Architecture Document

SWEN90007

Online Examination System

Team: A Team

In charge of:

Student Name	Username	Student Id	Email
Zihan Zhang	zhazz	963790	zhazz@student.unimelb.edu.au
Chengeng Liu	chengengl	813174	chengengl@student.unimelb.edu.au
Hao Ding	dingh2	825702	dingh2@student.unimelb.edu.au

SCHOOL OF
COMPUTING &
INFORMATION
SYSTEMS

Revision History (this template's)

Date	Version	Description	Author
25/09/20	01.00-D01	Initial draft	Hao Ding
26/09/20	01.00-D02	Added architectural patterns	Hao Ding
27/09/20	01.00-D03	Added sequence diagrams	Hao Ding
01/10/20	01.00-D04	Added example of use&test	Hao Ding
04/10/20	01.00-D05	Added functionality testings	Zihan Zhang
04/10/20	01.00-D06	Added Architectural layers	Hao Ding
04/10/20	01.00	Improve format	Zihan Zhang

Table of Contents

1. Introduction

1.1 Proposal

This document illustrates the high level architecture of the online examination system, which can be seen from the domain model diagram and the class diagram. Moreover, this document records the implementations and justifications of the architectural patterns.

1.2 Target Users

This document is mainly intended for SWEN90007 students and teaching team. In addition, the developing team and the system administrator are also encouraged to use this document as a reference to the system,

1.3 Conventions, terms and abbreviations

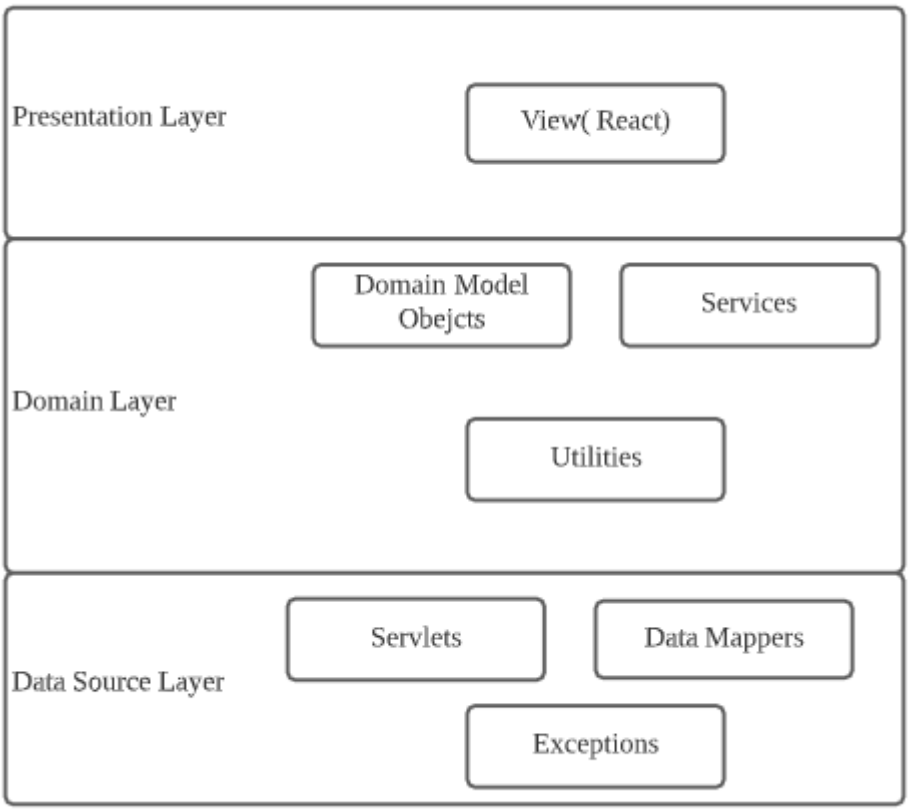
This section explains the concept of some important terms that will be used throughout this document. These terms are detailed alphabetically in the following table.

Term	Description
LMS	Learning Management System
O2R	Object to Relational

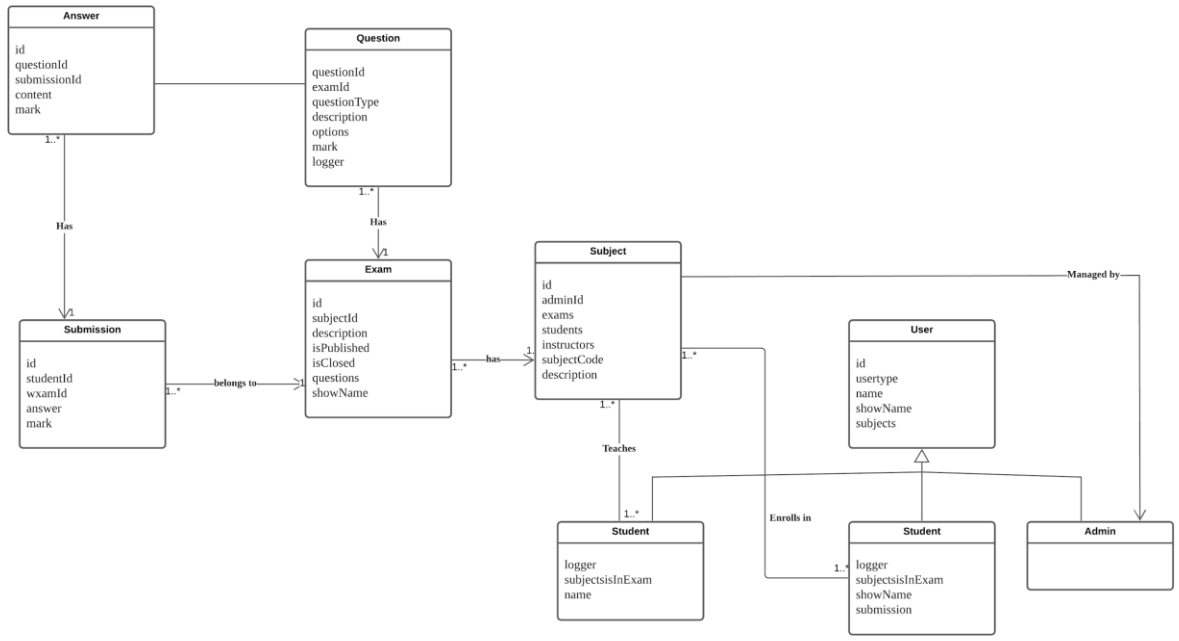
2. Architectural representation

We designed the system according to three layers: presentation layer, domain layer, and data source layer.

