Architecture Document

SWEN90007

LMS

Team: SDA Wombat

In charge of: Haobei Ma, Yu Guo, Yifan He

SCHOOL OF COMPUTING & INFORMATION SYSTEMS

Revision History

Date	Version	Description	Author
10/08/2020	01.00-D1	Initial draft	All team members
11/08/2020	01.00-D2	Added use cases 1-4	Haobei Ma
13/08/2020	01.00-D3	Added use cases 5-9	Yifan He
13/08/2020	01.00-D4	Added use cases 10-11	Yu Guo
14/08/2020	01.00-D5	Added user case diagram	Yu Guo
14/08/2020	01.00-D6	Updated user case diagram	Yifan He
15/08/2020	01.00-D7	Rechecked user cases and added use cases 12-13	Haobei Ma
22/08/2020	01.00-D8	Review the document	All team members
22/08/2020	01.00	First version of the document	All team members
11/09/2020	0.2.00-D1	Updated document based on feedback	Haobei Ma
14/09/2020	02.00-D2	Added Logical View	Haobei Ma
17/09/2020	02.00-D3	Added Process View	Haobei Ma
19/09/2020	02.00-D4	Added Development View	Yifan He
25/09/2020	02.00-D5	Added Physical View	Yu Guo
01/10/2020	02.00-D6	Added Scenarios	All team members
03/10/2020	02.00	Second version of the document	All team members
15/10/2020	03.00-D1	Updated document based on feedback	All team members
17/10/2020	03.00-D2	Included security patterns	Haobei Ma
21/10/2020	03.00-D3	Updated class diagram	Haobei Ma
24/10/2020	03.00-D4	Included concurrency	All team members
26/10/2020	03.00-D5	Review the document	All team members
29/10/2020	03.00	Third version of the document	All team members

Contents

■ Introduction

This document specifies the system's architecture LMS, describing its main standards, module, components, *frameworks* and integrations.

1.1 Proposal

The purpose of this document is to give, in high level overview, a technical solution to be followed, emphasizing the components and *frameworks* that will be reused and researched, as well as the interfaces and integration of them.

1.2 Target Users

This document is aimed at the project team, with a consolidated reference to the research and evolution of the system with the main focus on technical solutions to be followed.

■.1 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.

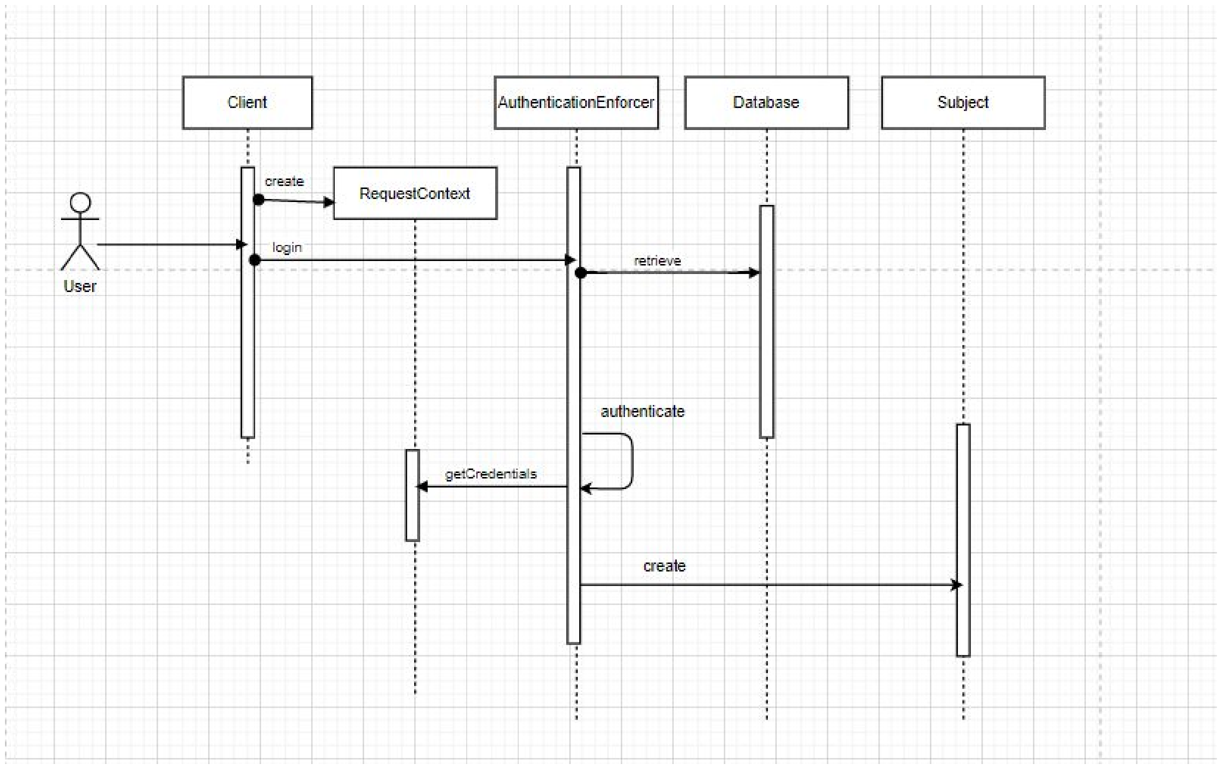
Term	Description
Component	Reusable and independent software element with well defined public interface, which encapsulates numerous functionalities and which can be easily integrated with other components.
Module	Logical grouping of functionalities to facilitate the division and understanding of software.
MCQ	Multiple Choice Questions. Questions which have a correct answer from choices offered as a list.

■ Architectural Objectives and Restrictions

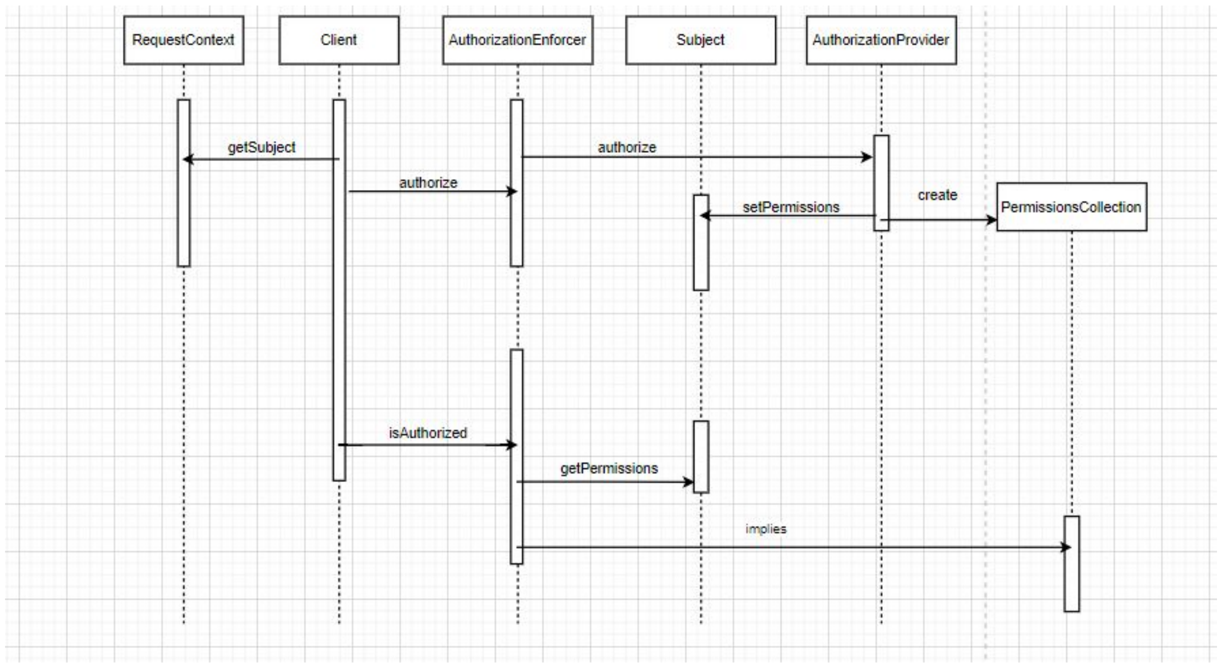
The defined architecture's main objective is to make the system scalable, reliable, have high data integrity and security. As a system that is designed to be an online examination application, data integrity is of utmost importance. Privacy would be another big concern, and students will be more likely to use a system that is secure. The system has to be reliable to ensure that the students will not receive an error during the middle of their exams. The system, being in its initial stages, will have to be scalable to ensure longevity of the system and allow improvements to be made to include more features.

