

Architecture Document (Continue)

**SWEN90007 Software Design and Architecture
Semester 2, 2020**

Team – Super Girls

1049113 Jianjing Yao (jianjingy)
jianjingy@student.unimelb.edu.au

1054195 Lu Wang (lw2)
lu.wang4@student.unimelb.edu.au

1095044 XuelingLiu (xuelingl1)
xuelingl1@student.unimelb.edu.au



SCHOOL OF
**COMPUTING &
INFORMATION
SYSTEMS**

Revision History

Date	Version	Description	Author
19/10/2020	01.00-D01	Design the content of this document	Jianjing Yao
21/10/2020	01.00-D02	Add Section 1 details to the document	Jianjing Yao
22/10/2020	01.00-D03	Add Section 2.1 details to the document	Jianjing Yao
23/10/2020	01.00-D04	Add Section 2.2 details to the document	Jianjing Yao
24/10/2020	01.00-D05	Add Section 2.2.1-2.2.2 details to the document	Jianjing Yao Xueling Liu
25/10/2020	01.00-D06	Add references to the document	Jianjing Yao
26/10/2020	01.00-D07	Review and discuss details in completed parts	Jianjing Yao Xueling Liu Lu Wang
27/09/2020	01.00-D08	Add Section 2.1.1 and Section 2.2.3 details to the document	Jianjing Yao Xueling Liu
28/10/2020	01.00-D09	Add class diagram and sequence diagrams	Xueling Liu Jianjing Yao
29/10/2020	01.00	Add user manual and appendix First version of the document	Xueling Liu Jianjing Yao
30/10/2020	02.00-D01	Update class diagram and sequence diagrams	Xueling Liu Jianjing Yao
31/10/2020	02.00-D02	Check grammar and adjust some details Review the document and add git tag Slightly change the diagrams	Xueling Liu Jianjing Yao Lu Wang
01/11/2020	02.00	Final version of this document	Xueling Liu Jianjing Yao Lu Wang

Table of Contents

1. Introduction	1
1.1 Project Overview	1
1.2 Target Users.....	1
1.3 Conventions, Terms and Abbreviations.....	1
2. Architectural Patterns.....	2
2.1 Concurrency	2
2.1.1 Offline Concurrency.....	2
2.1.2 Problems in Concurrency.....	2
2.1.3 Patterns in Offline Concurrency.....	3
2.1.4 Design Rationale and Implementation Details.....	4
2.2 Security.....	7
2.2.1 Authentication Enforcer.....	7
2.2.2 Authorisation Enforcer.....	8
2.2.3 Secure Pipe.....	8
3. Architectural Representation	8
3.1 Logical View	9
3.1.1 Class Diagram	9
3.2 Process View	15
3.2.1 Sequence Diagram.....	15
4. User Manual.....	19
4.1 Log in as an Instructor	19
4.2 Log in as an Administrator	20
5. Appendix	20
5.1 Link to the Application.....	20
5.2 Github Link	20
5.3 Git Release Tag.....	20
6. References	20

1. Introduction

1.1 Project Overview

This project is about an online examination application. The aim is to manage the exams online from the perspective of students and instructors. Based on the requirements given by the teaching team of SWEN90007, students can take exams of associated subjects and check results. Instructors can manage exams, including creating, editing, publishing and marking exams.

This document is designed to provide a comprehensive overview of the architecture (especially concurrency and security) of this project, which serves as a communication medium between the software architect and the team members regarding architecturally significant decisions which have been made on the project.

In this document, target users and conventions are introduced in section 1. Architectural patterns are listed in section 2. Architectural representation (class diagram and sequence diagrams) are introduced in section 3. User manual which guides the end users to access the system is shown in section 4, followed by the appendix including Github link, tag and Heroku link.

1.2 Target Users

This document is designed for team Super Girls, teaching team of SWEN90007, and potential end users who may use this online exam application in the learning management system (LMS).

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
Architectural Pattern	An architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture. A basic set of architectural patterns is important for the implementation of the project. [1]
Concurrency	Concurrency is the ability of different parts or units of a program, algorithm, or problem to be executed out-of-order or in partial order, without affecting the final outcome. [2]
Security	Security, as part of the software development process, is an ongoing process involving people and practices, and ensures application confidentiality, integrity, and availability. [3]
Architectural Representation	Architectural representation describes the adopted model in the system. “4+1” framework is always used as a base. [4]
User Manual	User manual is intended to tell people how to use the system in a correct way. [5]

2. Architectural Patterns

2.1 Concurrency

Our online exam application needs to support concurrent access to the data in data source layer, which means that multiple processes within the system will be trying to access the same data at the same time. If all of the processes are trying to read the data only, then this is not such an issue, but if some are trying to write, then problems can occur. [6]

2.1.1 Offline Concurrency

First, we need to distinguish between system transaction and business transaction. [6]

- System transaction – transactions supported by transactional resources. E.g., a database transaction (a group of SQL commands delimited by instructions to begin and end it).
- Business transaction – those at the enterprise level that may bring together several system transactions.

System transactions can be categorised based on their relationship to the business transactions that they implement. [6]

- Request transaction – a transaction that spans exactly one request to the system.
- Long transaction – a transaction that spans two or more requests to the system.
- Late transaction – a transaction that does as much reading as possible, calculates what changes are required, and then does an update as late as possible.

Business transactions often take multiple requests to complete, which means we need to use a single system transaction to implement one results in a long transaction. Using a long transaction means that we can avoid a lot of awkward problems. However, most transaction systems don't work very efficiently with long transactions. The application won't be scalable because long transactions will turn the database into a major bottleneck. In addition, the refactoring from long to short transactions is both complex and not well understood. [7]

For this reason, many enterprise applications cannot risk long transactions. In this case, we need to break the business transaction down into a series of short transactions. This means that we control concurrency for business transactions that span over multiple system transactions [6], which raises a problem called offline concurrency. Whenever the business transaction interacts with a transactional resource, such as a database, that interaction will execute within a system transaction in order to maintain the integrity of that resource. [7]

In the following sections, we will focus on offline concurrency on the server side of our online exam application, which means we need to handle multiple requests from many users that want to access the same data on the data layer.

2.1.2 Problems in Concurrency

There are a lot of different kinds of problems related to concurrency. The most common ones are lost update (interference) and inconsistent read.

- Lost update – Alice starts to update a file (Version 1), and then Bob starts to update (Version 1). Bob finishes early and clicks "Save" (Version 2). Then Alice finishes and clicks "Save" (Version 3). As is shown in Figure 1, Version 2 is lost forever.