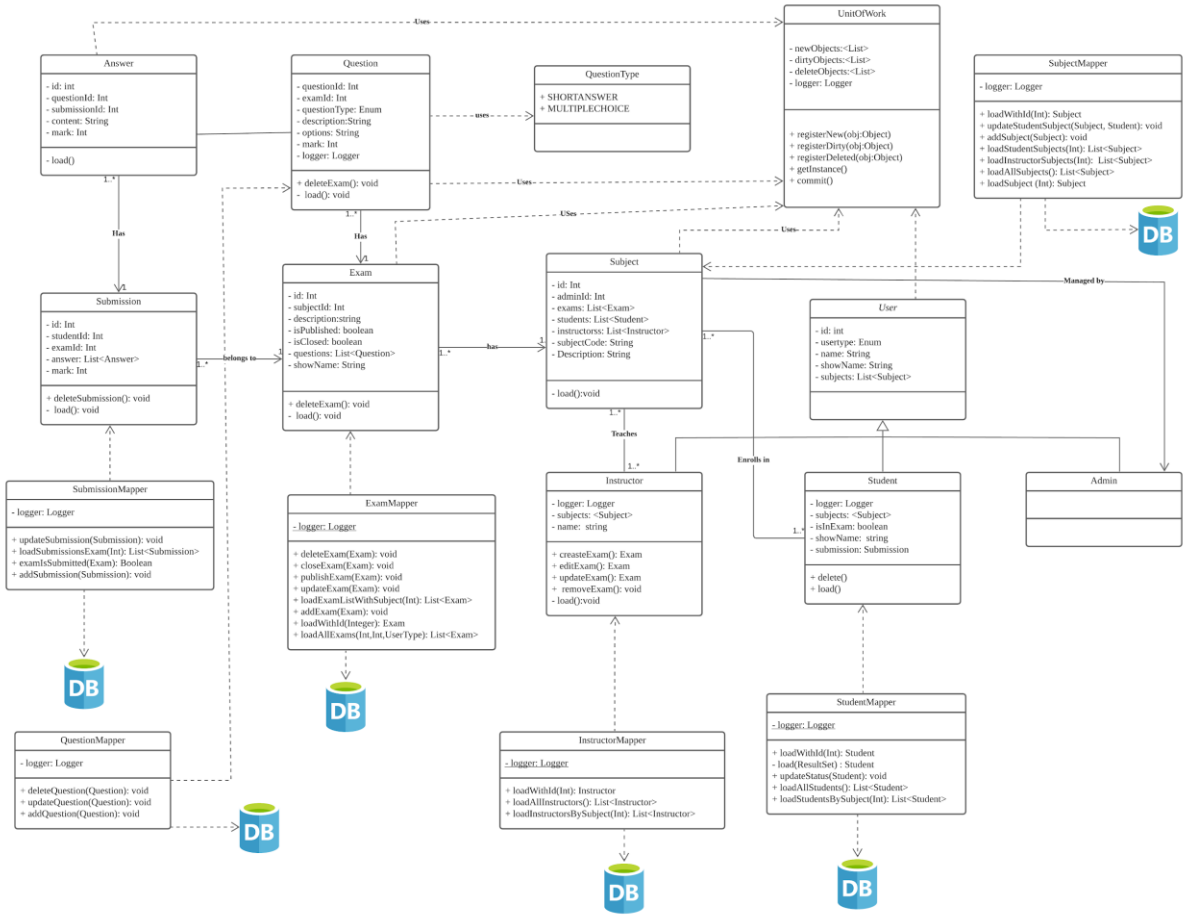
- The presentation layer displays the system to the users, it is the interface between user interactions and the system. We used React as the frontend framework. All the user inputs are sent from here to the lower layers.
- The domain layer contains all the objects as well as their business logic. Furthermore, all the objects to relational behavior patterns are implemented in this layer. Although this layer is not connected with the database directly, we utilized the service and utility components to track the modifications on objects before sending the requests to the data source layer.
- The data source layer controls the requests sent in and out from the database. We used the data mapper as the gateway to manage the requests sent from the higher layers, this layer maps the domain layer objects to the corresponding tables and rows in the database. Nevertheless, all the objects to relational structural patterns are implemented in this layer.



2.1 Domain Model Diagram:



2.2 Class Diagram:



3. Architectural Patterns

This section illustrates all the architectural patterns we implemented in this project; we also gave the justifications for the intentions to use the patterns. Furthermore, we provided one example of use for each pattern and a sequence diagram is accompanied with each pattern to visual the example of use.

We can sort the patterns into two main categories.

Objects to Relational Structural Patterns ensures the efficiency of integrity of fetching data from the database, and saving the updated data to the appropriate tables. O2R structural patterns used in this project includes: Unit of Work, Lazy Load and Identity Map.

Objects to Relational Behavioral Patterns effectively map the domain objects to the relational tables in the database. O2R behaviour patterns used in this project includes: Identity Field, Foreign Key Mapping, Embedded Value, ASsociation Table Mapping, and Single Table inheritance.