■ Process View

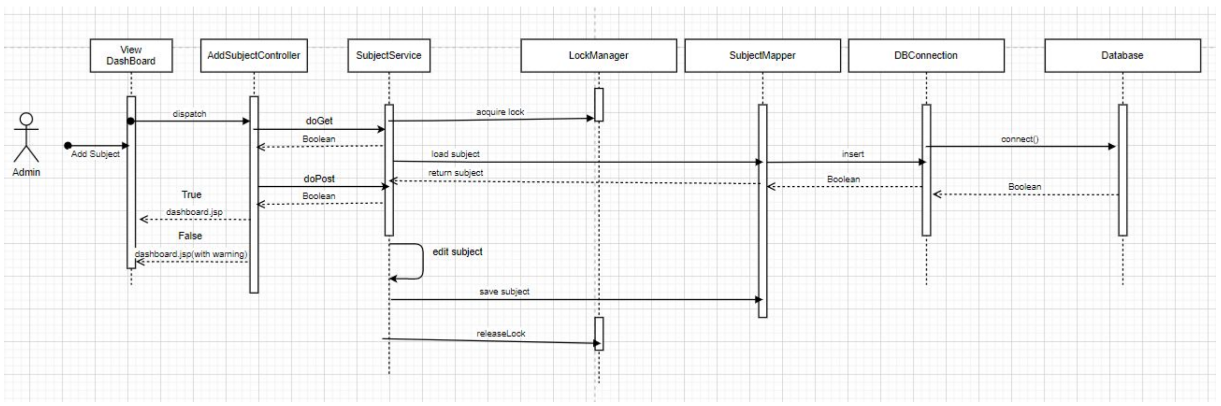
The section shows the mapping of the logical architecture elements to the processes and threads of the system execution. The figures below illustrate this mapping.



