Software Architecture Report for Hogwarts O.W.L.s Exam System

SWEN 90007 Software Design and Architecture Report (Part 3)

Git release tag

SWEN90007_2020_Part3_SDA4

Deployment link

https://swen90007-2020-sda4.herokuapp.com/

Contributors

Akhmetzhan Kussainov - 1026301

Robert Sharp - 186477

Fanyu Meng - 1026364

Paritosh Wadhavane - 1004023

Person	Job	Hours	Contribution Level
Akhmet	Redeploy the application on Heroku	1	High
Akhmet	Facilitate administrator login	1	Low
Akhmet	Integrate Exams service with Exams data Mapper	1	Medium

Akhmet	Merge front-end and backend code for	2.5	Medium
Akhmet	Integrate the new version of the front end and update the backend to fit the integration	12	HIgh
Fanyu	Research SD Programming	1	Medium
Paritosh	Create a facade with Exams and Exam Content (including Questions and Marking)	9	High
Paritosh	Make .gitignore and delete all class models in repo	0.5	Medium
Paritosh	Make login support Admin.	0.5	Low
Paritosh	Create a facade with Scriptbooks and Multiattempts/Shortattempts	4.5	High
Paritosh	Create a facade with Marking and Results for Student and Teacher (JSP/Servlet/Service)	6	High
Robert	Authorisation and Authentication	4.5	Med
Fanyu	Pessimistic Offline Lock	2	High
Fanyu	Sequence diagram and Description	1	Med
Ackmet	Continue Integration	5	High
Robert	Creation of subjects,	5	Med

	alerts for subject creation and login		
Robert	Integration	10	High
Fanyu	Component Unit test for ExclusiveLockManager and debug	3.5	High
Fanyu	Create Test Cases and Testing	2.5	Med
Paritosh	Working on class diagram	6	High
Paritosh	Bug fixes	1.5	High
Ackmet	Deployment, sequence diagrams, pseudo code for checking exam, description of security patterns	4	High
Robert	Integration, optimistic concurrency	10	High

