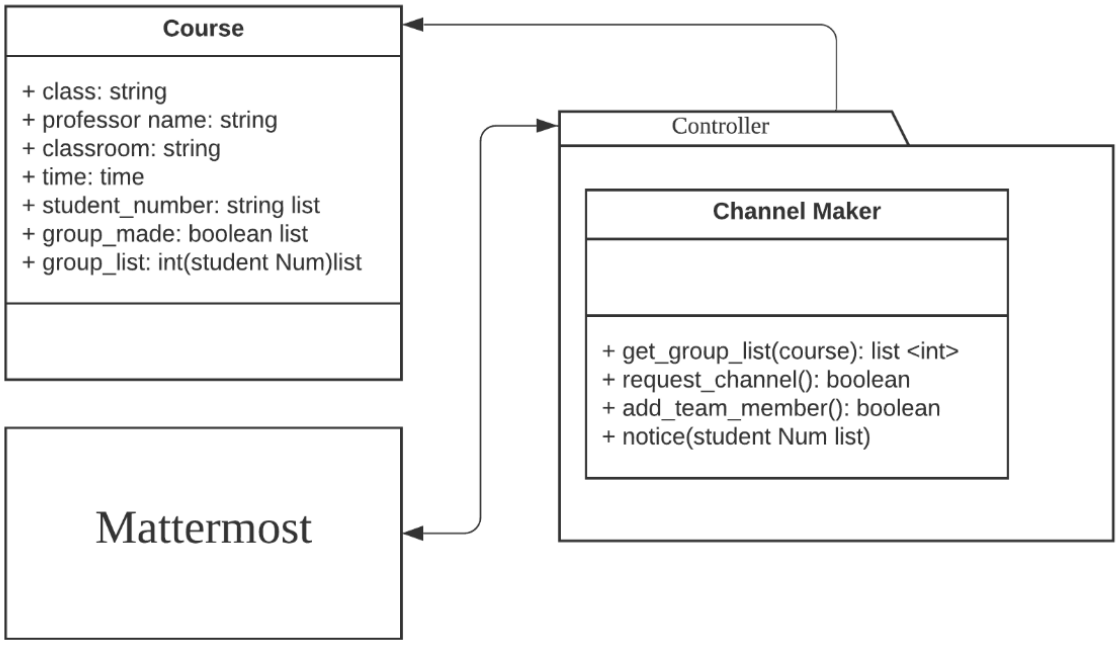
5.3.2.3. Sequence Diagram



[Figure 21] Sequence diagram – Comment blind system

5.3.3. Mattermost Channel Generating System

5.3.3.1. Class Diagram

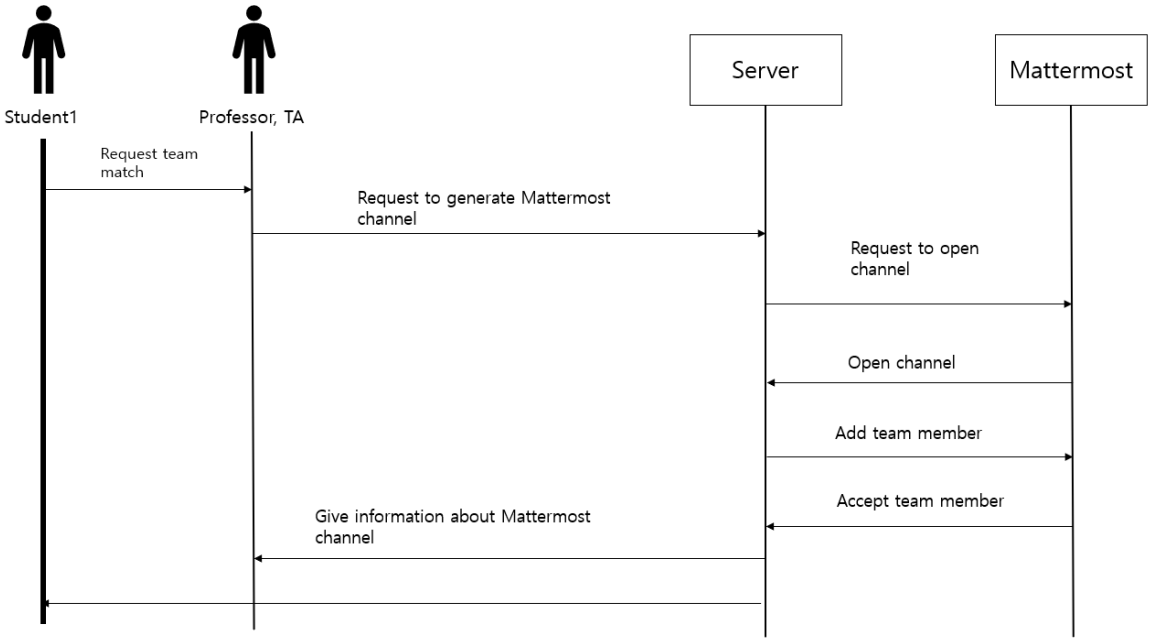


[Figure 22] Class diagram – Mattermost channel generating system

5.3.3.2. Channel maker

It manages the channel making of the matter most. If a new team is generated, the channel maker makes a request for a new channel, and adds team members in the channel. The result would be noticed to team members.

5.3.3.3. **Sequence Diagram**



[Figure 23] Sequence diagram –Mattermost channel generating system

6. Protocol Design

6.1. Objectives

This chapter describes what structures are used for the protocols which are used for interaction between each subsystem, especially the Hello Team Project website and the server. Also, this chapter describes how each interface is defined.

6.2. JSON

JSON(JavaScript Object Notation) is an open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value). It is a very common data format, with a diverse range of applications, such as serving as a replacement for XML in AJAX systems.

In the Hello Project, all network communications except for the database are requested via REST communications. The request, response, which is passed through the REST API, has a

JSON structure.

6.3. Authentication

6.3.1. Login

[Table 1] Specification of Login API

Attribute	Detail		
Protocol	HTTPS		
Method	POST		
URI	/api/v1/login		
Request	Header		X-CSRF-TOKEN: string
	Body		id: string password: string
Response	Success	Status Code	200 OK
		Header	set-cookie: Bearer Token
		Body	none
	Failure	Status Code	400 Bad Request, 401 Unauthorized