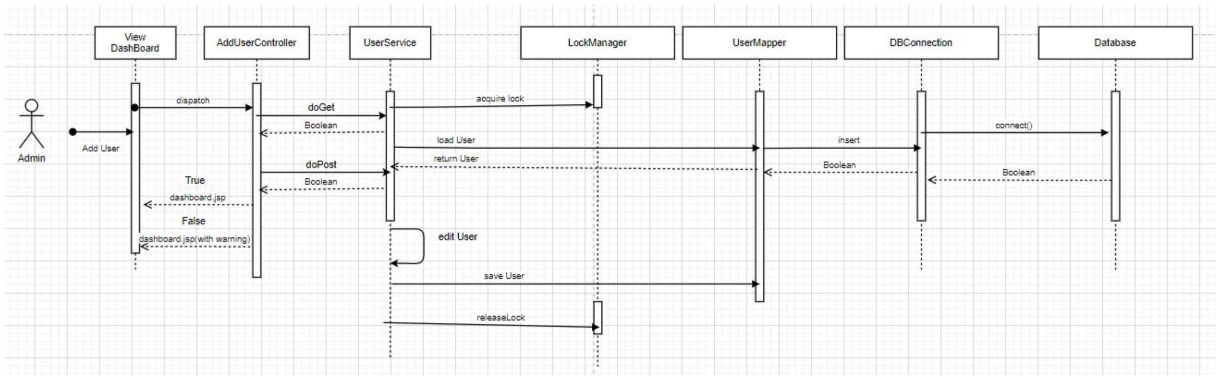
User Logins



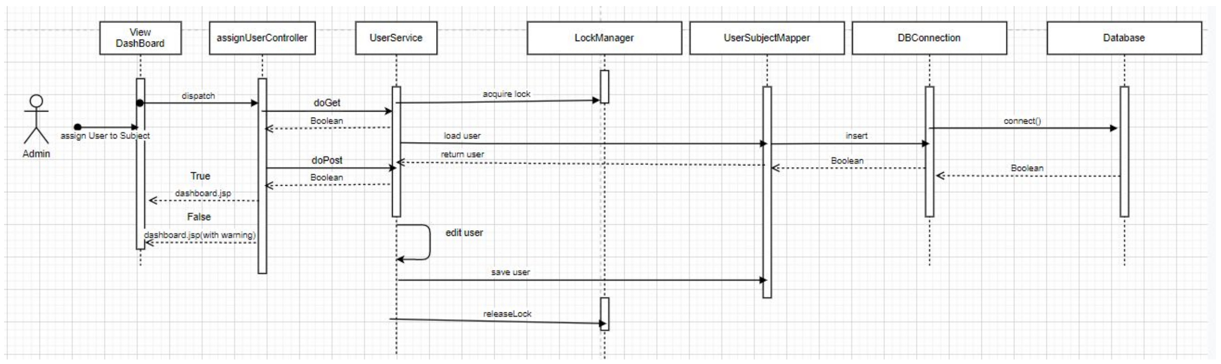
Authorization Enforcer



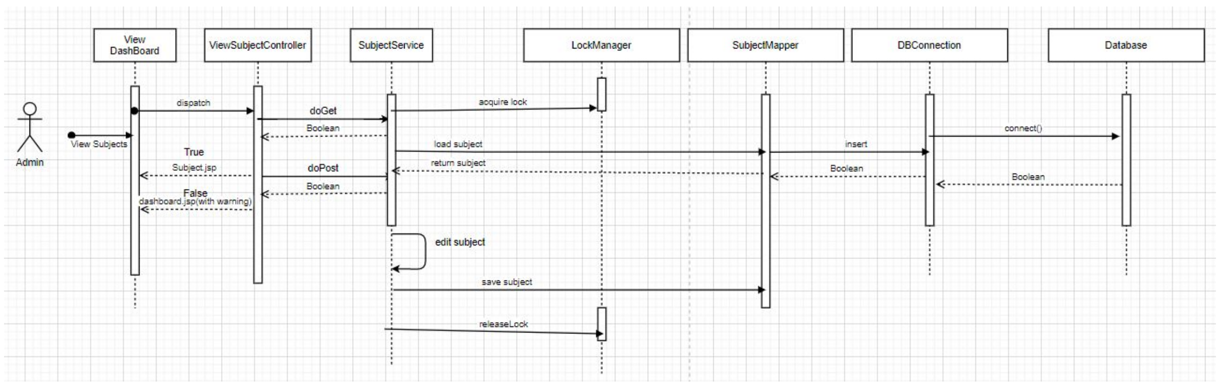
Admin adds subjects



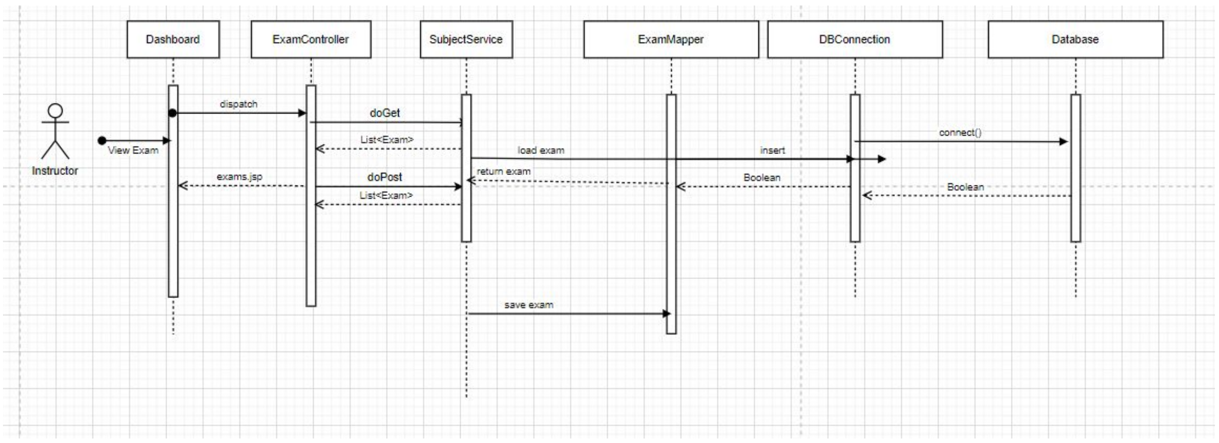
Admin adds user



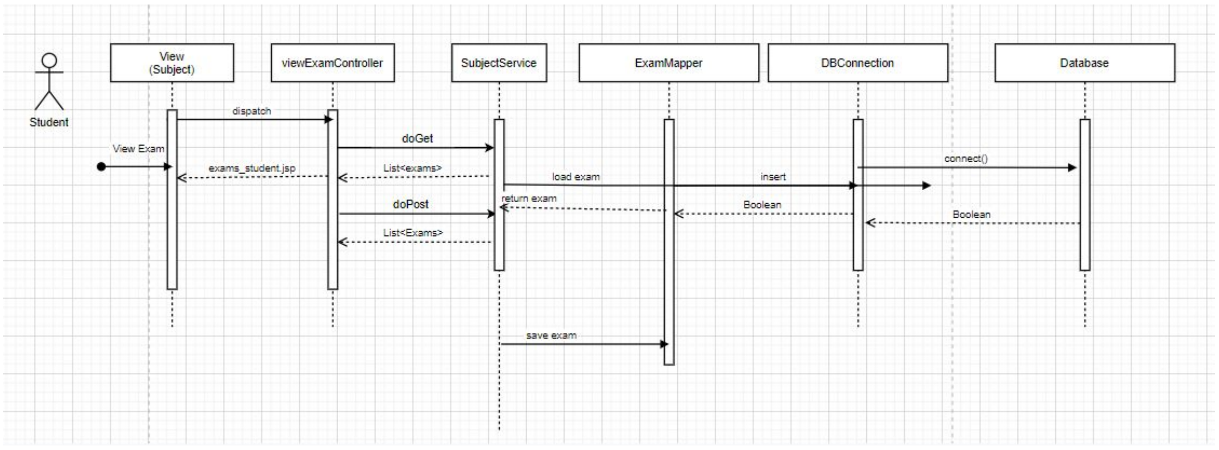
Admin assigns user to subject



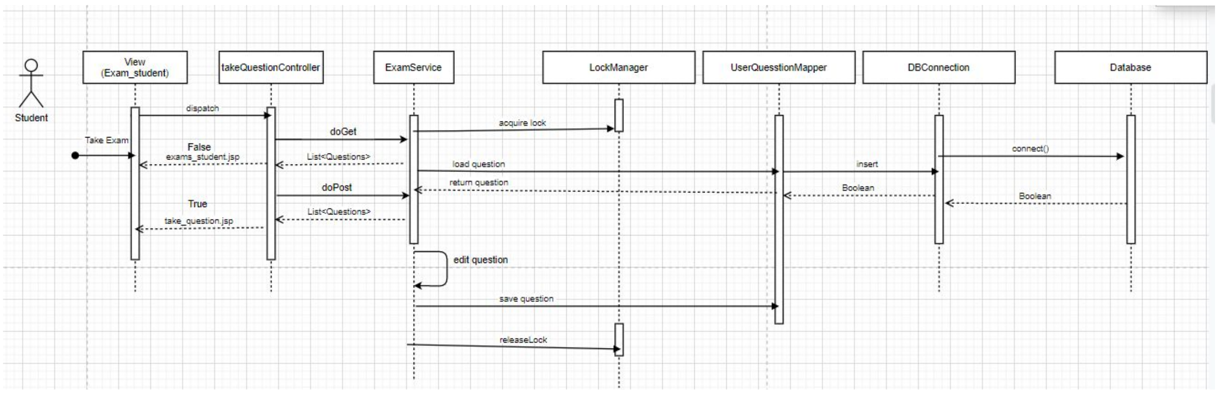
Admin views subjects



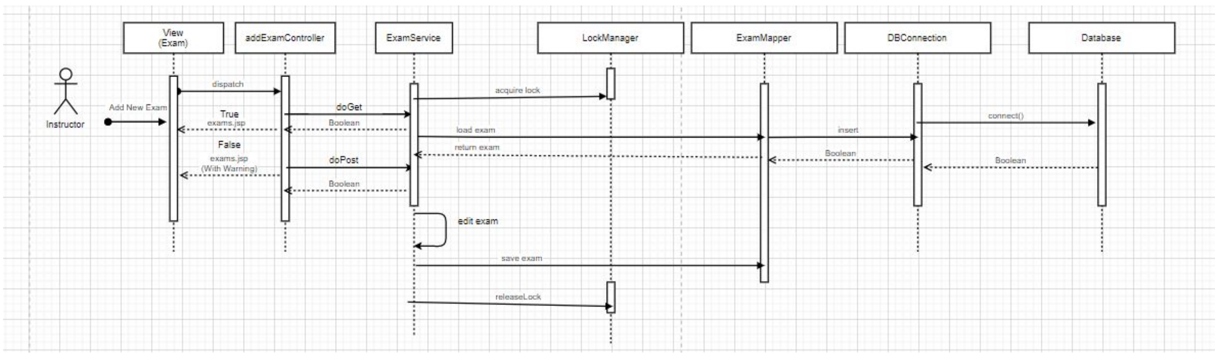
Instructor views exams