3.1 < Identity Field Pattern >

Pattern	Reason

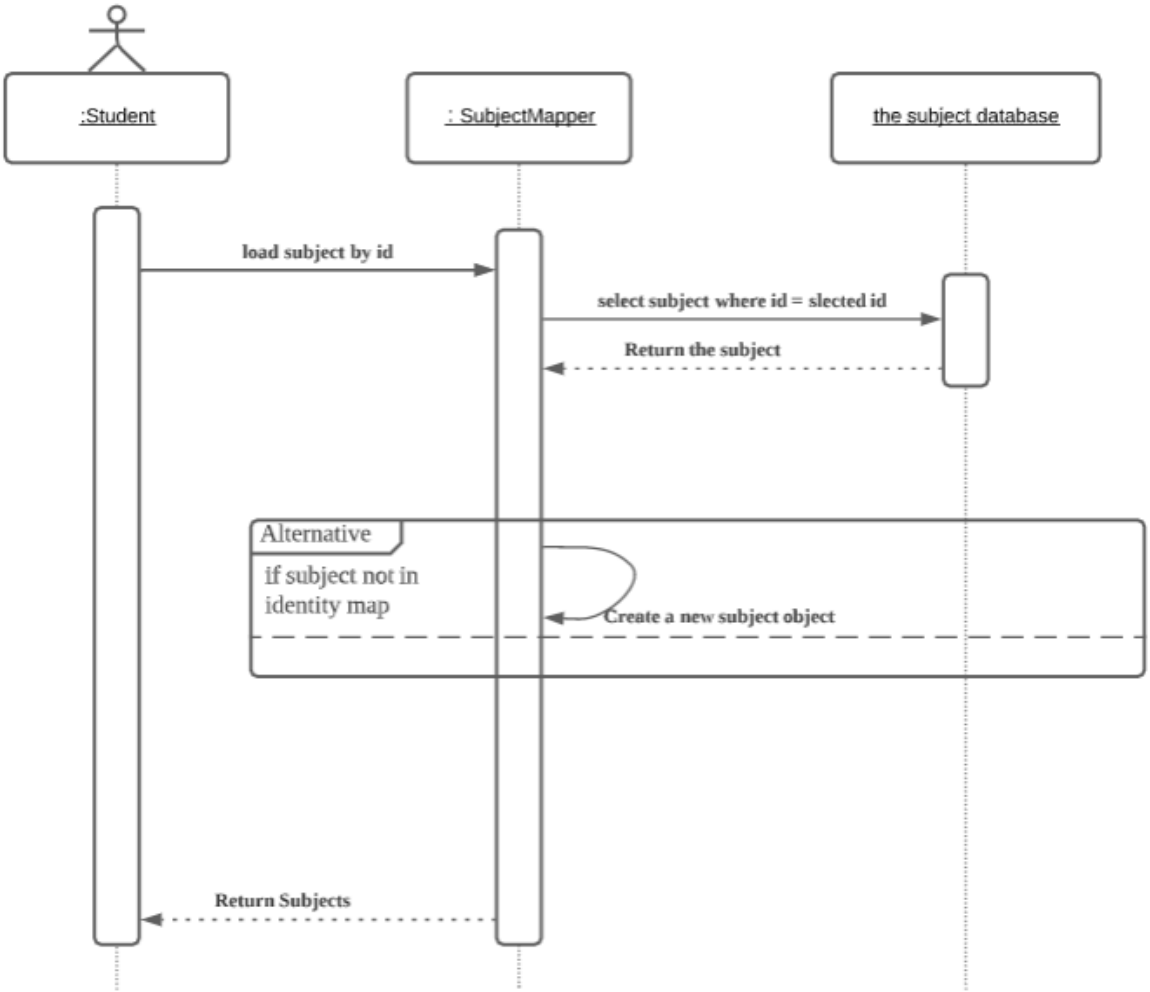
Identity Field	This pattern keeps track of an identity between a specific database row and a local object. This pattern is easy to implement, easy to understand, it also helps the system to effectively identify the relevant rows in the database.

Implementation:

We implemented the identity field pattern whenever there is a mapping relationship between a specific row in the database and an in-memory object. For example, each subject has an auto-generated id, this key is simple and database unique. As a result, the database can use the id to identify the specific row in the subject table, so we can accurately perform operations to the selected subjects. Nevertheless, all the objects in the design have a database id field, so we can map the in-memory objects to the corresponding database rows.

Examples of Use:

When a student opens his/her dashboard, the system will first find all the related subjectId(s) with the student. The system then retrieves the subjects from the subject table by using the subjectId(s). Moreover, the subject database finds the wanted row and sends the data back to the system.



3.2 < Data Mapper Pattern >

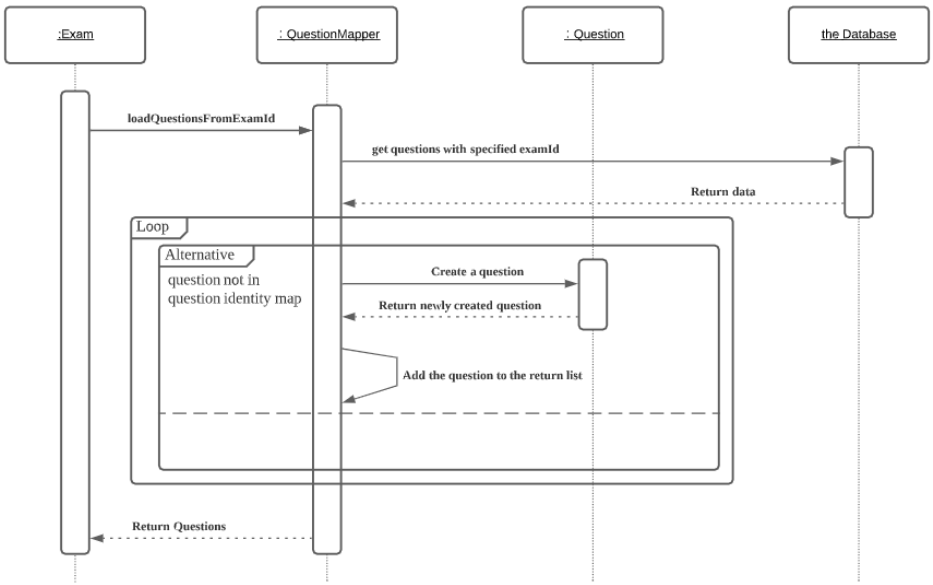
Pattern	Reason
Data Mapper	The data mapper effectively keeps the database and the objects independent of each other. This pattern decreases the overall coupling by adding an extra layer between the domain layer and the data source layer. Therefore, the domain layer objects need not to know any SQL codes, or even not be aware of any information of the database. Moreover, since we already used the domain model pattern in this project, the data mapper pattern is very compatible with the domain pattern. For example, the domain model objects can be created by the data mappers from the database. Nevertheless, data mappers and domain objects can be efficiently reused.