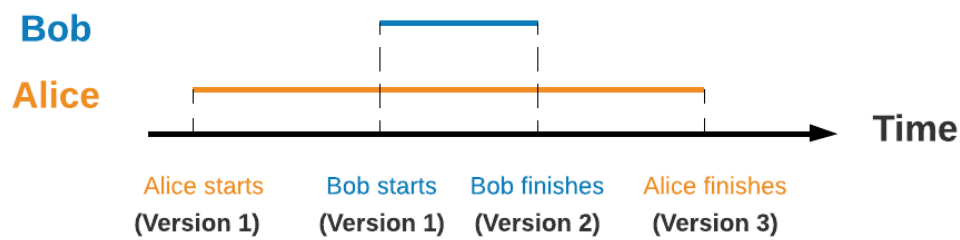


Figure 1 Lost Update

- Inconsistent read – Alice starts to read the value of x. Then Bob updates both x and y. Alice reads the new value of y, and calculates the sum of x and y. Figure 2 shows that 15 is not a correct answer, whereas the answer can be 12 (before updating) or 17 (after updating).

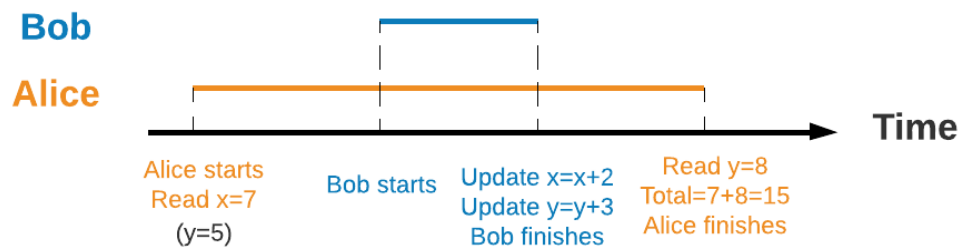


Figure 2 Inconsistent Read

2.1.3 Patterns in Offline Concurrency

In order to solve the problems in concurrency, there are three types of patterns to manage issues in offline concurrency, namely optimistic offline lock, pessimistic offline lock and implicit lock. Table 1 shows the basic description of these three patterns [6].

Pattern	Basic Description
Optimistic Offline Lock	Allows multiple transactions to access shared data simultaneously
Pessimistic Offline Lock	Allows only one business transaction at a time to operate on the data
Implicit Lock	Allows a framework or layer supertype code to acquire offline locks

Table 1 Offline Concurrency Patterns

In our project, we will mainly consider the first two patterns, namely Optimistic Offline Lock and Pessimistic Offline Lock. Table 2 shows the detailed differences between them.

Aspect	Optimistic Offline Lock	Pessimistic Offline Lock
Conflict	Conflicts are detected at commit time	Conflicts are avoided
Implementation	1. Associate a version number with each record in the database.	1. Lock the ID, or primary key. Obtain the lock before the object is loaded from the database.

	2. compare the version of the record when it was read with the current version in the database.	2. Release the lock when the business transaction finishes.
Pros	<ul style="list-style-type: none"> • Good liveness • Simplicity 	No work/data loss
Cons	Lost work/data	<ul style="list-style-type: none"> • Low liveness • Harder to program
When to use	<ul style="list-style-type: none"> • The probability of conflict between concurrent transactions is low • Loosing work/data is not a big deal 	<ul style="list-style-type: none"> • The probability of conflict between concurrent transactions is high • Loosing work/data is painful for users

Table 2 Comparison between Optimistic and Pessimistic Offline Lock

Moreover, there are three types of Pessimistic Offline Lock which is shown in Table 3.

Type of Lock	Property	Liveness	Correctness	Complexity
Exclusive write lock	Lock to edit data	Good	Inconsistent Reads	Easy
Exclusive read lock	Lock to load data (either to read or edit)	Bad	Yes	Easy
Read/write lock	<ul style="list-style-type: none"> • Lock to read, lock to write • Multiple read locks & single write lock • Read locks and write lock are mutually exclusive 	Good	Yes	Hard

Table 3 Different Types of Pessimistic Offline Lock

It is important to choose the most appropriate type of Pessimistic Offline Lock according to the actual situation. We need to follow some basic instructions shown below. [6]

- Maximize liveness
- Meeting business needs
- Consider correctness
- Minimize code complexity

2.1.4 Design Rationale and Implementation Details

In our project, we need to deal with the concurrency issues. Table 4 shows all situations of the concurrency issues in our project.

Situation	Description
A	Multiple instructors can create exams for the same subject simultaneously.
B	Multiple instructors can update the same exam simultaneously.
C	Multiple instructors can enter marks for the same student cohort simultaneously (in the table view).

D	Multiple instructors can enter marks for the same exam and the same student (in the detailed view) simultaneously.
----------	--

Table 4 Concurrency issues in our project


For Situation A, when two or more instructors are creating a new exam for the same subject, they will all add a new record to the “Exam” table associated with some records to the “Question” table in the database. During this process, there is no conflicts between them. Let’s imagine that the two instructors have created two exams with the same exam title. They can see both of the exams and discuss whose exam can remain in the system.

For Situation B, C and D, we choose to use pessimistic offline lock. The reasons for this are shown below.

- **The probability of conflict between concurrent transactions is high**
There are a lot of students enrolled in a subject, and it is not impossible that only one instructor is responsible a subject. It is more likely that two or more instructors handle different parts of a subject. Therefore, editing exams and marking submissions are common for instructors, which increased the probability of conflict.
- **Loosing work/data is painful for users**
Let’s imagine that an instructor Bob has spent several hours to think about the questions when he is editing an exam. When he wants to save that exam, he is told not to save it successfully. This is painful for instructors, and it wastes the instructors’ time. Marking the submission in detailed view is similar. An instructor Alice spent some time to read the answers of the submission and marked each answer. It will be painful if she cannot save those marks at the end.


Pessimistic Offline Lock pattern has three types of lock. We choose to use the Exclusive Write Lock to deal with Situation B, C and D. To make this decision, we have considered several factors which are shown in Table 5.

Factor	Priority*	Reason in Situation B, C and D
Business need (Requirement)	Must consider	<ul style="list-style-type: none"> • Viewing exams and editing exams are two use cases, which means read and write operation cannot affect each other. • Viewing marks and entering marks (in table view and detailed view) are separate use cases, so read and write operation cannot affect each other.
Liveness	Must consider	Exclusive Write Lock only locks to edit data, which gives much freedom when instructors read the exams or marks.
UI design	Should Consider	<ul style="list-style-type: none"> • For our UI design, we first list all exams of a subject. If an instructor chooses an exam, he will see all questions in that exam. He can choose to edit that exam or change the status of that exam. If he wants to edit it, he can click “EDIT” on the top.

		<ul style="list-style-type: none"> For the UI of the table view, if an instructor wants to enter a mark for a submission, he needs to click a  symbol and enter the mark. For the UI of the detailed view, if an instructor wants to enter marks for a submission, he needs to click “MARK” and enter the marks.
Correctness	Should Consider	The correctness of Exclusive Write Lock is not perfect, which means sometimes it will cause inconsistent read.
Complexity	Could Consider	The code complexity of Exclusive Write Lock is easy.