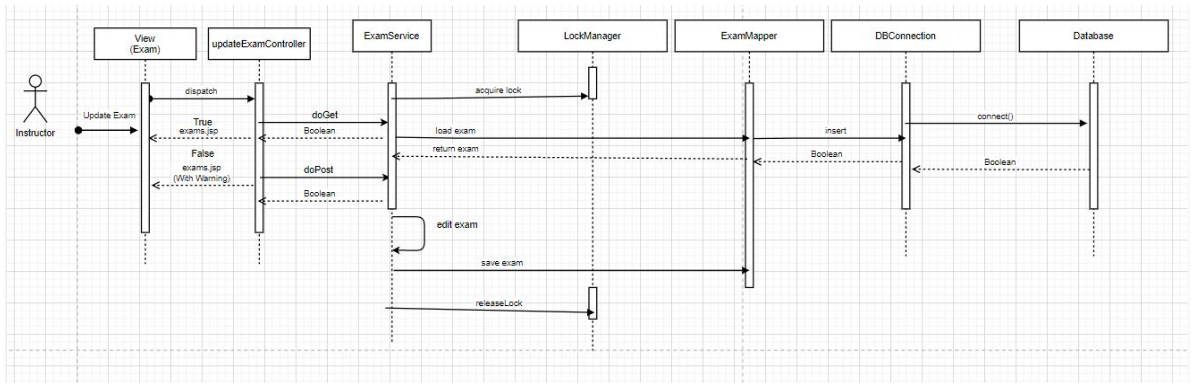
Student views exams



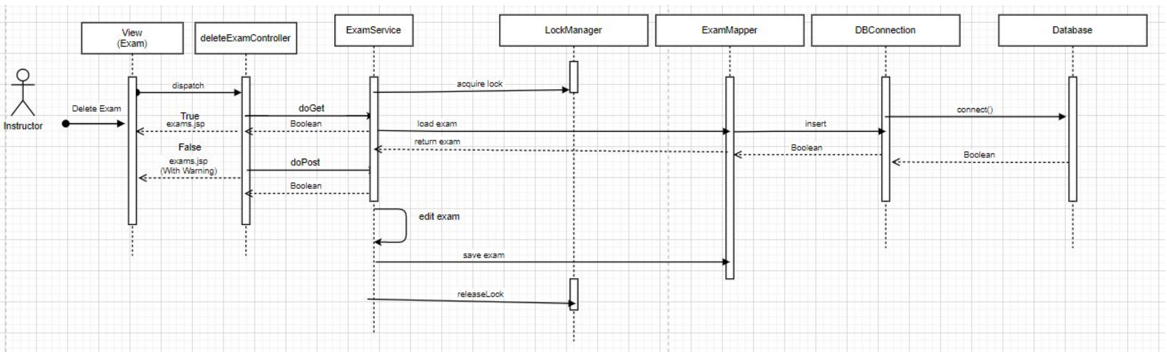
Student takes exams



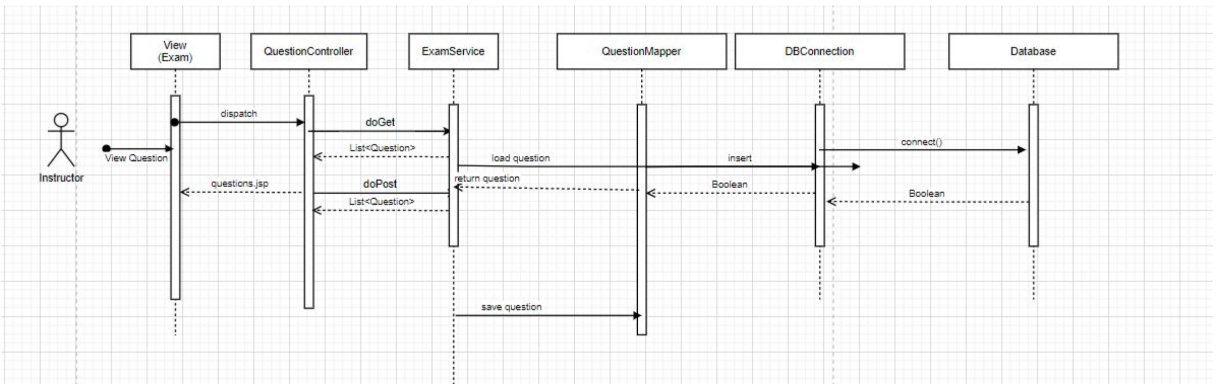
Instructor adds new exams



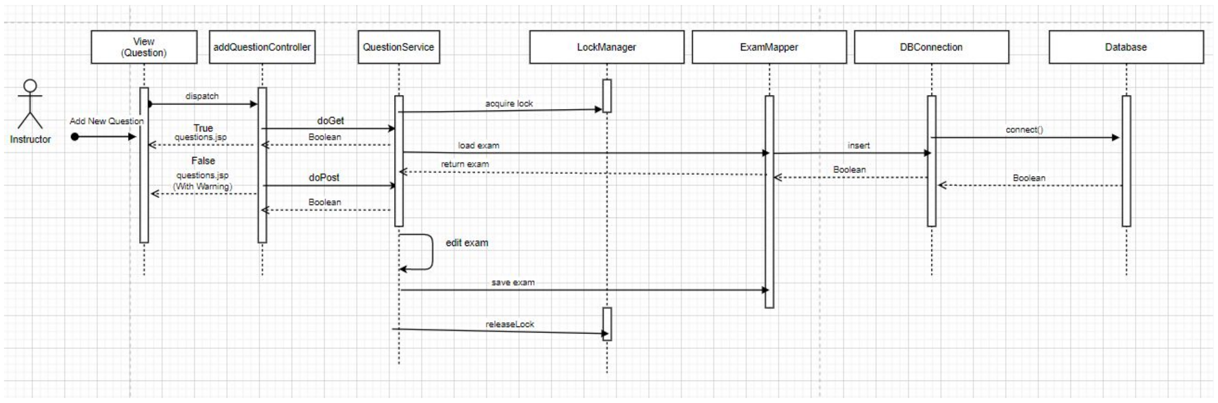
Instructor updates exam



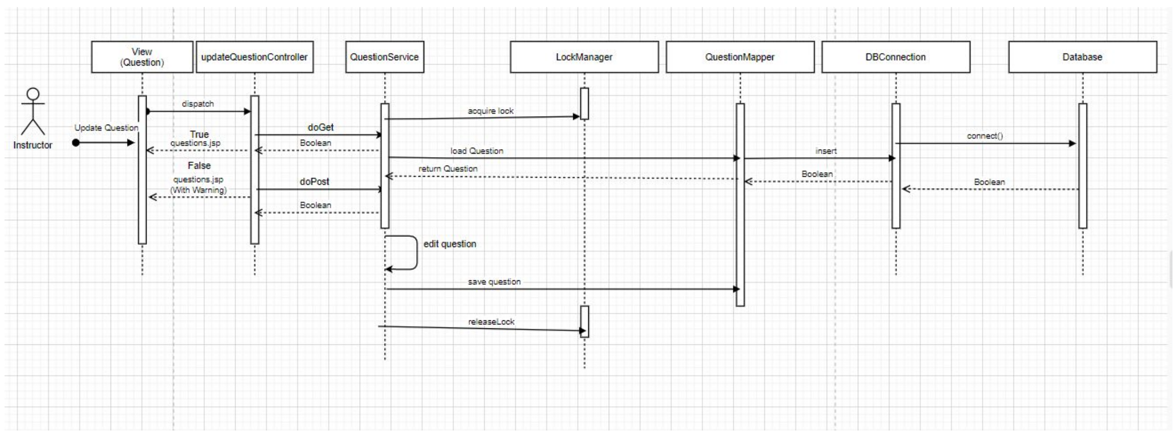
Instructor deletes exam



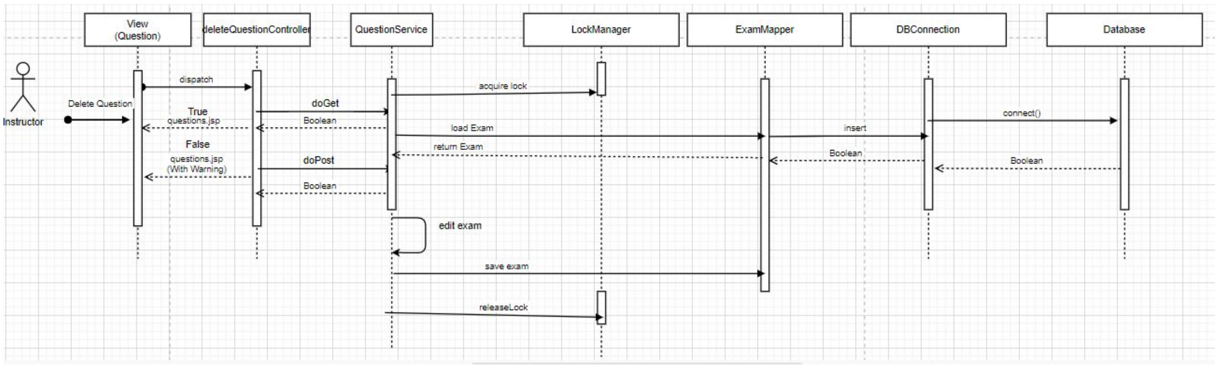
Instructor views questions



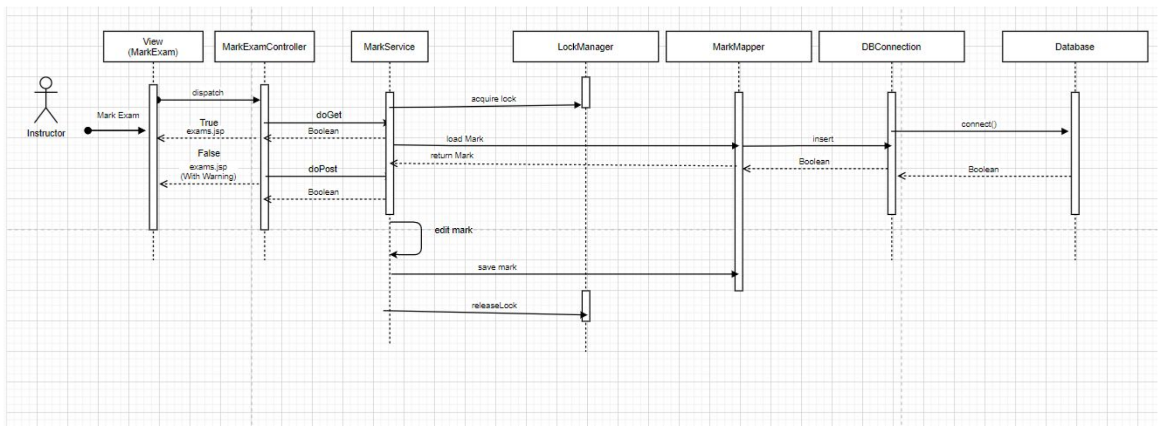
Instructor adds new questions