Implementation:

Because the data mapper pattern provides a gateway between the database and the domain objects. We implemented this pattern wherever an object has interactions with the database. For example, the question mapper helps the other objects to retrieve, update, add, and delete questions from the database.

Examples of Use:

When a student starts taking an exam, the system must load the exam with relevant questions. The exam object would call a find method(loadQuestionsFromExamId) on the Question Mapper. The mapper would retrieve all the questions with the specified examId from the database. The mapper utilizes the identity map pattern to check whether each question is already loaded. If the question has not been loaded yet, create a new question and add it to the return list; otherwise continue to the next question. Furthermore, the mapper returns the processed question list to the exam.



3.3 < Association Table Mapping Pattern >

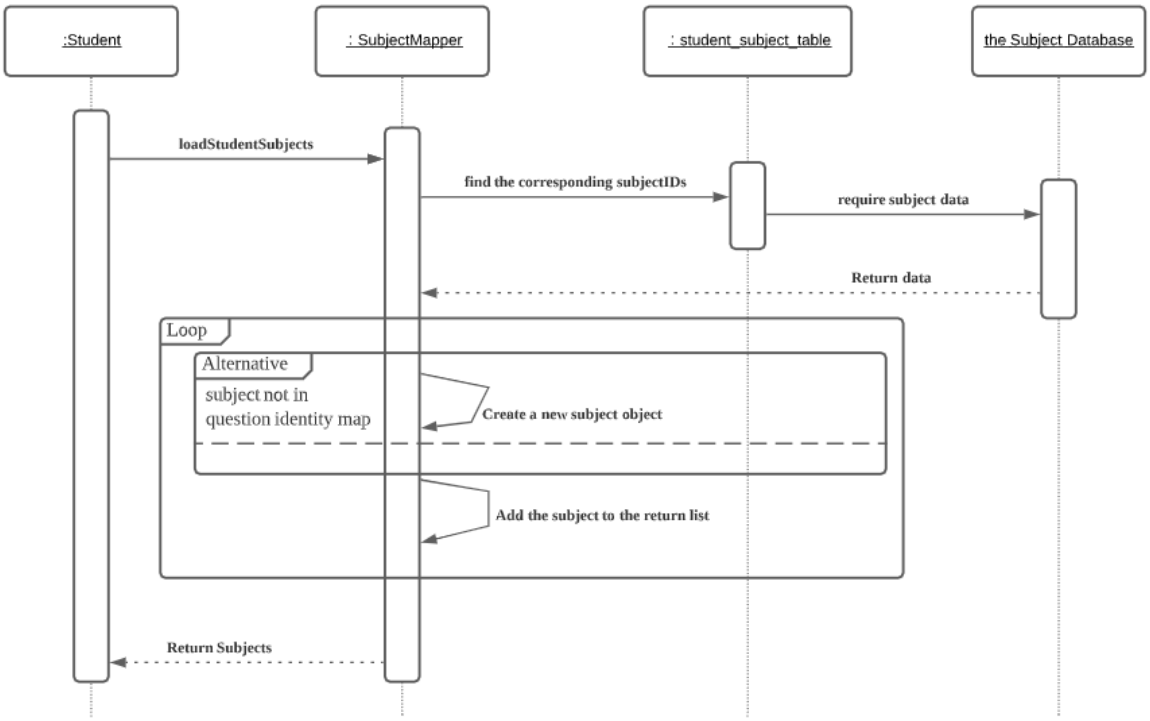
Pattern	Reason
Association Table Mapping	For many to many relationships, The association table mapping pattern creates a separate table to store foreign keys from both objects, which can mechanically map the relationships between the domain layer and the database.

Implementation:

The association table mapping pattern is implemented for all the many-to-many relationship objects. For example, a student can enroll in multiple subjects, whereas a subject can accommodate multiple students. As a result, we created a new table called student_subject_relation, which stores all the relevant studenId and subjectId in the same rows. Therefore, we can efficiently identify a stundent's enrolled subjects, as well as to find out the student list for a particular subject.

Examples of Use:

After logging into the system, the student is able to see all his/her enrolled subjects from the dashboard. The system first uses the student_subject association table to collect all the subjectIDs related to the student, and then the system sends requests to retrieve all the required subjects from the database. Finally, the system mechanically displays all the desired subjects to the student user.



3.4 < Foreign key Mapping Pattern >

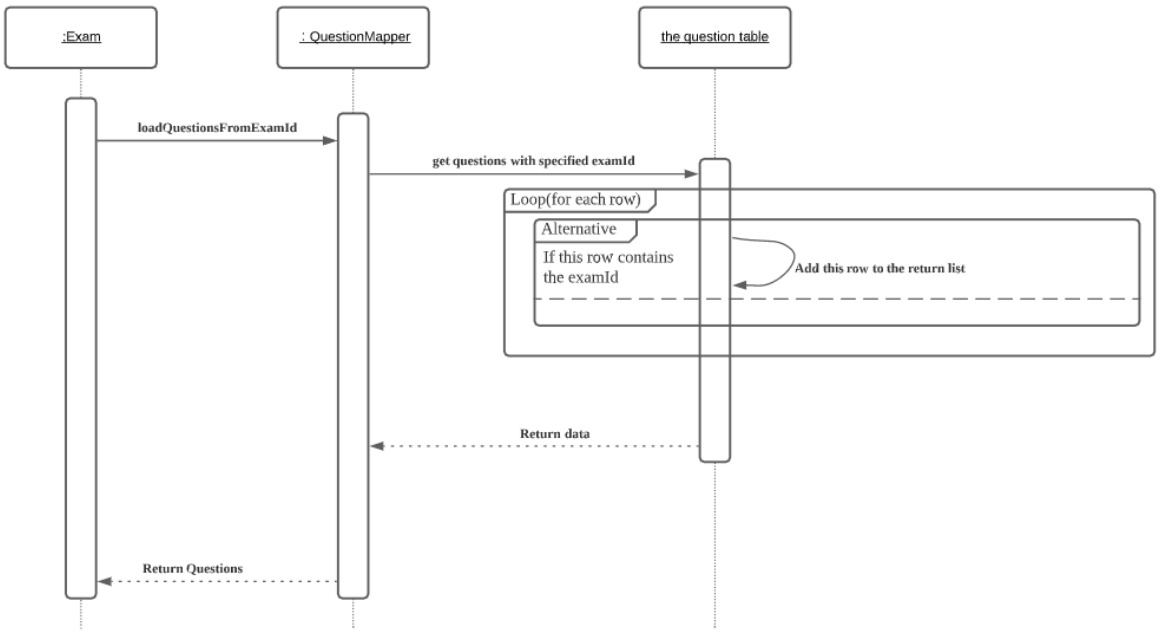
Pattern	Reason
Foreign key map	For many to many relationships, the association table mapping pattern creates a separate table to store foreign keys from both objects, which can mechanically map the relationships between the domain layer and the database.