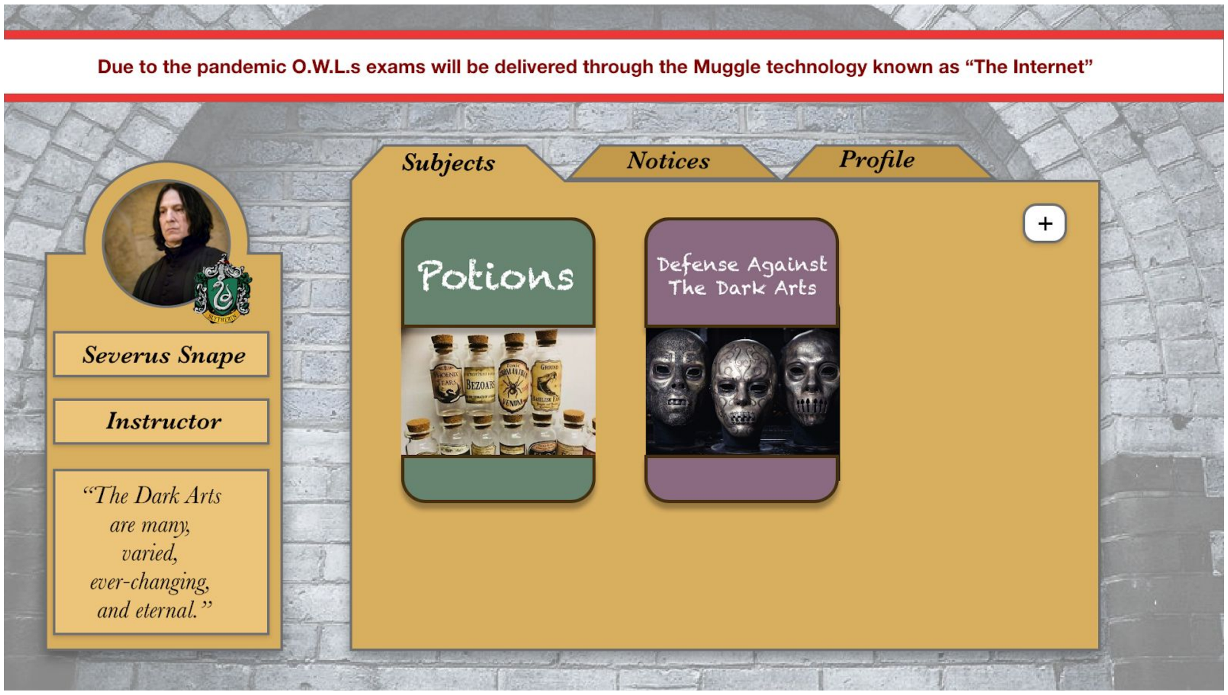
Revision History

Date	Version	Description	Author
4th Oct 11:30 am	1.0	Done structure, state diagram for exams, domain diagram, description of patterns, pasted personas in, pasted ui mockup in, pasted Fanyu’s use case diagram, wrote three user scenarios, wrote two test cases, wrote business rules	Robert

4th Oct 12:30 am	1.1	Sequence diagram for Unit of Work, Embedded Value	Fanyu
4 th Oct 9:00 pm	1.2	Identity Map sequence diagram, lazy load sequence diagram	Robert
4th Oct 9:30 pm	1.3	Domain Model Pattern sequence diagram	Fanyu
31 Oct 11:00 pm	2.0	Authentication, Authorisation and Security Pipe patterns description and attached diagrams	Akhmetzhan

Introduction

This is the Software Design Document for an Enterprise System for the Hogwartz School of Witchcraft and Wizardry. Due to the Covid pandemic, the school has moved online and uses the internet to undertake student assessment.


Due to the pandemic O.W.L.s exams will be delivered through the Muggle technology known as "The Internet"
Subjects
Notices
Profile
+
Potions
Defense Against The Dark Arts
Severus Snape
Instructor
"The Dark Arts are many, varied, ever-changing, and eternal."

A mockup for the UI design made by Robert. We did not have time to implement a complex

user interface.

The Exam system uses persistent data held in databases.

The system supports multiple users. Teachers (or instructors) create exams and mark them. Students take exams. Admin can create subjects.

Personas

1. Dobby - Administrator

	Description: House elf. Has high technical abilities. Should not be able to create exams as is not a qualified wizard. Problems: Some Hogwarts staff may be death eaters and Dobby needs to keep his eye on them. He should be able to view all details of subjects and their associated staff, students and exams. Goals/Needs: Be able to register subjects and their instructors.

2. Snape - Instructor

	Description: Teacher at Hogwarts (Slytherin house). He is a Potions Professor. May be a death eater. Teaches multiple subjects. Teaches “Defense against the Dark Arts” and “Potions“ by himself. He also co-teaches “Herbology“ with Professor Pomona Sprout. Low technical ability. Problems: Harry Potter.

	Dislikes marking multiple choice exams. Goals/Needs: To create exams. To mark exams.

3. Harry Potter - Student

	Description: Muggle student. Belongs to Gryffindor House. Moderate technical ability. Will be enrolled in multiple exams for multiple subjects. Problems: Dementors. Goals/Needs: Needs to know when exams will be. Needs to be able to take his exam and know his answers will be reliably recorded by the system.