*Priority - Must > Should > Could

Table 5 Reasons for choosing Exclusive Write Lock

For the implementation details, we assign a lock when instructors update exams (click “EDIT”) or enter marks (click the  symbol in table view or “MARK” in detailed view). All locks are stored in an additional table in our database. “Lock” table includes the following attributes.

- lockid – primary key of this table
- tablename – which table is locked
- id – which record (row) in that table is being locked
- owner – who lock that record
- time – when did the lock start

When the instructor finishes editing the exam or entering the mark (click “Save”), or he chooses to cancel the operation (click “Cancel”), the lock is released, which means the corresponding row in “Lock” table is deleted. Moreover, if an exam is being updated by an instructor, then all instructors who also want to edit the same exam at the same time will get a prompt which says that exam is hold by other instructor. Entering marks in both table view and detailed view is similar to editing exams.

As we choose to use the pessimistic lock pattern, it is important for us to choose an appropriate grain for locking (i.e. which table to be locked). For example, when editing an exam, we can choose to lock the exam or the exact question that is being edited. Table 6 shows the reasons for choosing the grain in each situation.

Situation	Grain	Reason
B	Exam	<ul style="list-style-type: none"> Editing exam includes editing the title of the exam. However, the “title” is stored in “Exam” table, so we need to lock the “Exam”. For the UI design, we did not provide a button which can save each question when editing an exam. There is only one “Save” button at the end of the edit page.
C	Submission	<ul style="list-style-type: none"> The attribute “total mark” is in the “Submission” table, so we lock the “Submission” table when the instructor enters a mark in table view.

D	Submission	<ul style="list-style-type: none"> For the UI design, we did not provide a button which can save each answer's mark when entering marks in detailed view. There is only one "Save" button at the end of the mark page. For the same submission, there will be some conflicts if different instructors can edit it in table view and detailed view at the same time. This means if an instructor is entering the mark in the table view, the detailed view of marking will also be locked and vice versa.
----------	------------	--

Table 6 Grains and Reasons when locking

We also consider the problem of deadlock. Let's imagine the situation that an instructor starts to edit an exam, but he closes the browser directly neither clicking "Save" nor clicking "Cancel". Will that exam be locked forever? The answer is no because we record the time of starting the lock, and we set that every lock is released after a fixed period (e.g. 2 hours). In this way, no record will be locked forever, and we can avoid the deadlock problem. Moreover, when an instructor chooses to log out (click "Log out"), we release all locks hold by that instructor, which can also help to avoid the deadlock problem.

2.2 Security

Security is a type of non-functional requirement. It is not possible to simply measure non-functional qualities after a system has been built, and then retrospectively fit those qualities in should the system fail to meet them. [6] In this section, we will describe three aspects related to security to ensure data integrity and confidentiality.

2.2.1 Authentication Enforcer

The authentication enforcer pattern specifies a centralised authentication mechanism, which encapsulates the detailed information of all users who handle all authentication operations at the presentation layer. Specifically, the enforcer needs to verify whether each request comes from an authenticated entity and ensure users are indeed who they say they are.

In our project, authentication mechanism consists of these two pieces of information. For the unique and public piece that establishes a statement of user identity, we use a *username*. For the private piece that verifies the identity statement, we use a *password*. To implement this pattern, there are four strategies, namely authenticated strategy, third-party provider strategy, an external library and the proprietary approach. Our team chooses to use the last strategy, which is to implement the authentication enforcer pattern by ourselves.

We introduce a concept of "token" in this section. For example, Bob is an end user of the system. Once Bob clicks the "SIGN IN" button, the data of username and password entered by Bob will be sent to back-end. Back-end will first confirm whether Bob is in the database. Then back-end will check the password provided by Bob is the same as the one stored in the database. If the password is correct, back-end will generate a token for Bob and send this token back to the front-end. Then all requests sent from Bob will be associated with this token* (store the token into the header of HTTP requests). In addition, back-end will create a map (key-value = user object-token) to record all valid tokens. We can further deal with the requests from all authenticated users (users who have a valid token).

*Note – Each token is stored in the local storage of a web browser, which means the first account will be replaced if we log in as two different accounts in the same web browser.

2.2.2 Authorisation Enforcer

The authorisation enforcer pattern provides a centralized point that encapsulates authorisation mechanism by defining a standard way to control access to specific functions or data in the system. It ensures that users can only access the resources that they are allowed to.

In our project, we have three types of users, namely administrator, instructor and student. Each of them is authorized to have different rights in the system. For the admin, he or she can create new subjects, add users (instructors or students) to the system and assign users to each subject. For instructors, they can manage an exam including creating, updating and marking exams. For students, they can take an exam once, and view their marks when the result of the exam is released.

When a user wants to do an operation in the system, the HTTP request is associated with a token. If the token is valid, back-end can get the user type from the token and authorise users with different functions. The basic idea is to use the token to direct different types of users to different pages. In addition, token can be used to check whether the user has the right to access each request. For example, if Alice is a student, and she wants to create an exam for a subject (instructor's right). This request should be denied. People may confuse how this can happen if Alice can only see the pages designed for students after she logs in. Actually, this can happen when using some testing tool (e.g. Postman) as we use the REST API to implement back-end.

2.2.3 Secure Pipe

Systems that send data over a network are open to security vulnerabilities, such as eavesdropping and spoofing. This is a problem that many Internet-based applications face. Applications that send sensitive data will need to address these vulnerabilities by sending the sensitive data over a secure network. The secure pipe pattern specifies a generic and standardised way to send requests over a secure network using encryption and decryption. The pattern is not application specific, which reduces the complexity of the solution, and also allows instances of it to be reused over multiple applications. [6]

In our project, the sensitive information is the users' password. We want to put all transmissions of passwords into a secure pipe. To achieve this, we apply the Advanced Encryption Standard (AES) method. By using AES, we encrypt the password provided by users in front-end, and sent the encrypted one to back-end. Back-end can use the corresponding method to decrypt it and compare the decrypted password with the one stored in database. In this way, we can ensure the integrity and privacy of passwords sent between front-end and back-end. In addition, the deployment of Heroku provides the HTTPS for us. HTTPS is used for secure communication over a computer network. In HTTPS, the communication is encrypted using Transport Layer Security (TLS) or, formerly, Secure Sockets Layer (SSL). [8] In this way, it double guarantee the secure pipe when we send the passwords from front-end to back-end.

3. Architectural Representation

Architectural representation describes the adopted model in the system. "4+1" framework is always used as a base. In this section, we include the updated views, namely logical view and process view. The updated class diagram and the sequence diagrams related to concurrency and security are shown in this section.

3.1 Logical View

The logical view is an object-oriented decomposition. The viewer of logical view is the end-users. It considers more on functional requirements, especially what the system should provide in terms of services to its users. This view shows the components (objects) of the system as well as their interactions and relationships.

3.1.1 Class Diagram

In software engineering, a class diagram in the Unified Modelling 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.

The class diagram is the main building block of object-oriented modelling. It is used for general conceptual modelling of the structure of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed. [9]

As shown in Figure 3, there are mainly five kinds of relationships between classes, namely dependency, realization, aggregation, composition and inheritance.