		Header	none
		Body	none

6.4. Profile

6.4.1. Get User Profile

[Table 2] Specification of Profile API (GET)

Attribute	Detail
Protocol	HTTPS
Method	GET
URI	/api/v1/info/:id

Request	Header		X-CSRF-TOKEN: string cookie: Bearer Token
	Body		none
Response	Success	Status Code	200 OK
		Header	none
		Body	content: string create_date: string
	Failure	Status Code	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
		Header	none
		Body	none

6.4.2. Set My Profile

[Table 3] Specification of Profile API (POST)

Attribute	Detail		
Protocol	HTTPS		
Method	POST		
URI	/api/v1/info		
Request	Header		X-CSRF-TOKEN: string cookie: Bearer Token
	Body		content: string
Response	Success	Status Code	201 Created
		Header	none
		Body	none
	Failure	Status Code	400 Bad Request, 401 Unauthorized, 403 Forbidden
		Header	none
		Body	none

6.4.3. Update My Profile

[Table 4] Specification of Profile API (PUT)

Attribute	Detail		
Protocol	HTTPS		
Method	PUT		
URI	/api/v1/info		
Request	Header		X-CSRF-TOKEN: string cookie: Bearer Token
	Body		content: string
Response	Success	Status Code	200 OK
		Header	none
		Body	none
	Failure	Status Code	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
		Header	none
		Body	none

6.4.4. Delete My Profile

[Table 5] Specification of Profile API (DELETE)

Attribute	Detail	
Protocol	HTTPS	
Method	DELETE	
URI	/api/v1/info	
Request	Header	X-CSRF-TOKEN: string cookie: Bearer Token
	Body	none

Response	Success	Status Code	200 OK
		Header	none
		Body	none
	Failure	Status Code	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
		Header	none
		Body	none

6.4.5. Upload My Profile Image

[Table 6] Specification of Profile Image API (GET)