Implementation:

We use foreign key mapping patterns whenever there is a one-to-many relationship between objects. For instance, each exam can have multiple question(s), but each question can only belong to one exam. Therefore, an examId foreign key is stored in the question database, so we can always map the question to the exam correctly.

Example of Use:

When an exam is trying to retrieve its questions from the question table, it sends a request to the question mapper with its examId. The question mapper finds all the rows containing the foreign key examId, and then returns the data to the exam object.



3.5 < Embedded Value Pattern >

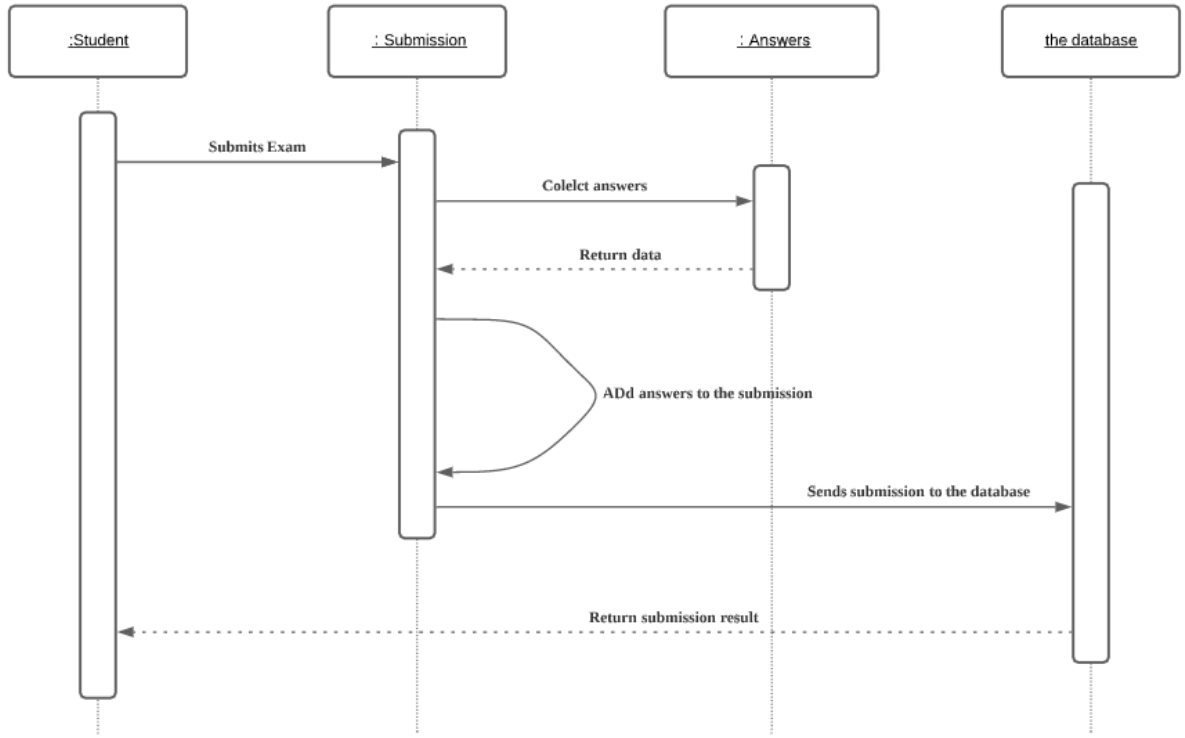
Pattern	Reason
Embedded Value	For a one-to-one relationship, the embedded value pattern can be used to map one object's values into another object's table. This is especially useful when the domain object is small. Therefore, we can have a much neater table in the database.

Implementation:

We use this pattern when the relationship between two objects is one to one. For example, each exam has only one submission, and each submission only belongs to one exam. Instead of creating two separate tables for each object, the embedded value pattern maps the values of the submission into the fields of the exam.

Example of Use:

When a student submits his/her exam, the exam object collects all the answers from the submission object. The values of the submission are embedded in the exam, and then sent to the database. Eventually, the student would receive a confirmation message if the submission went successful.



3.6 < Single table inheritance Pattern >

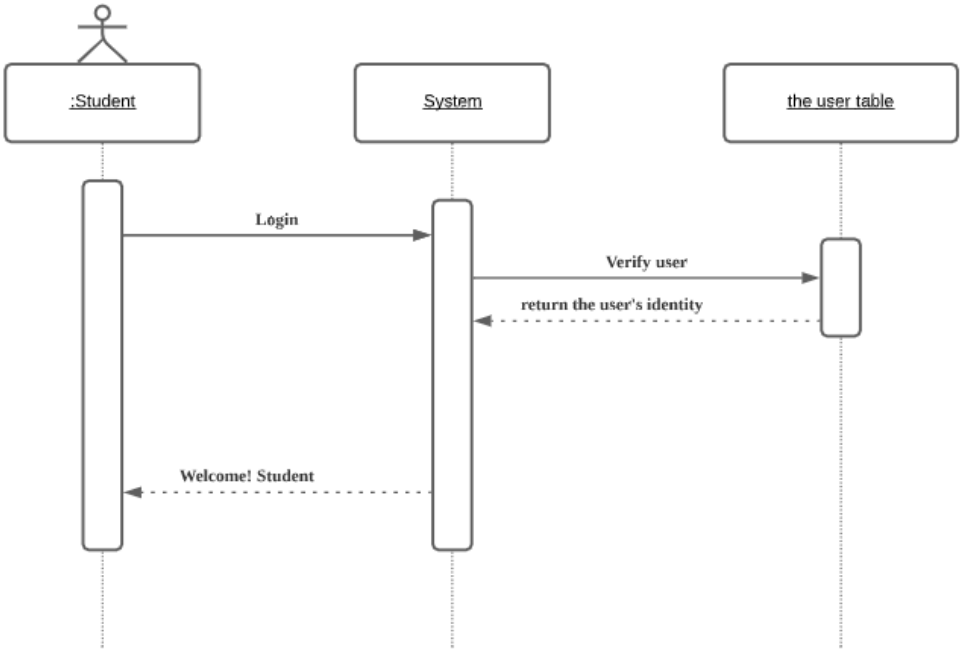
Pattern	Reason
Single table inheritance	This pattern only uses a single table to store all the inherited classes from one abstract parent class. This is useful when there are multiple similar types of one class, this pattern decreases coupling and increases neatness.