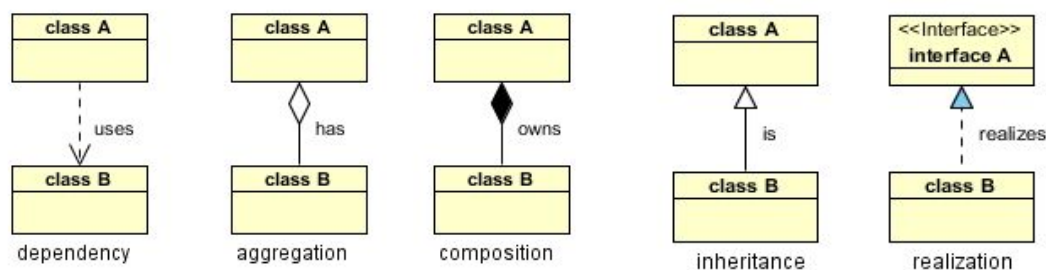


Figure 3 UML Class Diagram Relationships

Moreover, in a class, “-” represents private, “+” represents public, “#” represents protected, underscore represents static and italic represents abstract. In a method of a class, *{guarded}* means that the method uses the *synchronized* keyword in the Java code.

In our class diagram of part 3, classes with white background, black fonts and black relationship lines already exist in part 2, classes with different color backgrounds, blue fonts and blue relationship lines are what we added and modified in part 3. Specifically, blue represents package **domain**, orange represents package **mapper**, yellow represents package **service**, gray represents package **serviceImp**, dark green represents package **servlet**, and red represents package **util**.

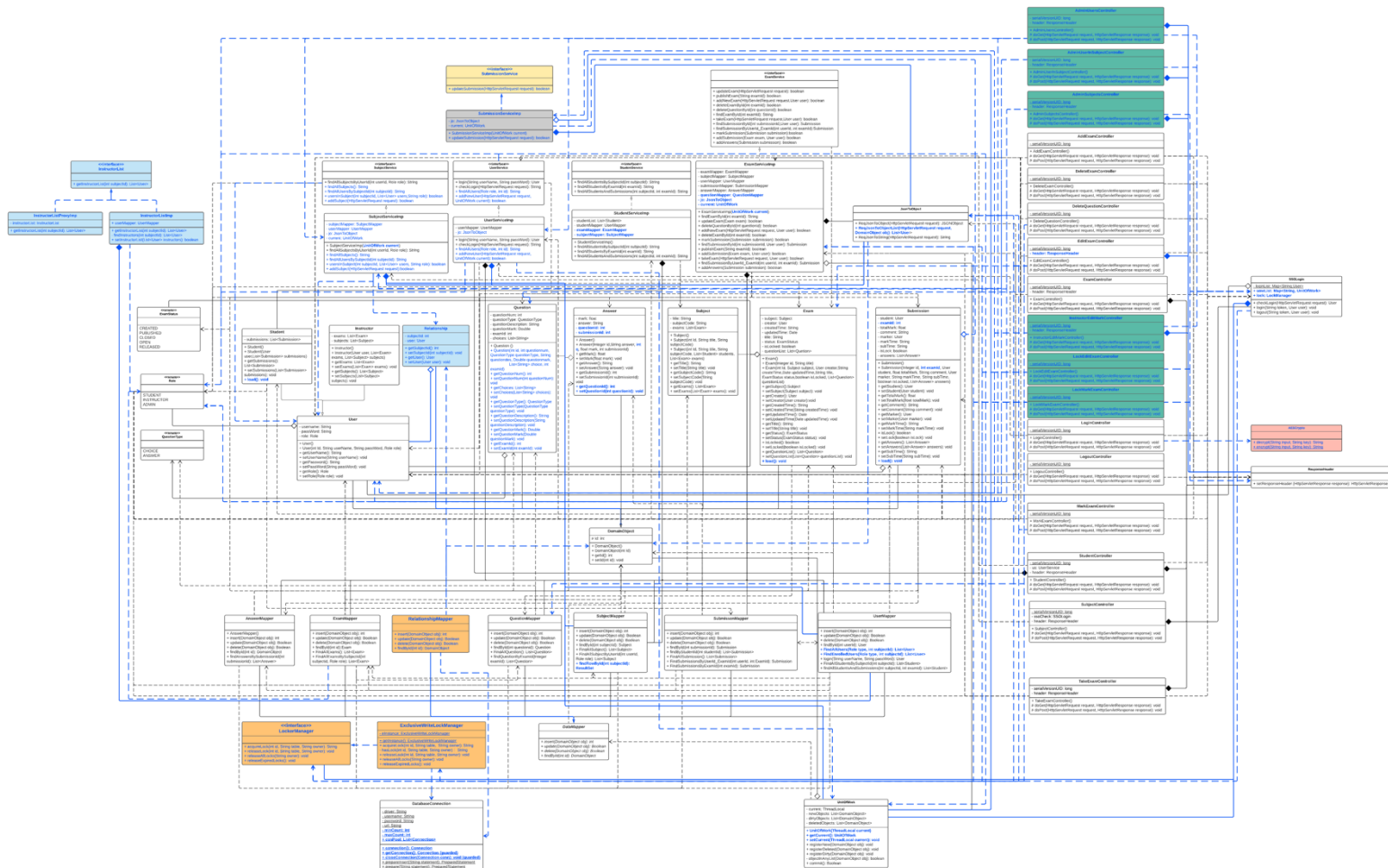


Figure 4 Class Diagram

(https://github.com/Olivia0012/SWEN90007_2020_SuperGirls/blob/master/docs/Part3_Figures/Part%203_Class%20Diagram.png)

The following series of tables show the specific relationships between classes. The blue font is the addition and modification we made in part 3.

Package	Class	Associated Class	Relationship
domain	Admin	User	Inheritance
		Role	Dependency
	Answer	DomainObject	Inheritance
		Question	Aggregation
	Exam	DomainObject	Inheritance
		Subject; User; ExamStatus	Aggregation
		QuestionMapper	Dependency
	Instructor	User	Inheritance
	Question	DomainObject	Inheritance
		QuestionType	Aggregation

	Relationship	DomainObject	Inheritance
		User	Aggregation
	Student	User	Inheritance
		Role; SubmissionMapper	Dependency
	Subject	DomainObject	Inheritance
	Submission	DomainObject	Inheritance
		User; Exam	Aggregation
		AnswerMapper	Dependency
	User	DomainObject	Inheritance
		Role	Aggregation
	InstructorListImp	InstructorList	Realization
		UserMapper	Composition
		Role	Dependency
	InstructorListProxyImp	InstructorList	Realization

Table 7-1 Package “domain”