Attribute	Detail		
Protocol	HTTPS		
Method	GET		
URI	/api/v1/s3/presigned		
Request	Header		X-CSRF-TOKEN: string cookie: Bearer Token
	Query		name: string (object_name)
Response	Success	Status Code	200 OK
		Header	none
		Body	presigned url: string
	Failure	Status Code	400 Bad Request, 401 Unauthorized, 403 Forbidden
		Header	none
		Body	none

6.5. Team Building

6.5.1. Request Team member

[Table 7] Specification of Request Team member API (POST)

Attribute	Detail		
Protocol	HTTPS		
Method	POST		
URI	/api/v1/team/request		
Request	Header		X-CSRF-TOKEN: string cookie: Bearer Token
	Body		course: id user: id
Response	Success	Status Code	200 OK
		Header	none
		Body	none
	Failure	Status Code	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
		Header	none
		Body	none

6.5.2. Accept Team member

[Table 8] Specification of Accept Team member API (POST)

Attribute	Detail	
Protocol	HTTPS	
Method	POST	
URI	/api/v1/team/accept	
Request	Header	X-CSRF-TOKEN: string cookie: Bearer Token
	Body	request_id: id

Response	Success	Status Code	200 OK
		Header	none
		Body	none
	Failure	Status Code	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
		Header	none
		Body	none

6.5.3. Reject Team member

[Table 9] Specification of Reject Team member API (POST)

Attribute	Detail		
Protocol	HTTPS		
Method	POST		
URI	/api/v1/team/reject		
Request	Header		X-CSRF-TOKEN: string cookie: Bearer Token
	Body		request_id: id
Response	Success	Status Code	200 OK
		Header	none
		Body	none
	Failure	Status Code	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
		Header	none
		Body	none

6.6. Evaluation

6.6.1. Show previous Evaluation

[Table 10] Specification of previous evaluation API (GET)

Attribute	Detail		
Protocol	HTTPS		
Method	GET		
URI	/api/v1/team/evaluation		
Request	Header		X-CSRF-TOKEN: string cookie: Bearer Token
	Query		course: id (course_id) user: id (user_id)
Response	Success	Status Code	200 OK
		Header	none
		Body	content: string
	Failure	Status Code	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
		Header	none
		Body	none

6.6.2. Evaluate Team Member

[Table 11] Specification of Evaluation API (POST)

Attribute	Detail	
Protocol	HTTPS	
Method	POST	
URI	/api/v1/team/evaluation	
Request	Header	X-CSRF-TOKEN: string cookie: Bearer Token

	Query		course: id (course_id) user: id (user_id)
	Body		content: string
Response	Success	Status Code	201 Created
		Header	none
		Body	none
	Failure	Status Code	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
		Header	none
		Body	none

6.6.3. Blind My Evaluate

[Table 12] Specification of blind evaluation API (DELETE)

Attribute	Detail		
Protocol	HTTPS		
Method	DELETE		
URI	/api/v1/team/evaluation/:id		
Request	Header		X-CSRF-TOKEN: string cookie: Bearer Token
	Body		none
Response	Success	Status Code	200 OK
		Header	none
		Body	none
	Failure	Status Code	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
		Header	none
		Body	none

6.7. Match Team

6.7.1. Create Teams and Mattermost Channels

[Table 13] Specification of match teams (POST)

Attribute	Detail		
Protocol	HTTPS		
Method	POST		
URI	/api/v1/team/:course_id		
Request	Header		X-CSRF-TOKEN: string cookie: Bearer Token

	Body		teams: [{
			type: enum,
			payload: {
			member?: [user_id],
			team_id?: id
			}
			}]

Response	Success	Status Code	200 OK
		Header	none
		Body	none
	Failure	Status Code	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
		Header	none
		Body	none