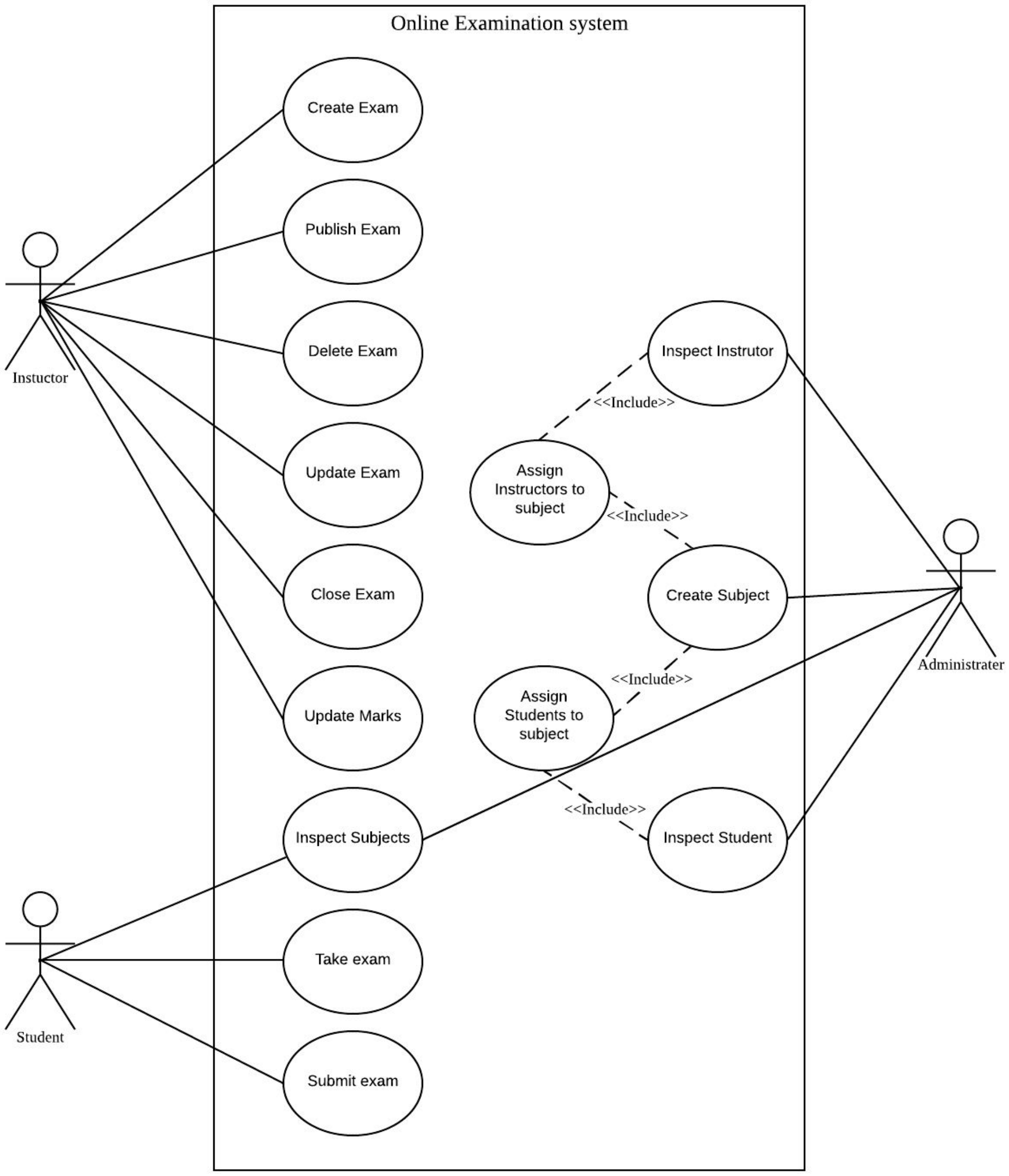
4. Dumbledore - Headmaster

	Description: Staff. Has very little technical knowledge. Problems: Muggle technology Goals/Needs: Needs to know information of subjects and students accessible by admin page without accidentally creating new subjects or clearing locks.

Use Cases

Please see the initial submission for use cases.

Use Case diagram



Use case diagram showing use cases for administrator, student and instructor.