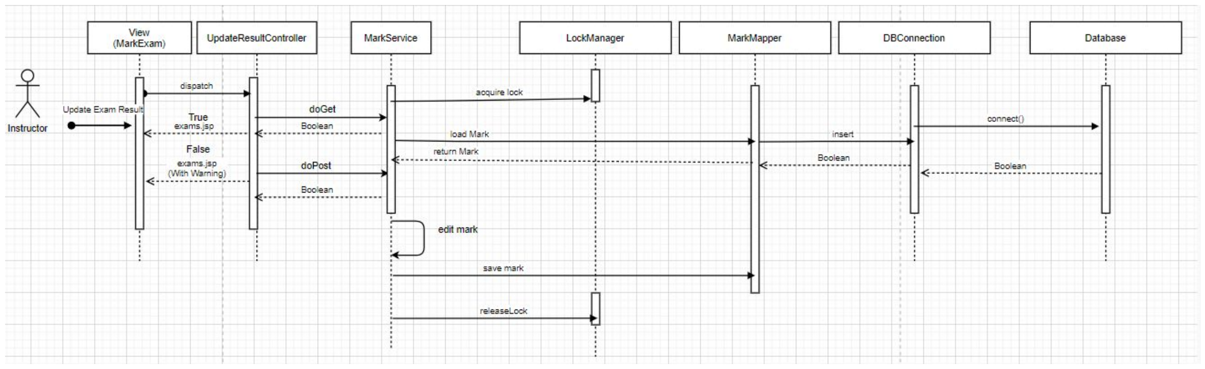
Instructor updates questions



Instructor deletes questions



Instructor marks exam



Instructor updates exam marks

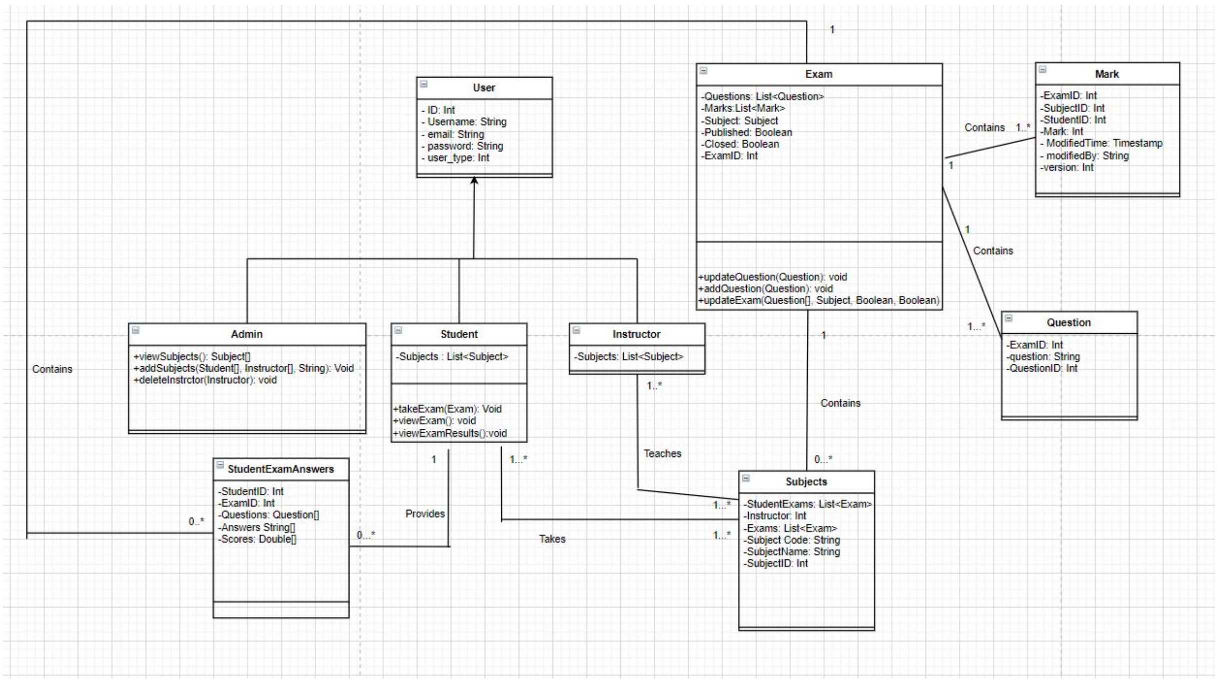
Figures: System execution diagram

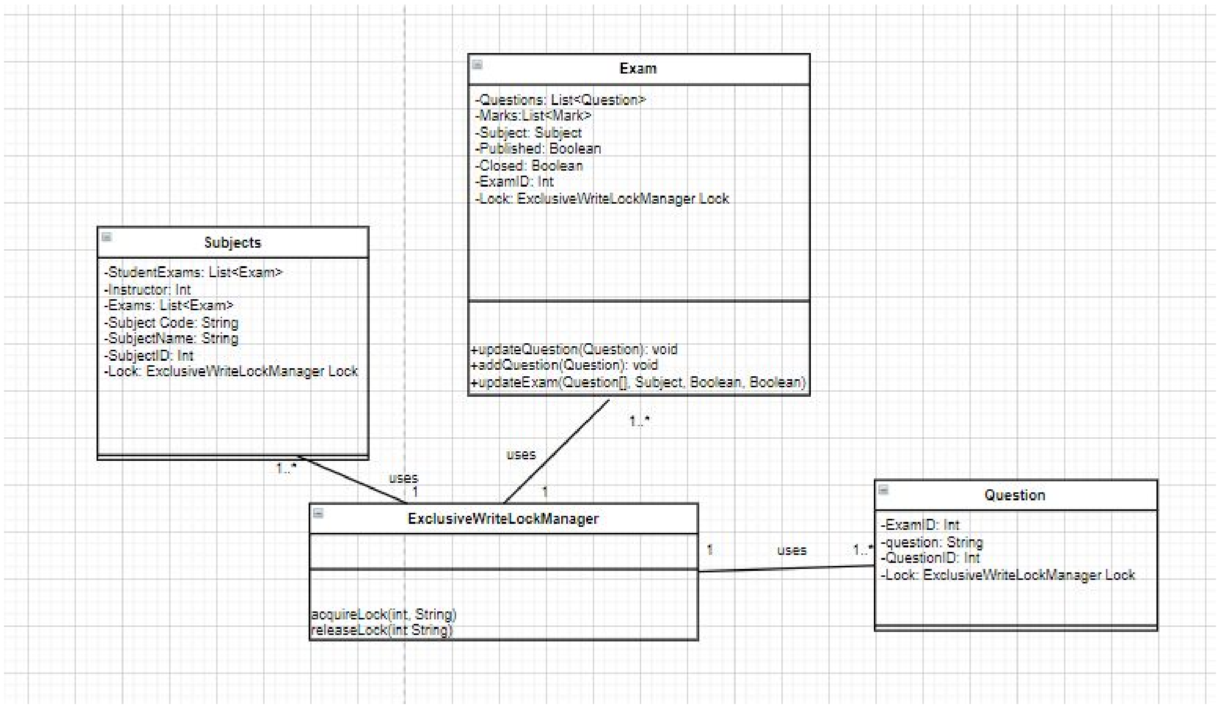
Note: Dashboard.jsp contains the list of subjects, hence there were no sequence diagrams for viewing subjects, since all users will automatically see the list of subjects upon logging in.