7. Database Design

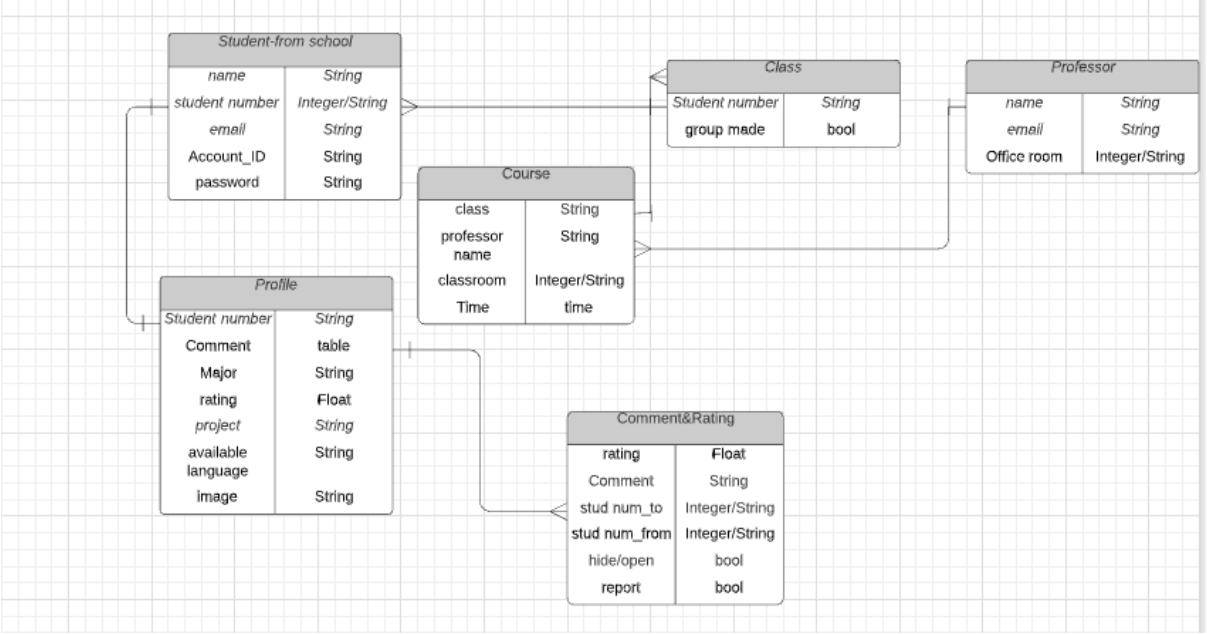
7.1. Objectives

This section describes the system data structures and how these are to be represented in a database. It first identifies entities and their relationship through ER-diagram (Entity

Relationship diagram). Then, it generates Relational Schema and SQL DDL (Data Description Language) specification.

7.2. Relational Schema

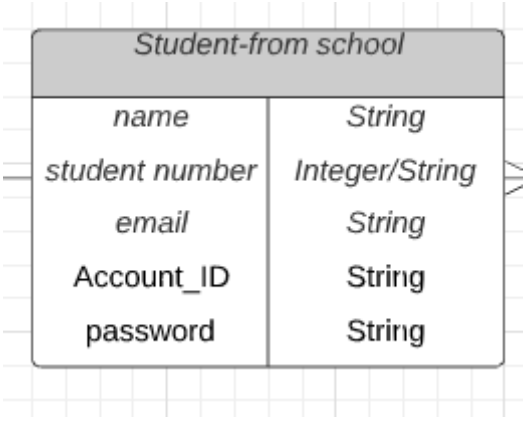
The system consists of six entities; Student-from school, profile, class, course, professor and Comment&Rating.



[Figure 24] Relational-Schema

7.2.1. Entities

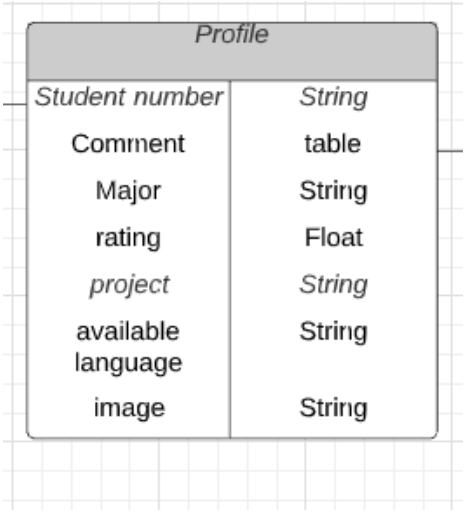
7.2.1.1. **Student-from school**



[Figure 25] Relation schema, Entity, Student-from school

Student-from school table is the table and information given by the university. From the university main database, we get name, student number, email address, account_ID and password. From the information the university provides, we made a table of it to receive basic information of the user and log-in with the university's account and password.