Implementation:

We have three types of users for the system: student, instructor and the administrator. We used an abstract class to record the common features such as name and userId. Moreover, we use the user_type field to distinguish each user's access level of using the system.

Example of Use:

After a user logging into the system, the system checks the type of the user. If the user type is student, the system displays the corresponding welcome message. In addition, the student has the ability to take exams, but the capability of managing exams is blocked for this kind of user type.



3.7 < Domain Model Pattern >

Pattern	Reason
Domain Model	Domain model pattern is an object oriented way to map a complex system. In this online examination system, it adds extensibility and makes the objects reusable. In addition, it makes it easier to manage the domain layer logics.

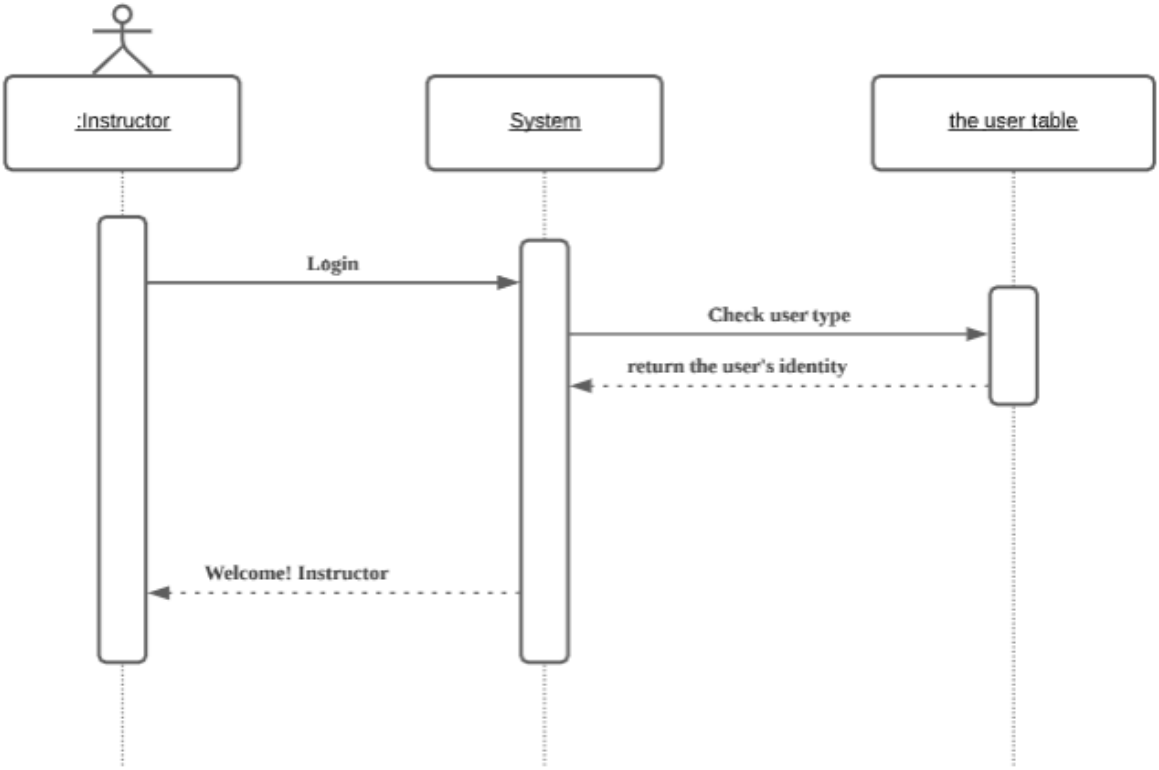
Implementation:

In this system, we have three types of users: student, instructor and admin. We created an abstract User class, and each of the three user types inherits from the user class. Most of the attributes remain the same, however, the student has an extra "isInExam" field. When calling an user class, we can only perform operations depending on the user's attributes and methods.

Example of Use & How to test:

Use case: logging in with different types of users.

If you log in the system with an instructor account, you will see an "instructor" identification on the left hand side. On the other hand, if a student account is used, the system will display a "student" identification.



3.8 < Lazy Load Pattern >

Pattern	Reason
Lazy Load	This pattern delays the loading until the point where the object needs it. Some domain objects may contain more information than what we needed, so the processing time of loading these unnecessary data can delay our program. Therefore, the lazy load pattern only loads a dummy object with on demand data initially, and continuously loads more data when required. Therefore, these patterns increase efficiency when we load an object.