Package	Class	Associated Class	Relationship
mapper	AnswerMapper	DataMapper	Inheritance
		DatabaseConnection; Answer; DomainObject; Question; Submission; QuestionMapper; ExamMapper; IdentityMap; UnitOfWork	Dependency
	DataMapper	DomainObject	Dependency
	ExamMapper	DataMapper	Inheritance
		DatabaseConnection; DomainObject; Exam; Question; Subject; User; ExamStatus; Role; SubjectMapper; QuestionMapper; IdentityMap; UnitOfWork	Dependency
	ExclusiveWriteLockManager	LockerManager	Realization
		DatabaseConnection	Dependency
	LockingMapper	DataMapper	Inheritance
		DomainObject	Dependency
	QuestionMapper	DataMapper	Inheritance

		DatabaseConnection; DomainObject; Question; QuestionType; IdentityMap ;	Dependency
		DataManager	Inheritance
	RelationshipMapper	DatabaseConnection; DomainObject; Relationship	Dependency
		DataManager	Inheritance
	SubjectMapper	DatabaseConnection; DomainObject; Exam; Subject; ExamMapper; Role; IdentityMap	Dependency
		DataManager	Inheritance
	SubmissionMapper	DatabaseConnection; DomainObject; Answer; Exam ; Submission; User; ExamMapper ; UserMapper; AnswerMapper; IdentityMap	Dependency
		DataManager	Inheritance
	UserMapper	DatabaseConnection; DomainObject; Exam; Student; Submission; User; SubjectMapper; SubmissionMapper; ExamMapper; Role; IdentityMap	Dependency
		DataManager	Inheritance

Table 7-2 Package “mapper”

Package	Class	Associated Class	Relationship
service	ExamService	Exam; Submission; User	Dependency
	SubjectService	Role; User	Dependency
	UserService	User; Role ; UnitOfWork	Dependency

Table 7-3 Package “service”

Package	Class	Associated Class	Relationship
serviceImp	ExamServiceImp	ExamService	Realization
		ExamMapper; SubjectMapper; UserMapper; SubmissionMapper; AnswerMapper; QuestionMapper; JsonToObject; UnitOfWork	Composition

		Answer; Exam; Question; Submission; User; ExamStatus; Role; UnitOfWork	Dependency
	StudentServiceImp	StudentService	Realization
		UserMapper; ExamMapper; SubjectMapper; Student	Composition
		Exam; Subject	Dependency
	SubjectServiceImp	SubjectService	Realization
		Relationship; Subject; User; SubjectMapper ; Role	Dependency
		SubjectMapper; UserMapper; UnitOfWork; JsonToObject	Composition
	SubmissionServiceImp	SubmissionService	Realization
		Submission	Dependency
		JsonToObject	Composition
		UnitOfWork	Aggregation
	UserServiceImp	UserService	Realization
		UserMapper; JsonToObject	Composition
		User; Role ; UnitOfWork	Dependency

Table 7-4 Package “serviceImp”

Package	Class	Associated Class	Relationship
Servlet	AddExamController	User; ExamServiceImp; ResponseHeader; SSOLogin	Dependency
	AdminSubjectsController	User; Role; SubjectServiceImp; SSOLogin	Dependency
		ResponseHeader	Composition
	AdminUserInSubjectController	User; Role; SubjectServiceImp; JsonToObject; SSOLogin	Dependency
		ResponseHeader	Composition
	AdminUsersController	User; Role; SubjectServiceImp; UserServiceImp; SSOLogin	Dependency
		ResponseHeader	Composition
	DeleteExamController	User; ExamServiceImp; ResponseHeader; SSOLogin	Dependency
	DeleteQuestionController	Question; User; ExclusiveWriteLockManager ; LockManager ; QuestionMapper;	Dependency

		ExamServiceImp; ResponseHeader; SSOLogin	
EditExamController		Exam; User; Role; ExamServiceImp; ExclusiveWriteLockManager; LockManager; JsonToObject; ResponseHeader ; SSOLogin	Dependency
		ResponseHeader	Composition
ExamController		ResponseHeader	Composition
		Exam; User; ExamServiceImp; JsonToObject; SSOLogin	Dependency
InstructorEditMarkController		User; Role; SubmissionServiceImp; SSOLogin	Dependency
		ResponseHeader	Composition
LockEditExamController		User; Role; ExclusiveWriteLockManager; LockManager; ExamServiceImp; ResponseHeader; SSOLogin	Dependency
LockMarkExamController		User; Role; ExclusiveWriteLockManager; LockManager; ExamServiceImp; ResponseHeader; SSOLogin	Dependency
LoginController		User; UserServiceImp; ResponseHeader; SSOLogin; AESCrypto	Dependency
LogoutController		User; ResponseHeader; SSOLogin	Dependency
MarkExamController		Submission; User; Role; ExclusiveWriteLockManager; LockManager; JsonToObject; ExamServiceImp; ResponseHeader; SSOLogin	Dependency
StudentController		UserService; ResponseHeader	Composition
		User; Role; UserService; StudentServiceImp; UserServiceImp; SSOLogin	Dependency
SubjectController		ResponseHeader; SSOLogin	Composition
		User; SubjectServiceImp	Dependency
TakeExamController		ResponseHeader	Composition

		Submission; User; Role; ExamServiceImp; SSOLogin	Dependency
--	--	---	------------

Table 7-5 Package “servlet”

Package	Class	Associated Class	Relationship
shared	UnitOfWork	DomainObject	Aggregation
		DataManager	Dependency

Table 7-6 Package “shared”

Package	Class	Associated Class	Relationship
util	SSOLogin	User	Aggregation
		UnitOfWork; LockManager	Composition
		ExclusiveWriteLockManager	Dependency
	JsonToObject	DomainObject; User	Dependency

Table 7-7 Package “util”

3.2 Process View

The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behaviour of the system. Process view can solve the problems of concurrency, distribution, integrators, performance and scalability. The sequence diagram is a way to achieve the goal of process view.

3.2.1 Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. Sequence diagrams describe how and in what order the objects in a system function. Sequence diagram includes some components shown below.

a. Actor

System roles can be people, other systems or subsystems. It is represented by a little icon of a person.

b. Object

The object is at the top of the sequence diagram and is represented by a rectangle.

c. Lifeline

In the sequence diagram, each object has a vertical dashed line at the bottom centre, which is the lifeline (timeline) of that object.

d. Activation

The activation represents the operation performed at a certain period of time on the object timeline in the sequence diagram. It is represented by a very narrow rectangle.

e. Message

Message represents the information sent between objects. There are two types of messages in our sequence diagrams.

- *Synchronous Message* - The sender of the message passes control to the receiver of the message and then stops the activity, waiting for the receiver of the message to return the control. It is used to represent the meaning of "synchronization". It is represented by a solid arrow in solid lines.
- *Return Message* - It is necessary to return message of procedure calls. It is represented by a normal arrow in dashed lines.

f. Combination fragment

Several types of fragments exist in sequence diagram such as alternative (if-elseif-else or if-else), option (if) and loop (while). We only use "alternative" in our sequence diagram. It is represented as "Alternative". We design sequence diagrams for each concurrency situation. Figure 5 - Figure 8 show the sequence diagrams of our project in part 3.