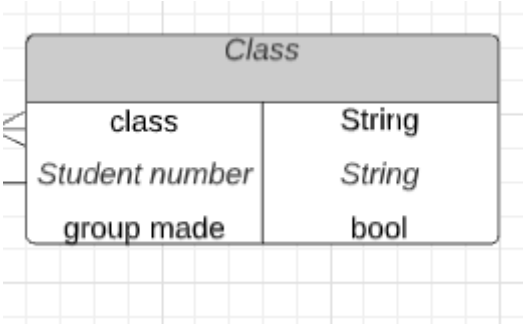
7.2.1.2. **Profile**



[Figure 26] Relation schema, Entity, Profile

Profile represents the profile of the user. It has a student number given by the student-from school table and has some datas which are major, rating, and available language. Also, user can write their experiences about the project and take care of the comment written by teammates. Finally, there are some images to show yourself. Image is saved in database with their directory

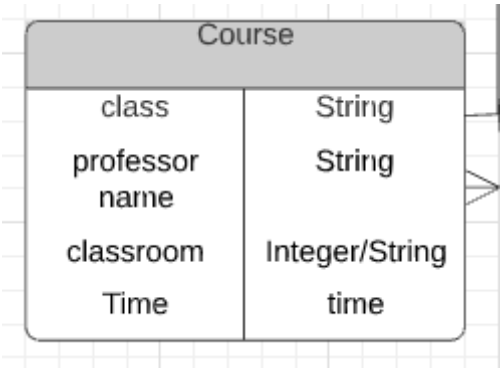
7.2.1.3. **Class**



[Figure 27] Relation schema, Entity, Class

Class table has three attributes, which are class student number and group made. Since our software focuses on making teams, we made an extra table to check if the student in that class has made a group or not using the flag.

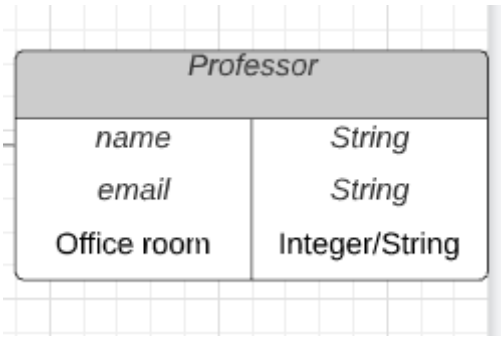
7.2.1.4. **Course**



[Figure 28] Relation schema, Entity, Course

Course contains information about classes held in that semester. Course table has class name, professor name, classroom and time information.

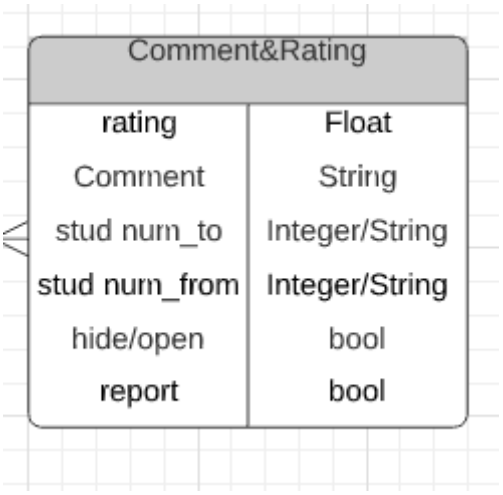
7.2.1.5. **Professor**



[Figure 29] Relation schema, Entity, Professor

Since professor is also a user of our software, we save some basic information about professor, such as name, email address and office room in our database.