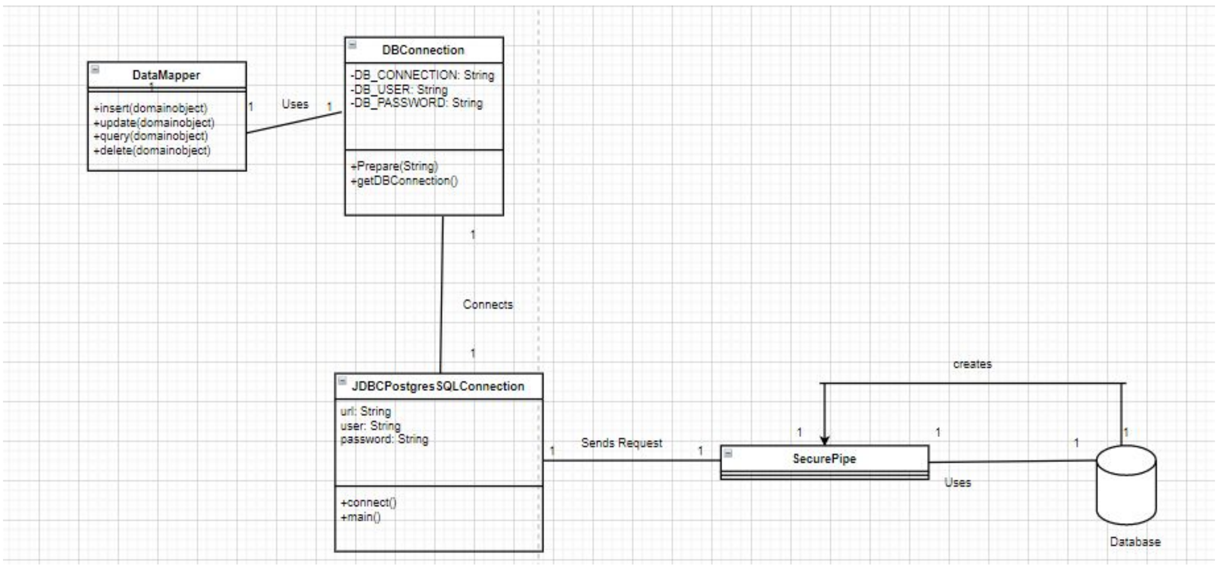
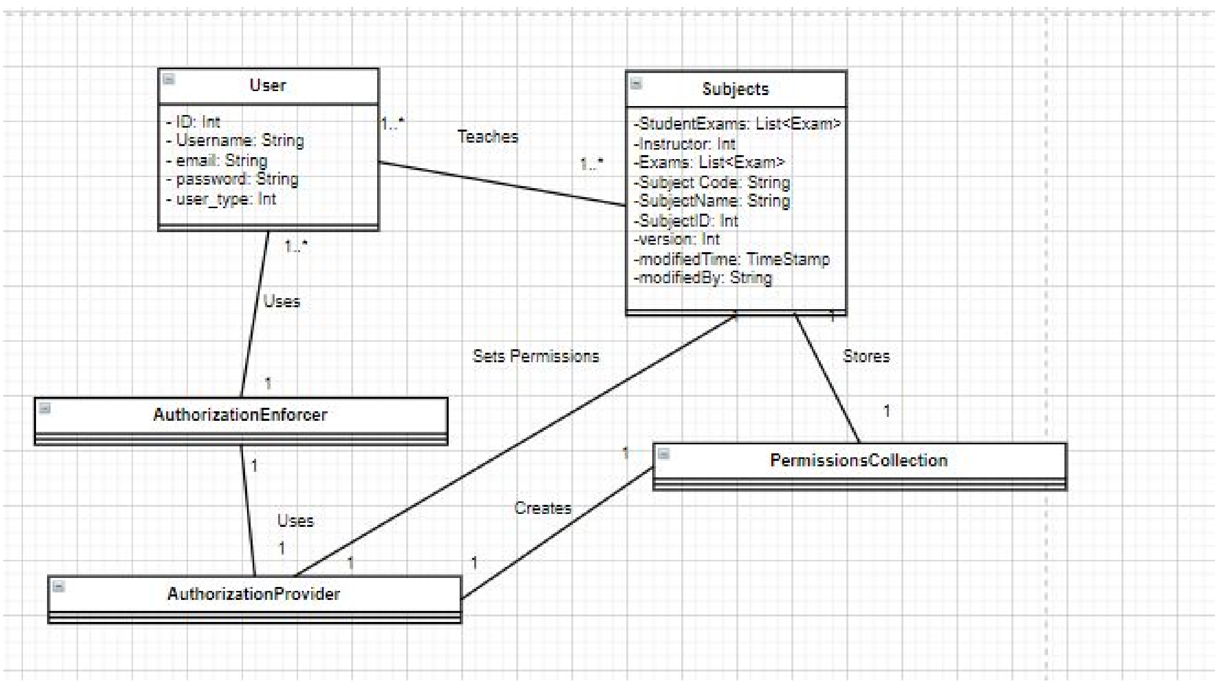
Students will be able to view their marks in the list view of exams. Hence, no sequence diagram for viewing marks is implemented.