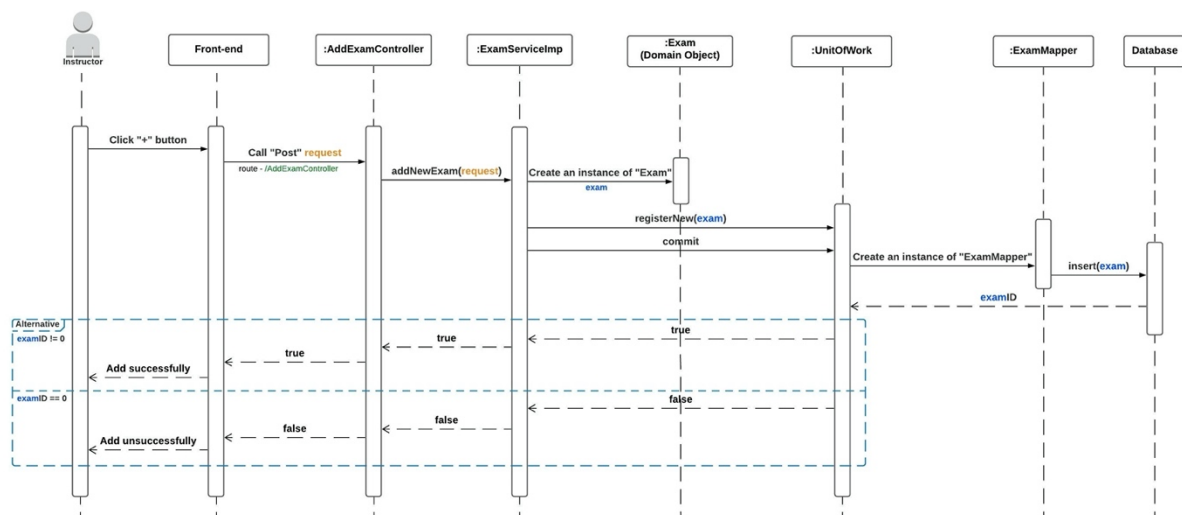


Figure 5 Sequence Diagram of "Add an exam"

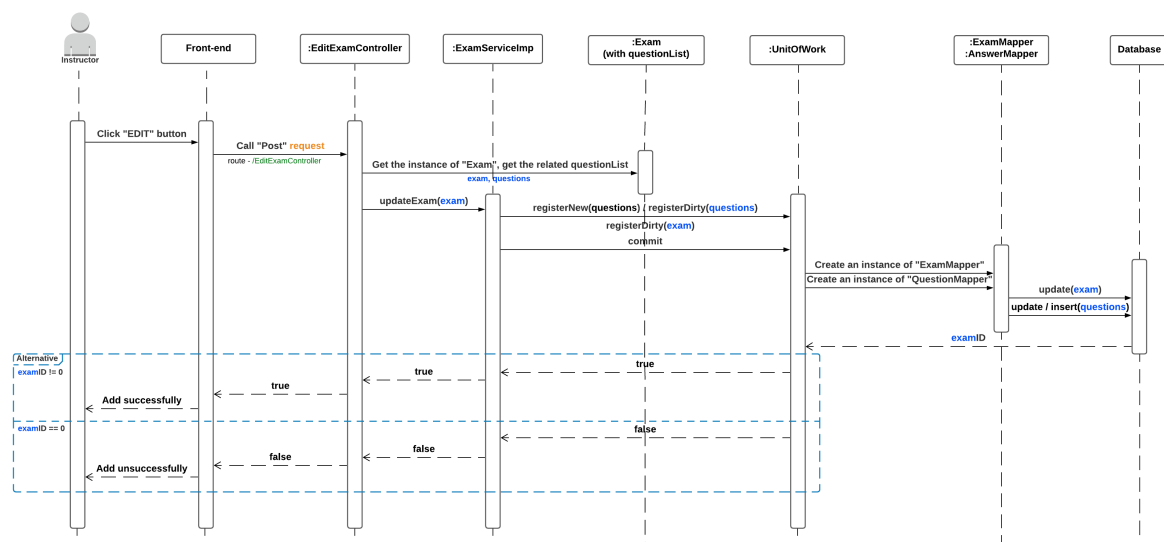


Figure 6-1 Sequence Diagram of "Edit an exam"

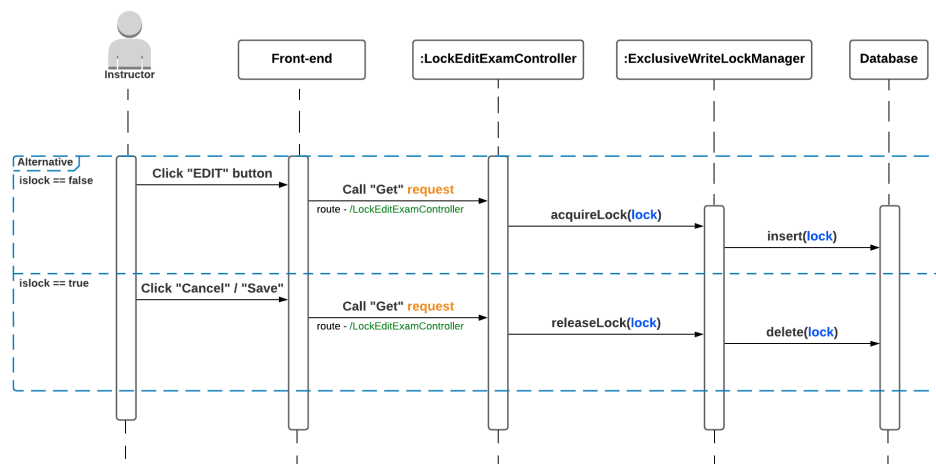


Figure 6-2 Sequence Diagram of “Edit an exam” (Concurrency)