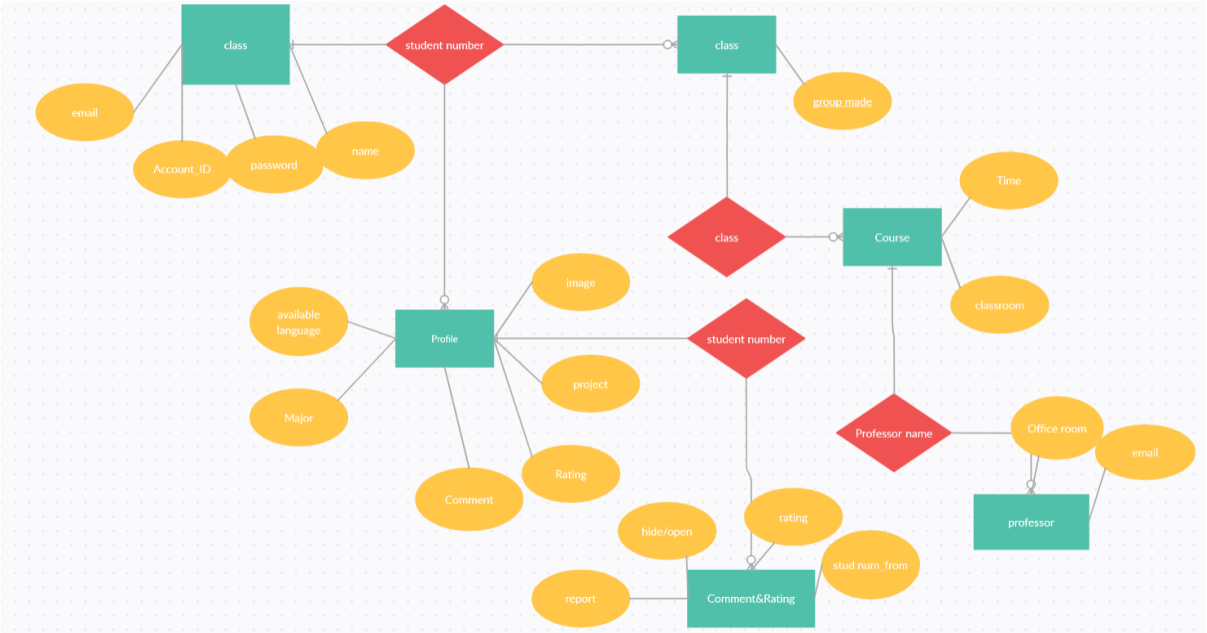
7.2.1.6. **Comment&Rating**



[Figure 30] Relation schema, Entity, Comment&Rating

One of our key functions of our software is to make comments and rate the teammate user. Database of our software has to store the comment and rating. Therefore, we made another table called Comment&Rating. At this table, we have students who got comments and ratings from others, and students who made comments and rates. Also, we store the comment and rates. Also, users can report about this comment and rating if the user thinks this is unfair and needs to be fixed or erased. Also, administrators can hide this comment and rates if they want. To allow this function, we made a hide/open and report flag on the database.

7.3. ER Diagram



[Figure 31] ER-diagram

ER-diagram expresses each entity as rectangular and their relationship as rhombus. The attribute of an entity is expressed as an ellipse. The unique attribute which uniquely identifies an entity is underlined. If an entity has a number of same attributes, that attribute is expressed as ellipse with double border line.

7.4. SQL DDL

7.4.1. Student-from school

```
CREATE TABLE Student-from school
(
 name STRING NOT NULL,
 student_number INT NOT NULL,
 email INT NOT NULL,
 student_ID STRING NOT NULL
 password STRING NOT NULL,
 PRIMARY KEY (student_number)
);
```

7.4.2. Profile

```
CREATE TABLE Profile
(
 student_number INT NOT NULL,
 comment STRING NOT NULL,
 rating INT NOT NULL,
 major STRING NOT NULL,
 project STRING NOT NULL,
 available_language STRING NOT NULL,
 image STRING,
   FOREIGN KEY (student_number) REFERENCES
Student-from school(student_number),
);
```

7.4.3. Class

```
CREATE TABLE Class
(
 class STRING NOT NULL,
 group_made BOOL  NOT NULL,
 student_number INT NOT NULL,
   FOREIGN  KEY  (student_number)  REFERENCES
Student-from_school(student_number),
);
```