Implementation:

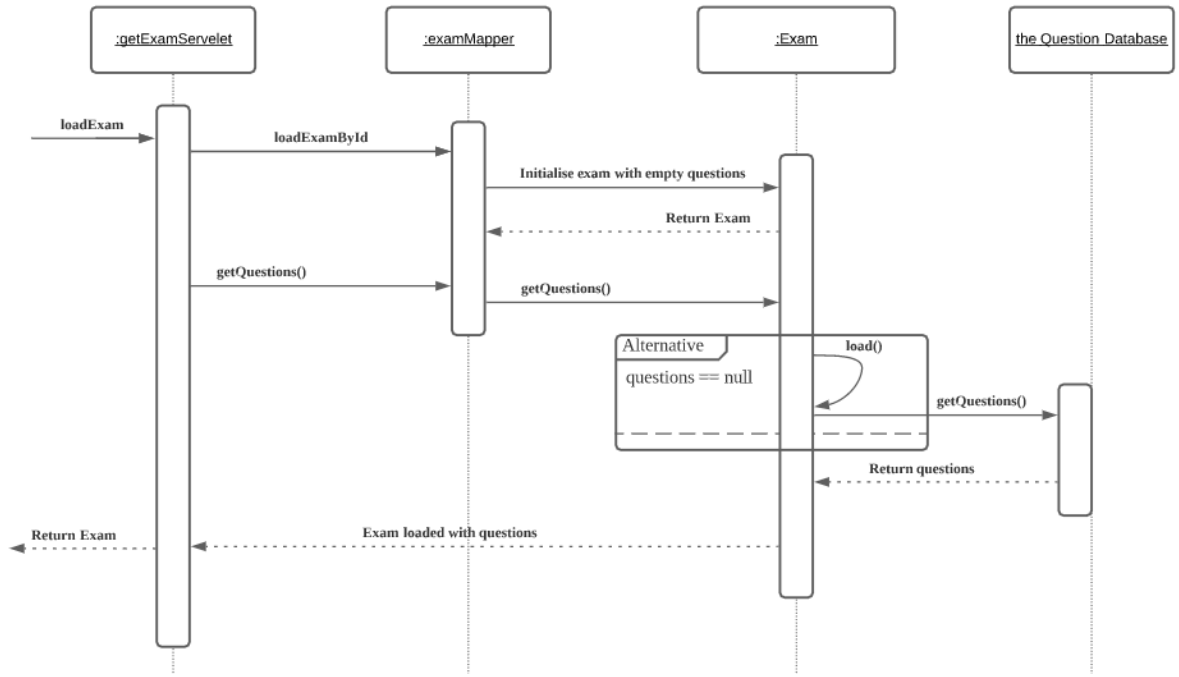
We use lazy loading whenever the domain object needs to fetch data from the database, this is evident when the "exam" class utilizes the ghost way of lazy loading data into the exam. For instance, some fields in the exam class are initialized to “null” when the class is created, and each field has a corresponding getter method that first checks whether the field is still "null". If all the fields are already loaded, it simply returns the exam object. On the other hand, if any of the fields is null, the exam object starts loading all the values from the relevant tables.

```
private void load() {
    System.out.println("Exam is reloaded");
    Exam exam = ExamMapper.loadWithId(this.id);
    if (this.description == null) {
        this.description = exam.getDescription();
    }
    if (this.questions == null) {
        this.questions = exam.getQuestions();
    }
    if (this.subjectId == null) {
        this.subjectId = exam.getSubjectId();
    }
}
```

Similarly, Answer, Question, Submission and User classes also utilize the load() method to perform lazy loading.

Example of use & How to test:

Log in with a student account and select a subject from the dashboard. When the exam starts, the system first loads an exam object with empty questions. Then the exam starts checking the values of each of its fields. Because the questions field is still null, the exam object utilizes the examMapper to load the questions from the database. Finally, exam questions are available to the student.



3.9 < Unit of Work Pattern >

Pattern	Reason
Unit of Work	In this online examination system, data exam data are being updated constantly. This pattern effectively utilizes a register to keep track of the changes. Therefore, we only need to commit minimal changes to the database. In addition The register not only

	maintains a simple way of tracking objects, but also increases the overall cohesion.

Implementation:

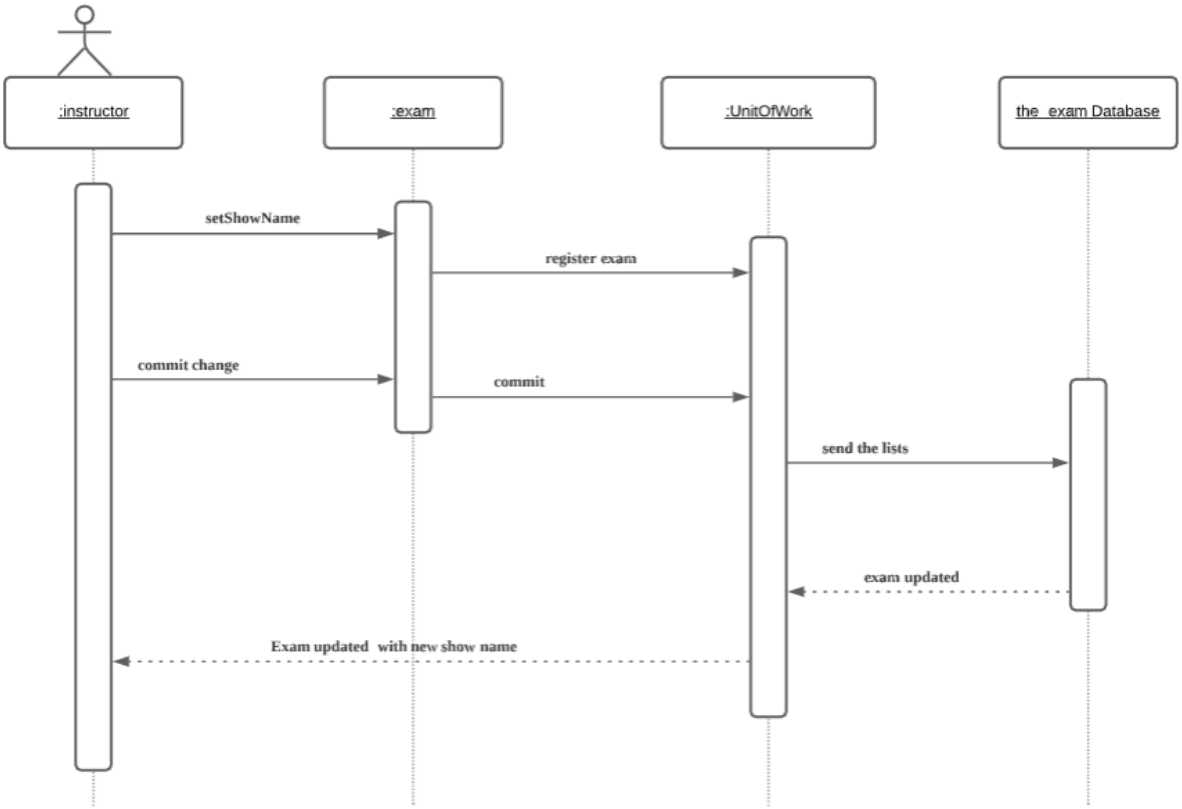
We created a UnitOfWork class to track four lists of the objects. For example, deletedObjectList is used to record deleted rows; new objects are registered into the newObjectList; dirtyObjectList records objects whose features have updated values since they were fetched from the database. Nevertheless, the remaining objects all keep the same attributes since they were fetched, so we do not need to perform any operations on them. Thus, no need to construct a separate cleanObjectList. Moreover, it's the caller's responsibility to register the object to the unit of work register. After committing, only the registered objects need to be updated in the database. Specifically, Answer, Exam, Question, Subject and User objects all used this pattern to track operations. Please see the UnitOfWork.java under the util folder for detailed implementation.

```
public class UnitOfWork {
    private static Logger logger = LogManager.getLogger(UnitOfWork.class);
    private static UnitOfWork instance;
    private ArrayList<Object> newObjectList = new ArrayList<Object>();
    private ArrayList<Object> dirtyObjectList = new ArrayList<Object>();
    private ArrayList<Object> deletedObjectList = new ArrayList<Object>();
```