User Scenarios

1. Dobby creates a new subject. It is called “How to breath underwater”. He enrols Harry Potter in this subject at the time of creation, since he is the only student who needs to learn this subject.
2. Snape creates a new midsemester exam for his subject “Defense against the Dark arts”. He publishes it. The time to begin the exam is 5:00 PM (United Kingdom time zone). The duration is 30 minutes. However, he finds out that some students had stolen the answers from his desk using an invisibility cloak. So Snape closes the Exam before the 5:00 PM start time.
3. Harry takes an final exam for Potions class. However he doesn’t submit his exam before the deadline. He asks the instructor to tell him what mark he got. The instructor views the scriptbooks for the exam, and tells Harry he received 0.

Business Rules

Here are the business rules specified in the application domain.

1. One administrator
2. A subject must have one or more instructors and one or more students

once an exam is published it cannot be unpublished

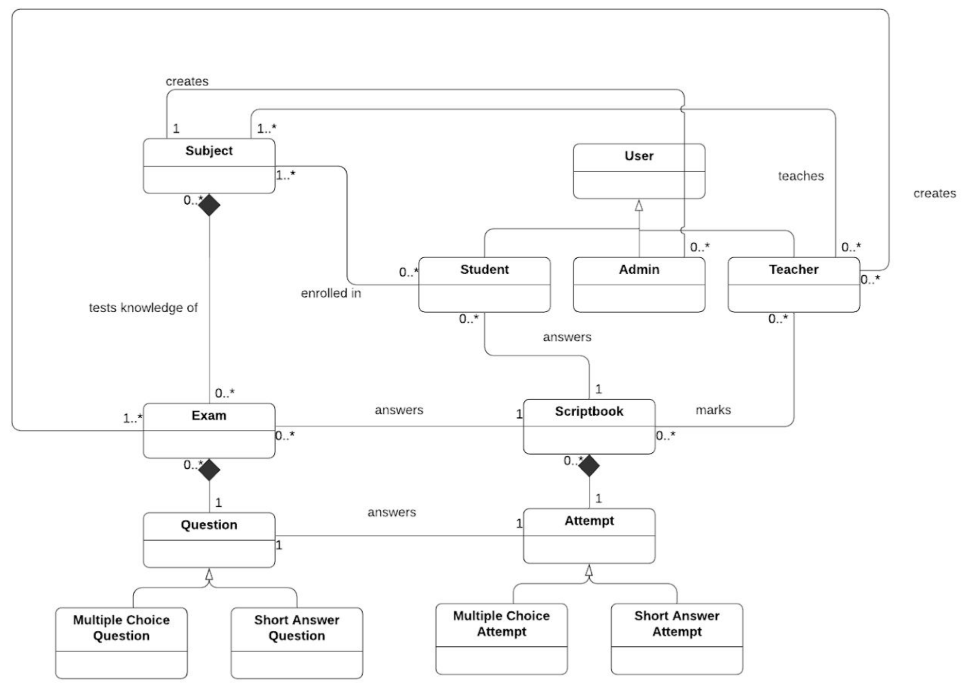
3. Exams have one or more questions

Here are some business rules we invented because they were necessary or seemed plausible:

1. Student can see every exam in subject, not just their year/semester (might help with revision).
2. exams for a subject can’t have the same name (we created this rule to make it easier for students to read)
3. A subject can’t have two exams with the same type for a year and semester (eg. can only have one mid semester exam).
4. Students can see all exams for a subject from previous years (to help with revision - our rule)
5. Teachers have a teacher number that starts with T, students have a student number that starts with S, and admin have an admin number that starts with A. This was a leftover from when Robert created the database and didn’t know how to create composite keys and is no longer necessary.
6. Students can’t also be teachers, teachers can’t also be admin etc unless they use separate accounts. This simplifies things (otherwise we would need a list for user type).
7. Exams consist of multiple choice questions shown as a seperate section to short answer questions and the numbering of questions for each section starts from 1.