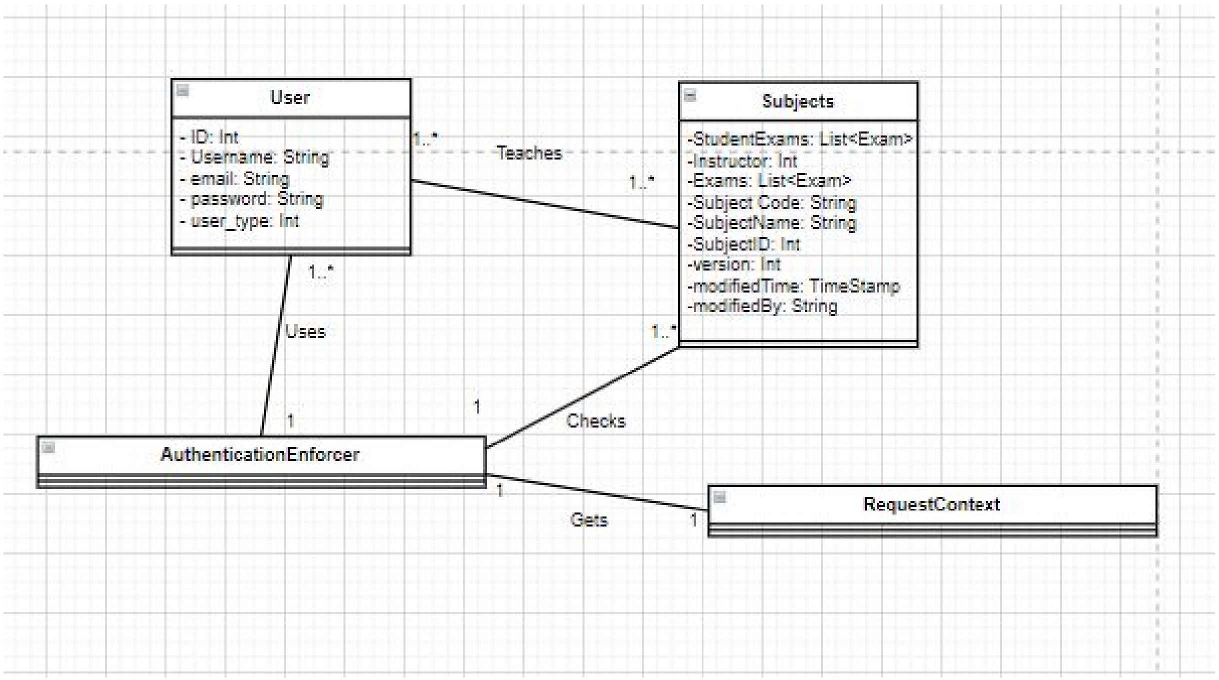
■ Development View

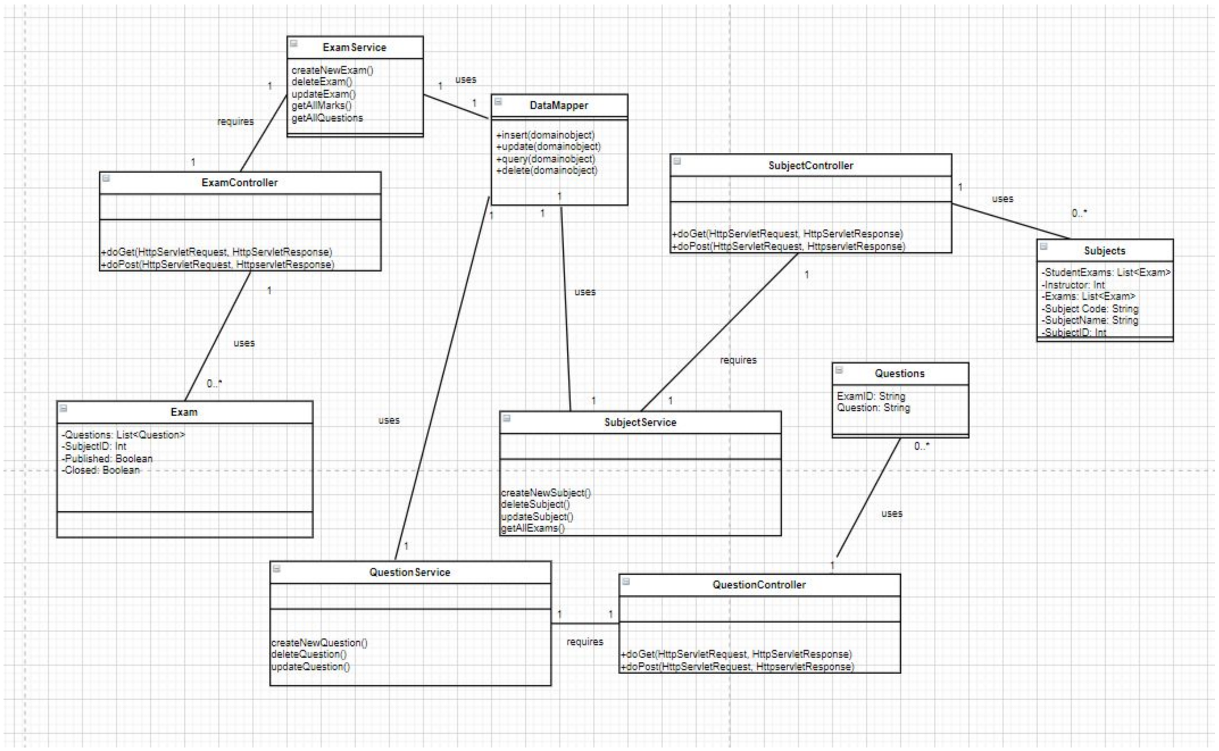
This section provides orientations to the project and system implementation in accordance with the established architecture. The figures below show the implementation view of system architecture. Since the previous document, the system has been updated to handle concurrency and security issues.

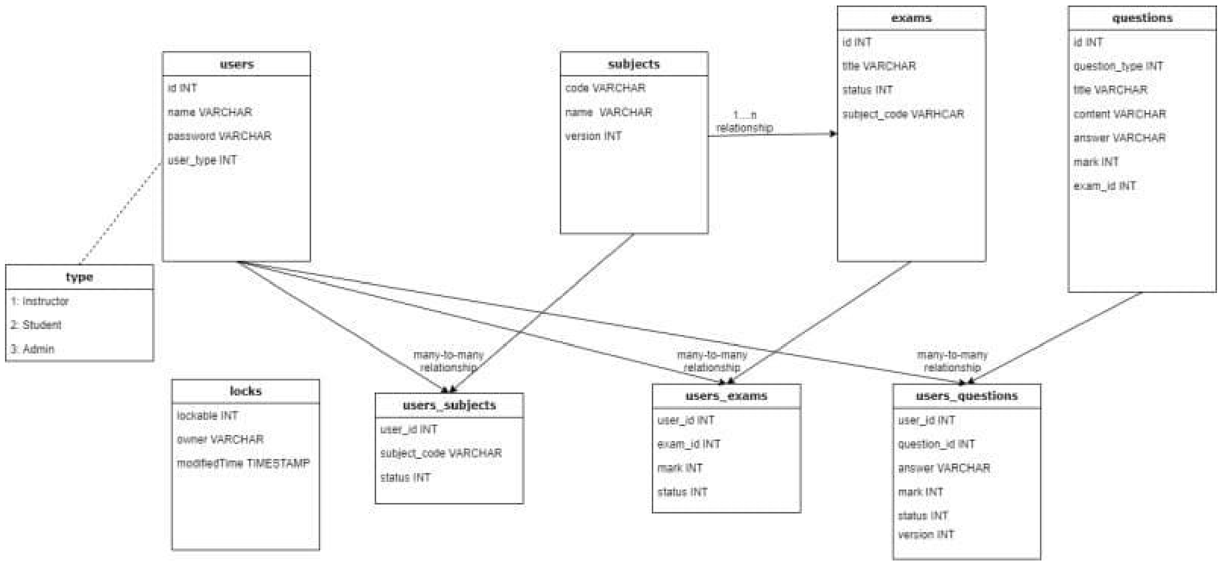



Exam
-Questions: List<Question>
-Marks:List<Mark>
-Subject: Subject
-Published: Boolean
-Closed: Boolean
-ExamID: Int
-Lock: ExclusiveWriteLockManager Lock
+updateQuestion(Question): void
+addQuestion(Question): void
+updateExam(Question[], Subject, Boolean, Boolean)
Subjects
-StudentExams: List<Exam>
-Instructor: Int
-Exams: List<Exam>
-Subject Code: String
-SubjectName: String
-SubjectID: Int
-Lock: ExclusiveWriteLockManager Lock
ExclusiveWriteLockManager
acquireLock(int, String)
releaseLock(int String)
Question
-ExamID: Int
-question: String
-QuestionID: Int
-Lock: ExclusiveWriteLockManager Lock
1..*
uses
1
uses
1..*
1
1
uses
1..*


DataMapper
+insert(domainobject)
+update(domainobject)
+query(domainobject)
+delete(domainobject)
Uses
1
1
DBConnection
-DB_CONNECTION: String
-DB_USER: String
-DB_PASSWORD: String
+Prepare(String)
+getDBConnection()
1
Connects
1
JDBCPostgresSQLConnection
url: String
user: String
password: String
+connect()
+main()
Sends Request
1
1
SecurePipe
creates
1
1
Uses
1
1
Database

User
- ID: Int
- Username: String
- email: String
- password: String
- user_type: Int
1..*
Teaches
1..*
Subjects
-StudentExams: List<Exam>
-Instructor: Int
-Exams: List<Exam>
-Subject Code: String
-SubjectName: String
-SubjectID: Int
-version: Int
-modifiedTime: TimeStamp
-modifiedBy: String
1..*
Uses
1
AuthorizationEnforcer
Sets Permissions
Stores
1
1
PermissionsCollection
1
Uses
1
Creates
1
1
1
AuthorizationProvider


User
- ID: Int
- Username: String
- email: String
- password: String
- user_type: Int
Subjects
-StudentExams: List<Exam>
-Instructor: Int
-Exams: List<Exam>
-Subject Code: String
-SubjectName: String
-SubjectID: Int
-version: Int
-modifiedTime: TimeStamp
-modifiedBy: String
Teaches
1..*
1..*
Uses
1..*
1
Checks
1
1..*
AuthenticationEnforcer
RequestContext
Gets
1
1