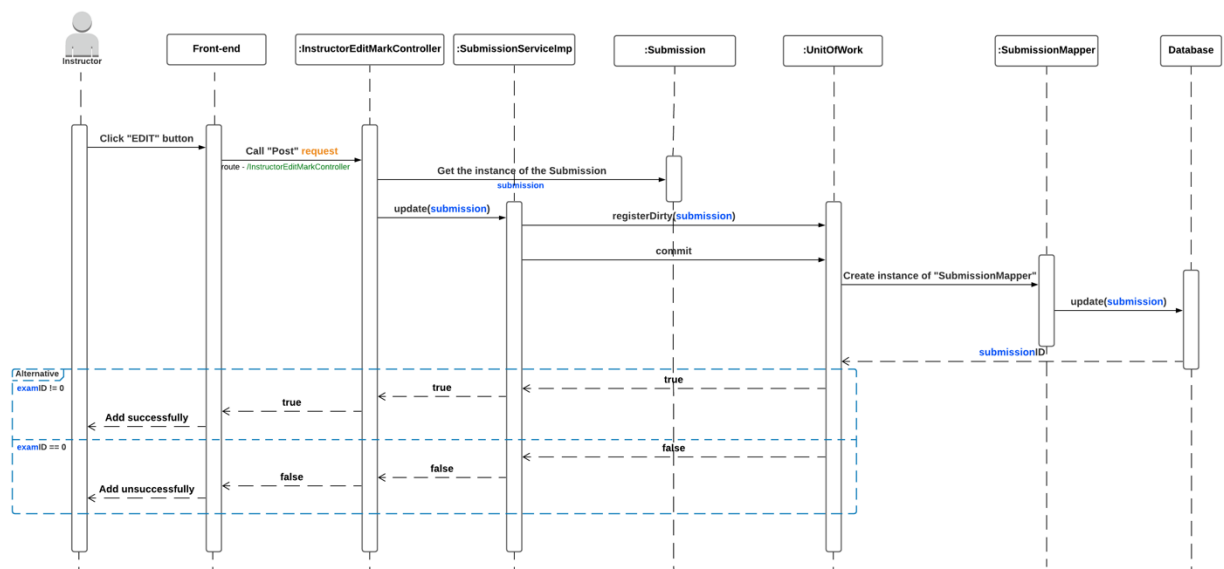


Figure 7-1 Sequence Diagram of “Edit total mark”

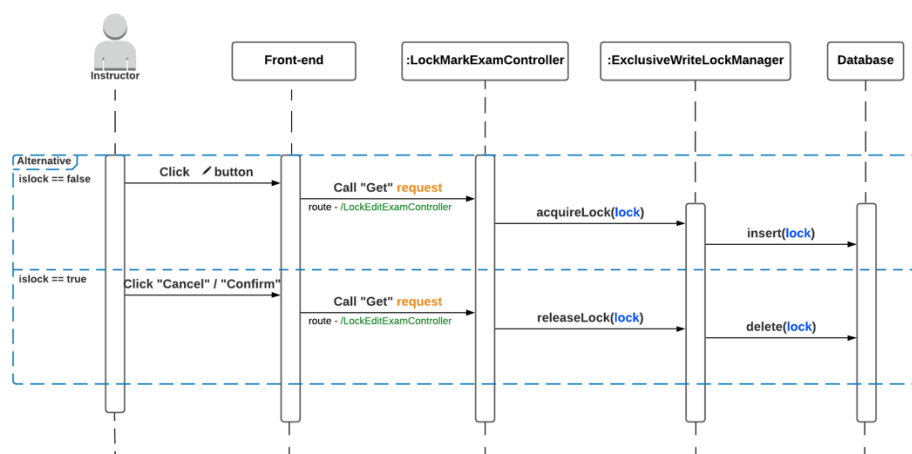


Figure 7-2 Sequence Diagram of “Edit total mark” (Concurrency)

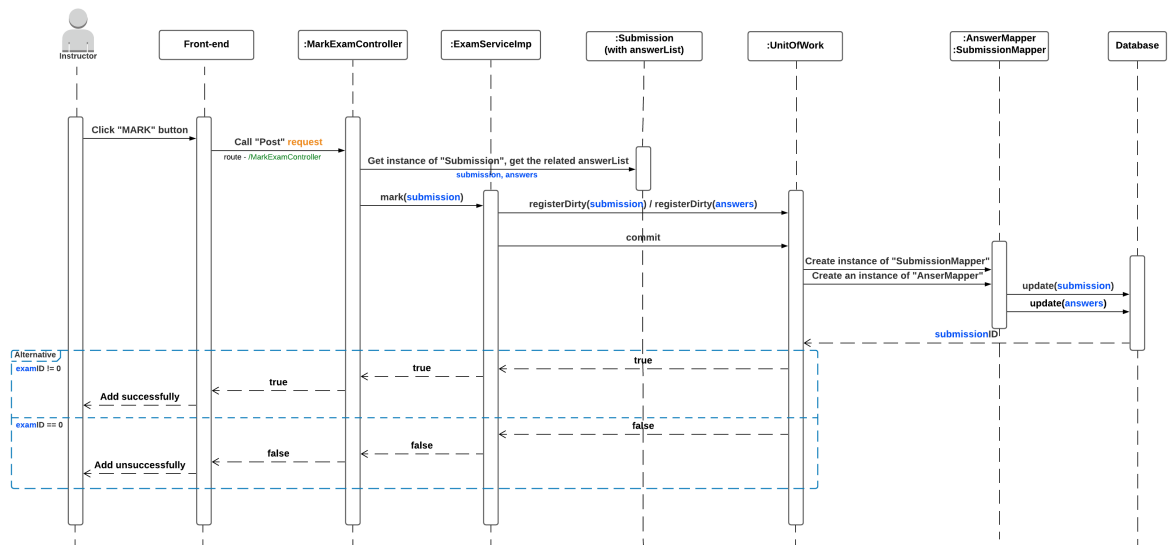


Figure 8-1 Sequence Diagram of “Mark a submission”

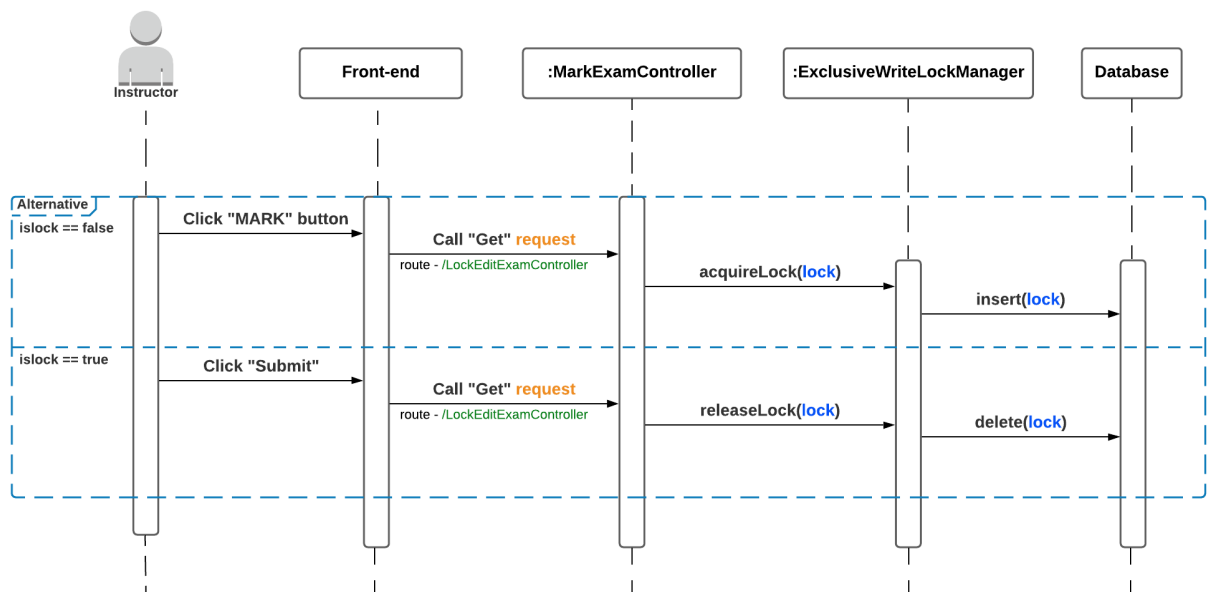


Figure 8-2 Sequence Diagram of “Mark a submission” (Concurrency)

In addition, we design a sequence diagram to illustrate the process of using Security patterns which is shown in Figure 9.