7.4.4. Course

```
CREATE TABLE Course
(
 class STRING NOT NULL,
 professor_name STRING NOT NULL,
 classroom STRING NOT NULL,
 time STRING NOT NULL,
  PRIMARY KEY (class),
FOREIGN  KEY  (professor_name)  REFERENCES
Professor(professor_name),
);
```

7.4.5. Professor

```
CREATE TABLE Professor
(
 name STRING NOT NULL,
 email STRING,
 office_room STRING NOT NULL,
);
```

7.4.6. Comment&Rating

```
CREATE TABLE Comment_Rating
(
 rating FLOAT NOT NULL,
 comment STRING NOT NULL,
 stud_num_to INT NOT NULL,
 stud_num_from INT NOT NULL,
 hide_open BOOL NOT NULL,
 report BOOL NOT NULL,
 PRIMARY KEY (stud_num_to),
);
```

8. Testing Plan

8.1. Objectives

This chapter illustrates plans on testing which has three major sub-groups of development testing, release testing, and user testing. These tests are crucial in that they detect potential errors and defects of the product and ensure flawless operation and stable release of a product to the university and students.

8.2. Testing Policy

8.2.1. Development Testing

Development testing is carried out mainly for synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce potential risks of software development and save time and costs.

In this phase, since the software has not undergone enough tests, it can be unstable, and components can collide with each other. Therefore, static code analyzing, data flow analyzing, peer code review, unit testing need to be carried out in this stage. Through these processes, we are mainly focusing on achieving 1) performance which defines the identity of our software, 2) reliability for ensuring safe and failure-free operation, and 3) security.

8.2.1.1. **Performance**

Since the fair-team matching algorithm takes most time-consuming operation in our system, it is crucial for developers to shorten the time by using a special algorithm. The system should give users(usually professor or TA) the result within 5 seconds. We would prepare test cases on various preferences and evaluate the speed of the fair-team matching function, and improve the flow of code regarding the fair-team matching algorithm and communication with the server.

8.2.1.2. **Reliability**

In order for the system to operate safely without failure, the sub-components and units comprising the system should operate and be connected correctly first. Therefore, we need to undergo development testing from the unit developing stage and check failure iteratively while each unit is integrated to the system.

8.2.1.3. **Security**

Securing information of users and the system is a crucial matter to be handled by developers. Regardless of the value of the information, it should be protected from unwanted visitors to the system.

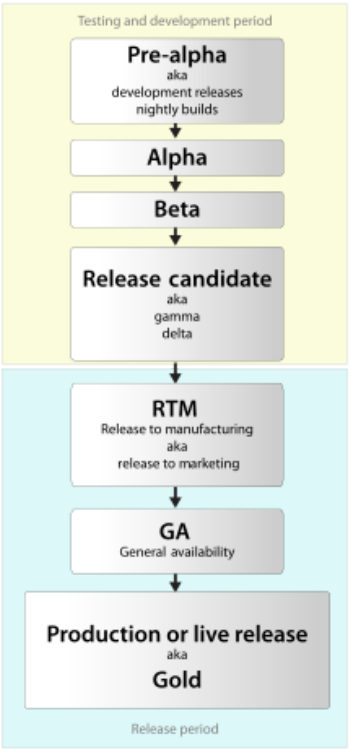
In order to achieve security of the system, we can access a nearly finished version of the app, identify security issues and write the report through manual code review. Also, we can take advantage of other web page security testing services provided by SUCURI, Qualys, etc., which would let developers know overlooked vulnerable points in the web.

8.2.2. Release Testing

One of the most critical parts of any software development project is to release the product to the customers(in our project, it is usually university). A technically good software can go wrong due to a wrong way of release. Therefore, release testing is inevitable for better connection between the product and the customer. Release testing is testing a new version of a software/application to verify that the software can be released flawlessly, so it doesn’t have any defects in its working. It should be carried out before release.

Based on Software Release Life Cycle, testing is generally initiated from the ‘Alpha’ version, where a basic implementation of the software is completed. We would start development

testing in Alpha version, and release Beta version for further testing including user and release testing.



[Figure 32] Software Release Life Cycle

After the Beta version is released, we would get feedback from actual users as well as developers.