ExamService
createNewExam()
deleteExam()
updateExam()
getAllMarks()
getAllQuestions
DataMapper
+insert(domainobject)
+update(domainobject)
+query(domainobject)
+delete(domainobject)
ExamController
+doGet(HttpServletRequest, HttpServletResponse)
+doPost(HttpServletRequest, HttpservletResponse)
SubjectController
+doGet(HttpServletRequest, HttpServletResponse)
+doPost(HttpServletRequest, HttpservletResponse)
Subjects
-StudentExams: List<Exam>
-Instructor: Int
-Exams: List<Exam>
-Subject Code: String
-SubjectName: String
-SubjectID: Int
Exam
-Questions: List<Question>
-SubjectID: Int
-Published: Boolean
-Closed: Boolean
Questions
ExamID: String
Question: String
SubjectService
createNewSubject()
deleteSubject()
updateSubject()
getAllExams()
QuestionService
createNewQuestion()
deleteQuestion()
updateQuestion()
QuestionController
+doGet(HttpServletRequest, HttpServletResponse)
+doPost(HttpServletRequest, HttpservletResponse)
requires
uses
uses
uses
uses
requires
uses
uses
requires
1
0..*



Figures. Development view of system architecture

1.3 Architectural Patterns

The figures below represent the class diagram

Patterns used	Reason	Where it will be used
Data mapper	Datamapper allows for loose coupling within the code by decoupling the database access from the domain subjects. Row data gateway is not used due to the high likelihood of the system being changed in the future, which can lead to database schema changing. By using a data mapper, the design of the domain layer can be separated from the design of the database layer, allowing them to be independently developed.	all queries, deletes and updates within the database will be made through mappers. The following list of mappers are used: AnswerMapper DataMapper (parent class) ExamMapper MarkMapper QuestionMapper SubjectMapper UserMapper UserQuestionMapper The mappers allow for the domain layer and the database layer to be less coupled. Should a change happen to the domain layer, the schema within the database is unaffected and can remain the same, with the mappers requiring updates to accommodate for the changes. (Major overhauls might still affect the database layer)

Optimistic Offline Lock	Optimistic Offline Lock is a good way to implement concurrency. The simplicity means that the code is easily understandable. This lock also ensures a high level of liveness of the program. However, one disadvantage of having this lock would be the loss of data that occurs when multiple users attempt to update the same parameters. Due to this reason, the optimistic offline lock is only used in scenarios where the loss of data is negligible.	Optimistic Offline Lock is only used when updating the overall marks of students. If there are conflicting updates to the mark of a single student, only the singular mark data is lost, with no major disadvantages. By using the optimistic offline lock in this scenario, it allows for more liveness.
Pessimistic Offline Lock	Another way to implement concurrency. Compared to optimistic offline lock, pessimistic offline lock does better at preventing loss of data at the cost of livenes. The type of pessimistic offline lock to be implemented is the exclusive write lock. In this scenario, users would still have access in reading the data. However, only one user is able to edit the data at any point in time. While this has a highest liveness than an exclusive read lock, the reads can potentially become inconsistent. A problem with this pattern is that there can potentially be a deadlock. A modified time variable is implemented to check the time of the last acquired lock whenever a new request is initiated. If the time is longer than 24 hours, the new user would be given the lock privileges	Used in exams, questions and marking each question of the exams. Optimistic offline lock is not used for these three objects due to the high loss of data that can occur when multiple instructors are editing these domain objects. Updating or deleting questions to exams can result in huge changes to the exam, and these changes could potentially be lost if another instructor was concurrently updating the exam. (Adding will not result in an error due to the autoincrement feature of the database) Hence, only one instructor is able to edit a given exam or question at a time. While there will be inconsistent reads, eventual consistency guarantees that all users will be able to see the same exam after all the edits are complete (When the exams are published, no one will be able to edit the exam. Refreshing the page after will lead to everyone having the same read).
Authentication Enforcer	A solution to ensuring that requests are from the right users. This pattern allows for reusability by having all authentication logic in one place. It is easily maintainable and editable. Additionally, it works well with authorization enforcers and secure pipe to ensure the entire system is safe.	Used for users of the system. Admins, instructors and students all have different abilities within the system. These are implemented through the authentication enforcer to ensure that each request is given by the right user with the correct credentials. For example, a student will not be able to request the server to update his marks.

Authorization Enforcer	Ensures that users are only allowed to access resources that they are allocated. This prevents security leaks and promotes separation of responsibility. Different users will have different functions that will not overlap each other.	Used for users of the system. Admins, instructors and students all have different access powers to different resources within the system. This ensures that no users will be able to view data that is not privy to them. Students and instructors both have access to subjects, exams and questions. However, they each will have their own respective views based on their user identity. courses/SWEN90013/exams will allow instructors to edit, publish, delete, close, mark or update the exam, whereas for a student, he can only take the published exams and view his results. courses/SWEN90013/exams/100/View Answer (detail mark view) and courses/SWEN90013/exams/100/View Mark (table mark view) will only allow the user to access this if the user is an instructor. Should a student attempt to navigate to this link, there will be an authorization error and attempt to navigate the user to the previous page.
Secure Pipe	Prevents sensitive information being transferred over a network from being leaked or tempered. Mainly done through cryptography. Hackers that access these data will be unable to understand the contents being transferred without a key. While this pattern comes at a small cost to performance, it is a small price to pay for security.	Used for the transfer of data between the database and the system of the user. This works well with the authorization enforcer to ensure that only people with the permissions have access to the data in the database. It is a good pattern to consider. However, since the product is deployed on Heroku, there is not much emphasis placed on this pattern.
Foreign Key Mapping	Allows for tables to be easily connected to other tables within the database for one to many relationships. The table with many relationships will contain a foreign key so that it can easily determine which entry in the other table it is related to.	Every table within the database that maps many to one entry in another table will contain the foreign key of the associated tables. For example, many questions can be associated with a single exam. In this case, each question will contain a foreign key that relates to the primary key of the said exam.

		User_subjects, user_exams, user_questions, subjects contain the foreign key of users Exams contain the foreign keys of subjects Questions contain the foreign keys of exams User_questions contain the foreign keys of questions
Association Key Mapping	Allows for tables to be easily connected to other tables within the database for many to many relationships. It is done by adding a table between these two tables, forming two one-to-many relationships, which allow for foreign key mapping to identify the relationships	Every table within the database that maps many to many entries to another table will use the association key mapping. Multiple users can be mapped to multiple subjects, questions and exams. In these cases, tables are established that keep track of these many-to-many relationships.

■ Scenarios

Heroku link: https://swen90007-lms.herokuapp.com
Git release tag: SWEN90007_2020_Part3_<SDA_Wombat>
The following accounts have been created for the project.

Student

username: Student
password: 123

username: Student2
password: 123

Instructor

username: Instructor
password: 123

username: Instructor2
password: 123

Admin

username: Admin
password: 123

Scenarios:

Student

- Student logs in (Student is shown a list of subjects)
- Student views exams in a subject
- Student attempts an exam for the first time
- Student views the exam result after the instructor has marked it.

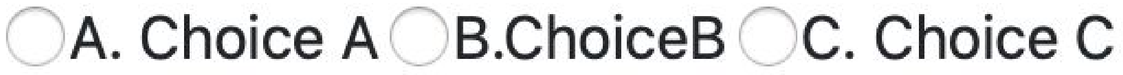
Instructor

- Instructor logs in (Instructor is shown a list of subjects)
- Instructor views exams in a subject
- Instructor creates a new exam
- Instructor creates a new question*
- Instructor deletes/updates an existing exam
- Instructor deletes/updates an existing question
- Instructor publishes exam
- Instructor marks student answers
- Instructor updates student answers

Admin

- Admin logs in (Admin is shown a list of subjects and users)
- Admin adds a new subject
- Admin adds a new user (For user type: place 2 for student, 3 for instructor)

* For multiple choice questions, place the answer content in the format of: A. Choice A#B.ChoiceB#C. Choice C. This will produce the following result:



Note: The database response is slow, please give it some time before refreshing.