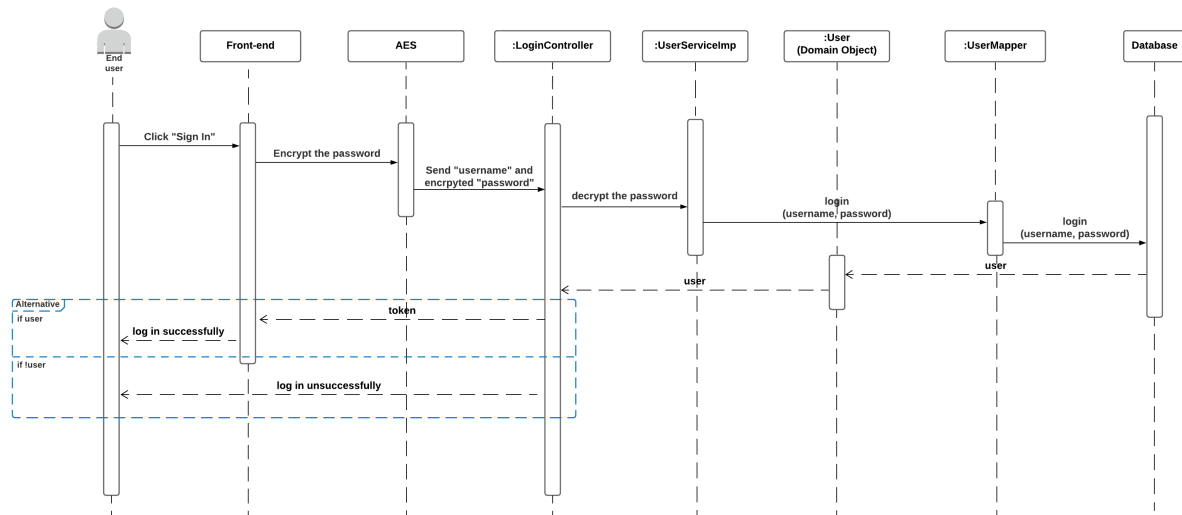


Figure 9 Sequence Diagram of Security patterns

4. User Manual

4.1 Log in as an Instructor

- To test the concurrency situations, please use **two web browsers** to open the link <https://frontend-react-test-01.herokuapp.com/>
- In one browser, enter the username *Edu* and password *111* to log in. In another browser, enter the username *Maria* and password *111* to log in. They are assigned to the same subject SWEN90007.
- Edit the same exam simultaneously.
 - Maria* chooses “Final Exam” and clicks “EDIT” button at the top right.
 - Edu* chooses “Final Exam” and clicks “EDIT” button at the top right when *Maria* is editing the final exam.
 - The system shows a pop-up window and tells *Edu* that he cannot edit this exam because another instructor is editing it.
 - When *Maria* exits the operation of editing the exam (click “Cancel” or “Save” button), *Edu* clicks the “EDIT” button again, and he can edit the exam now.
- Enter total mark for the same student simultaneously (in the table view).
 - Maria* chooses “Final Exam” and clicks “VIEW” button.
 - Maria* clicks the “pencil” icon of Susan in the table view.
 - Edu* clicks “VIEW” button of “Final Exam” and wants to enter the total mark for Susan simultaneously.
 - The system shows a pop-up window and tells *Edu* that he cannot enter the total mark for Susan because another instructor is marking it.
 - When *Maria* exits the operation of marking Susan (click “Cancel” or “Confirm” button), *Edu* clicks the “pencil” icon, he can mark Susan now.
- Enter marks for the same submission simultaneously (in the detailed view).
 - Maria* chooses “Final Exam” and clicks “VIEW” button.
 - Maria* clicks the “MARK” button of Susan in the detailed view.
 - Edu* clicks “VIEW” button of “Final Exam” and wants to mark Susan’s submission in detail simultaneously.
 - The system shows a pop-up window and tells *Edu* that he cannot mark Susan’s

submission because another instructor is marking it.

- When *Maria* exits the operation of the detailed view (click “Submit”), *Edu* clicks the “MARK” button, he can mark Susan’s submission in detail now.

4.2 Log in as an Administrator

- Enter <https://frontend-react-test-01.herokuapp.com/> in web browser.
- Enter the username *admin* and password *111* to login.
- Click the “Subject Management” on the left, the admin can manage all subjects in the system.
 - Click “ADD SUBJECT” button at the top right. Enter the “Subject Code” and “Subject Title”. Click “SUBMIT” button, the admin can add a subject to the system.
 - Click “ASSIGN” button for each subject. Choose the username need to be assigned. Click “SUBMIT” button, the admin can assign instructors and students to each subject.
- Click the “User Management” on the left, the admin can manage users in the system.
 - Click “ADD USER” button at the top right. Select “Role”, enter the “Username” and “Password”. Click “SUBMIT” button, the admin can add a user to the system.

5. Appendix

5.1 Link to the Application

<https://frontend-react-test-01.herokuapp.com/>

5.2 Github Link

https://github.com/Olivia0012/SWEN90007_2020_SuperGirls

5.3 Git Release Tag

SWEN90007_2020_Part3_SuperGirls

6. References

- [1] “Architectural pattern”, En.wikipedia.org, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Architectural_pattern. [Accessed: 23- Oct- 2020].
- [2] “Concurrency (computer science)”, En.wikipedia.org, 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Concurrency_\(computer_science\)](https://en.wikipedia.org/wiki/Concurrency_(computer_science)). [Accessed: 25- Oct- 2020].
- [3] “Software development security”, En.wikipedia.org, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Software_development_security. [Accessed: 25- Oct- 2020].
- [4] The University of Melbourne, “Architecture document”, 2020.
- [5] “User guide”, En.wikipedia.org, 2020. [Online]. Available: https://en.wikipedia.org/wiki/User_guide. [Accessed: 23- Oct- 2020].
- [6] The University of Melbourne, “Software Design and Architecture Subject Notes for SWEN90007”, 2020.
- [7] M. Fowler, Patterns of enterprise application architecture. Boston, Mass.: Addison-Wesley, 2015.
- [8] “HTTPS”, En.wikipedia.org, 2020. [Online]. Available: <https://en.wikipedia.org/wiki/HTTPS>. [Accessed: 26- Oct- 2020].
- [9] “Class diagram”, En.wikipedia.org, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Class_diagram. [Accessed: 26- Oct- 2020].