Example of use & How to test:

Use case: updating the show name of an exam

Login with an instructor account and go to the subject dashboard. Click the subject that you want to update, and then choose the exam you want to modify. Press the edit button to proceed through the updating page. Furthermore, remove the current exam show name, and then type in the new show name. After clicking the save button, this exam will be added to the “dirtyObjectList”. Finally, commit the change, only the updated exam will be sent to the database, the remaining exams will be unchanged. Refresh the current page, the exam will display the newly updated show name.



4. Source Code Directories Structure

```
.
├─.github
│  └─workflows
├─docs
│  ├─deliverables
│  └─minutes
│
└─src
    ├─frontend
    ├─java
```

Heroku Link	https://online-examination-app.herokuapp.com/
Release Tag	SWEN90007_2020_Part2_A_Team
LucidChart Link to the diagrams	https://app.lucidchart.com/documents/edit/e477d18e-9290-45a3-83c9-17046ac7ee1c#?folder_id=home&browser=icon

5. Testing Instructions

Please note that:

1. the register function has not been implemented
2. If a student cannot see any exams, it is either because no exams have been created for that subject, or the instructor has not publised the exam. If it is the second case, after the instructor has published the exam, the student should go back to previous page, then go into the subject page again to be able to see the newly published exam. Refreashing the page will not work because refreashing the page will not send request to the server.

5.1 Administrator

Login Credentials:

Username	**Password**
eduardo	any password, e.g. 123

5.1.1 View all Subjects

- Log into the system using admin credentials
- Click 'Subjects' on the left menu
- All existing subjects should be shown

5.1.2 View all Instructors, Students, Exams

- Continue from the above steps
- Click any subject card you want to view
- A list of exams, instructors, and students that associated with the subject will be shown
- The status of exams (published, closed) can also be shown, but admin cannot change the status

5.1.3 Add new Subject

- Continue from the above steps
- Click 'Subjects' on the left-side menu bar or on the top breadcrumbs link
- Scroll down to the end of the subject list
- Click 'Add Subject' button
- Enter information of the new subject, including assign instructors and students to the subject
- Create subject

5.2 Instructor

Login Credentials:

We have three predefined instructors in the database, they teach different subjects, feel free to choose which instructor to login:

Username	Password
maria	any password, e.g. 123
toby	any password, e.g. 123
ben	any password, e.g. 123

5.2.1 View Subjects

- Log into the system using instructor credentials
- Click 'Subjects' on the left menu
- All subjects that the instructor teaches should be shown

5.2.2 View Exams

- Continue from above step
- Click the subject you wish to view
- A list of exams will be shown
- Click any exam to enter into a detailed view of the questions, but instructor cannot answer the questions

5.2.3 Create Exam

- Click the subject on the top breadcrumb link to go back to the list of exams page
- Click create exam button to enter into exam form page
- Enter exam information and add questions, click submit button to create the exam
- By default, the newly created exam is unpublished, which means it is not available to students
- Since you have not published the exam yet, so you cannot close the exam yet

5.2.4 Update Exam

- Click edit button of an exam and goes to the exam form
- Modify and update the existing exam information and questions
- Click submit button to update the exam

5.2.5 Delete Exam

- Click delete button of an exam

5.2.6 Publish Exam

- Click publish button of an exam
- If published, the exam cannot be unpublished, it can only be closed

5.2.7 Close Exam

- Click close button of an exam

- All students who enrolled in this subject and have not submitted by themselves will be forced to submit

5.2.8 Mark Exam

- Click mark button of an exam
- A list of all submissions of that exam will be available
- In the table view, can directly assign total mark for each submission
- Click any submission to enter into detailed view
- Assign mark to each question, the total mark can be updated automatically
- Click save marks and goes back to the main page
- Click submit all marks to store the marks into database

5.3 Student

Login Credentials:

We have six predefined students in the database, they have enrolled in different subjects, feel free to choose which student to login:

Username	**Password**
zhazz	any password, e.g. 123
tony	any password, e.g. 123
ash	any password, e.g. 123
mark	any password, e.g. 123
steve	any password, e.g. 123
james	any password, e.g. 123

5.3.1 View Subjects

- Login using student credentials
- Click subjects button on the left-side menu bar
- All subjects that the student enrolls will be shown

5.3.2 View Exams

- Click the subject you wish to view
- A list of available exams will be shown, unpublished exam will not be shown
- If an instructor publishes the exam, the student please goes back to the previous page (subject main page), then clicks the subject again to see the newly published exam. Simply refreshing the page will not work.

5.3.3 Submit Exam

- Click any available exam to take the exam. If the exam has closed, or the student has attempted the exam, the system will refuse the student to enter the exam
- Question will be presented one by one, student can go back to update their question
- Click submit button to submit the exam
- If the exam has been closed, then the system will refuse the student to submit