8.2.3. User Testing

We should set up possible scenarios and realistic situations that can proceed with necessary user tests. We assume that there would be 200 users on the web simultaneously. After setting this situation, we would distribute Beta versions of the web to them and collect user reviews while carrying out our own use cases test.

8.2.4. Testing Case

Testing case would be set according to 3 fundamental aspects of the application– function(interaction), performance, and security. We would set 5 test cases from each aspect (15 cases in total) and proceed testing on the web and would make an evaluation sheet.

9. Development Plan

9.1. Objectives

This chapter illustrates the technologies and environment for the development of the application.

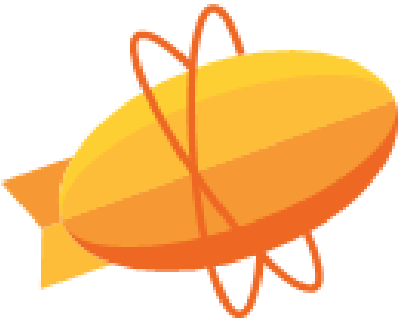
9.2. Frontend Environment

9.2.1. Adobe Photoshop

[Figure 33] Adobe Photoshop logo

It is a raster graphics editor developed and published by Adobe Inc. for Windows and macOS. Photoshop will be used to design user interfaces.

9.2.2. Zeplin



[Figure 34] Zeplin logo

Zeplin is a collaboration tool between designers and developers. Zeplin automatically generates specs, assets and code snippets from designs tailored for the desired platform. Zeplin will be used to convert photoshop assets to css assets.

9.3. Backend Environment

9.3.1. Git

[Figure 35] Git logo

Git is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows. And github also will be used to save and manage the project progress.

9.3.2. MariaDB

[Figure 36] MariaDB logo

MariaDB is a community-developed, commercially supported fork of the MySQL relational database management system (RDBMS), intended to remain free and open-source software under the GNU General Public License. MariaDB will be used to establish the database system that stores information of users and courses.

9.3.3. Postman

[Figure 37] Postman logo

Postman is a collaboration platform for API development. Postman's features simplify each step of building an API and streamline collaboration. It will be used at REST API testing. Specifically we will check requests of CRUD which means create, read, update, delete. After we check the correct input case and it's output, we will check the case of wrong input and it's output. For example we will check the error message like 401, 403, 404 when there is no user session or no authority.

9.4. Constraints

The system will be designed and implemented based on the contents mentioned in this document. Other details are designed and implemented by selecting the direction preferred by the developer, but the following items are observed.

- Use the technology that has already been widely proven.
- The service response speed does not exceed 3 seconds.
- Develop the student team project tool such as Gantt chart or ghost leg.
- Avoid using technology or software that requires a separate license or pays for royalty. (Exclude this provision if this is the only technology or software that the system must require.)
- Decide in the direction of seeking improvement of overall system performance.
- Decide in a more user-friendly and convenient direction
- Use open source software whenever possible
- Consider the system cost and maintenance cost

- Consider future scalability and availability of the system
- Optimize the source code to prevent waste of system resources
- Consider future maintenance and add sufficient comments when writing the source code
- Develop with HTML5, CSS3, and Javascript ES6

9.5. Assumptions and Dependencies

All systems in this document are written on the assumption that they are designed and implemented based on web site and open source. Therefore, all contents are written based on the HTML5, CSS3, and Javascript ES6.

10. Supporting Information

10.1. Software Design Specification

This software design specification was written in accordance with the IEEE Recommendation (IEEE Recommended Practice for Software Design Description, IEEE-Std-1016).

10.2. Document History

[Table 14] Document History

Date	Version	Description	Writer
2021/05/04	0.1	Addition of 1, 2, 3	Byeongsu Woo
2021/05/06	0.2	Addition of 9, 10	Byeongsu Woo
2021/05/08	0.3	Addition of 4	Jaehyun Ju
2021/05/11	0.4	Addition of 6	Seongwook Lim
2021/05/11	0.5	Addition of 7	Heesoo Jung
2021/05/13	0.6	Addition of 8	Min Jang
2021/05/13	1.0	Distribution	