Domain model

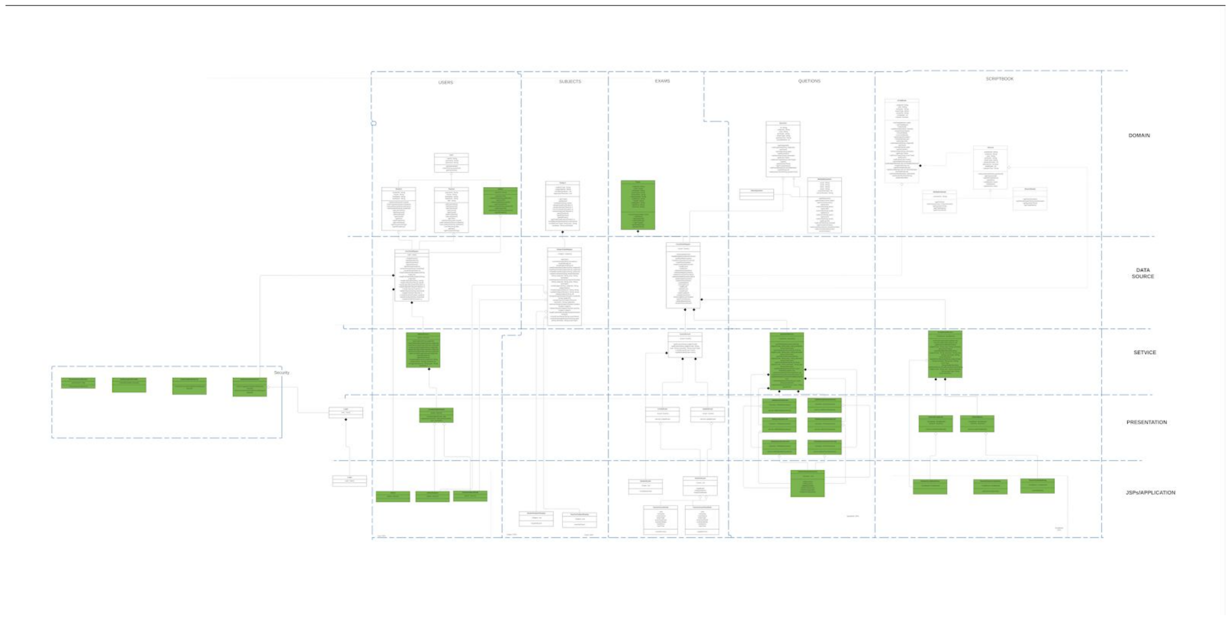
The domain model:



The Domain Model. We assumed that when admin deletes a subject, the associated exams would also be deleted. If that was not the case, then there would be an aggregation relationship rather than composition relationship seen in the diagram.

Class diagram

The class diagram:



UML Class Diagram

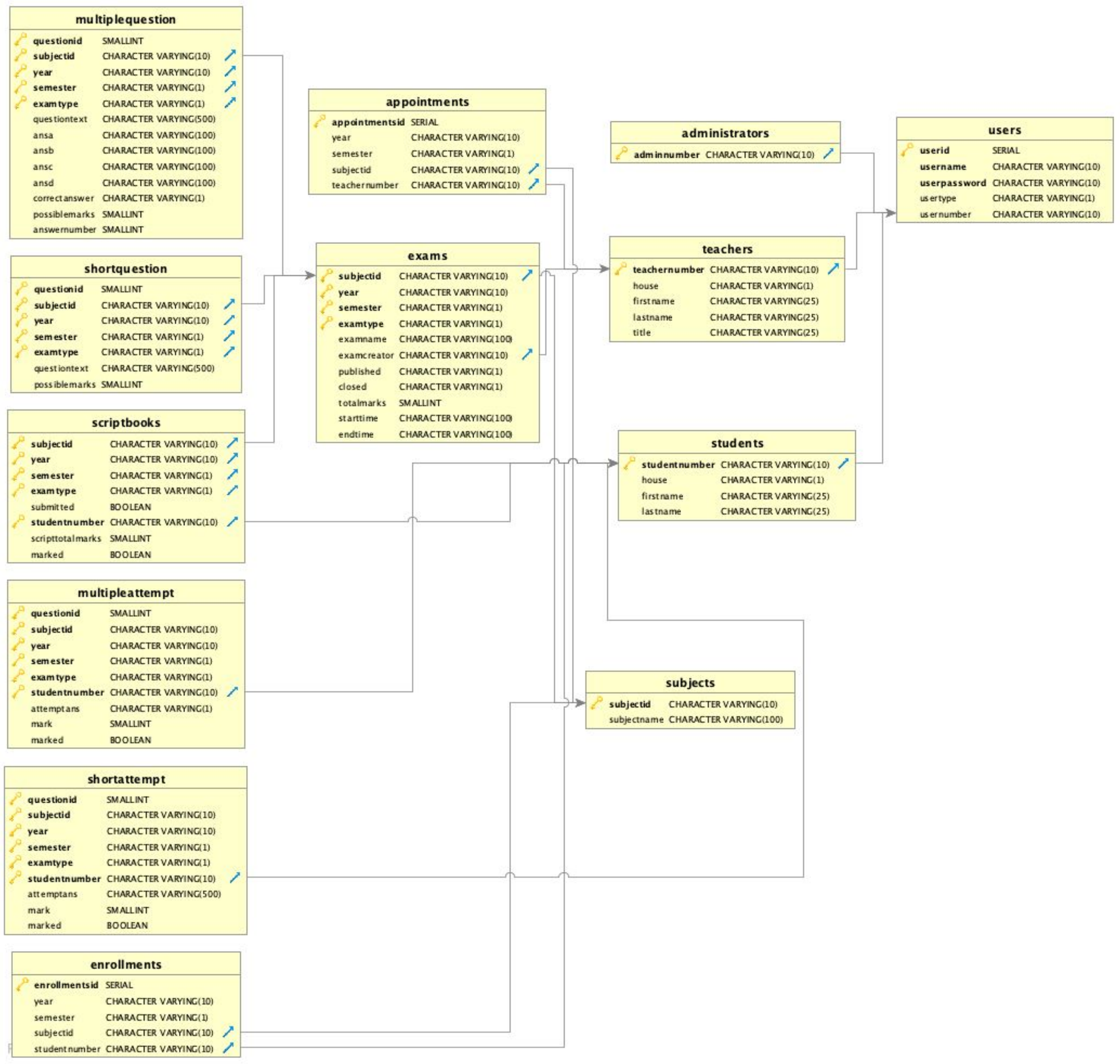
Check out individual models here. (Or scroll down to the appendix). A large version as a pdf is also uploaded to the git.

Architecture

Our application had three layers: Presentation, Domain and Data. The Presentation layer dealt with information presented to users. The Data layer is the SQL database holding the data. The Domain layer includes the data mapper classes that take the data from the database and transform it into domain objects.

Database Design

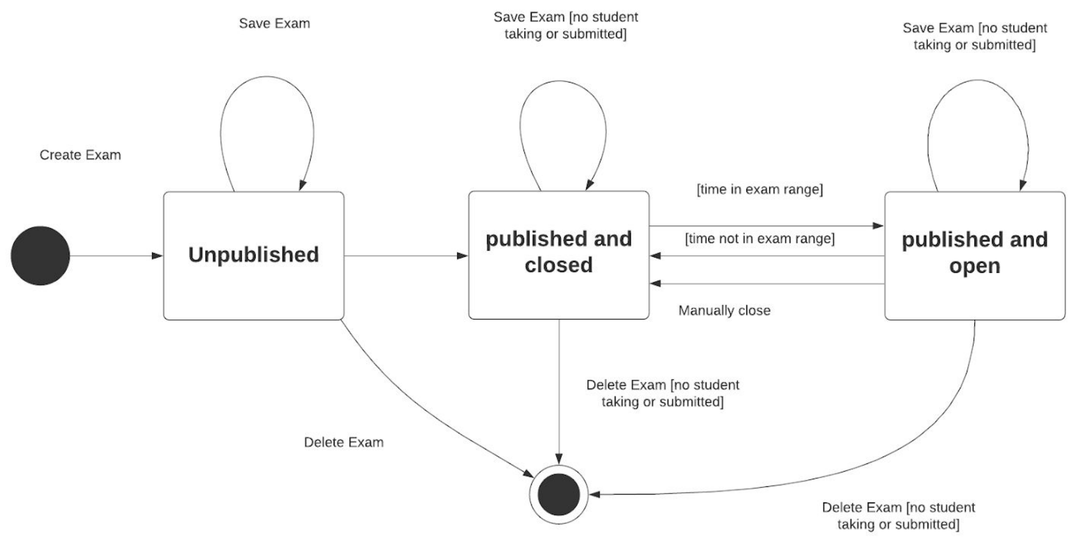
This is the design for the database:


multiplequestion
questionid SMALLINT
subjectid CHARACTER VARYING(10)
year CHARACTER VARYING(10)
semester CHARACTER VARYING(1)
examtype CHARACTER VARYING(1)
questiontext CHARACTER VARYING(500)
ansa CHARACTER VARYING(100)
ansb CHARACTER VARYING(100)
ansc CHARACTER VARYING(100)
ansd CHARACTER VARYING(100)
correctanswer CHARACTER VARYING(1)
possiblemarks SMALLINT
answernumber SMALLINT
shortquestion
questionid SMALLINT
subjectid CHARACTER VARYING(10)
year CHARACTER VARYING(10)
semester CHARACTER VARYING(1)
examtype CHARACTER VARYING(1)
questiontext CHARACTER VARYING(500)
possiblemarks SMALLINT
scriptbooks
subjectid CHARACTER VARYING(10)
year CHARACTER VARYING(10)
semester CHARACTER VARYING(1)
examtype CHARACTER VARYING(1)
submitted BOOLEAN
studentnumber CHARACTER VARYING(10)
scripttotalmarks SMALLINT
marked BOOLEAN
multipleattempt
questionid SMALLINT
subjectid CHARACTER VARYING(10)
year CHARACTER VARYING(10)
semester CHARACTER VARYING(1)
examtype CHARACTER VARYING(1)
studentnumber CHARACTER VARYING(10)
attemptans CHARACTER VARYING(1)
mark SMALLINT
marked BOOLEAN
shortattempt
questionid SMALLINT
subjectid CHARACTER VARYING(10)
year CHARACTER VARYING(10)
semester CHARACTER VARYING(1)
examtype CHARACTER VARYING(1)
studentnumber CHARACTER VARYING(10)
attemptans CHARACTER VARYING(500)
mark SMALLINT
marked BOOLEAN
enrollments
enrollmentsid SERIAL
year CHARACTER VARYING(10)
semester CHARACTER VARYING(1)
subjectid CHARACTER VARYING(10)
studentnumber CHARACTER VARYING(10)
appointments
appointmentsid SERIAL
year CHARACTER VARYING(10)
semester CHARACTER VARYING(1)
subjectid CHARACTER VARYING(10)
teachernumber CHARACTER VARYING(10)
exams
subjectid CHARACTER VARYING(10)
year CHARACTER VARYING(10)
semester CHARACTER VARYING(1)
examtype CHARACTER VARYING(1)
examname CHARACTER VARYING(100)
examcreator CHARACTER VARYING(10)
published CHARACTER VARYING(1)
closed CHARACTER VARYING(1)
totalmarks SMALLINT
starttime CHARACTER VARYING(100)
endtime CHARACTER VARYING(100)
administrators
adminnumber CHARACTER VARYING(10)
users
userid SERIAL
username CHARACTER VARYING(10)
userpassword CHARACTER VARYING(10)
usertype CHARACTER VARYING(1)
usernumber CHARACTER VARYING(10)
teachers
teachernumber CHARACTER VARYING(10)
house CHARACTER VARYING(1)
firstname CHARACTER VARYING(25)
lastname CHARACTER VARYING(25)
title CHARACTER VARYING(25)
students
studentnumber CHARACTER VARYING(10)
house CHARACTER VARYING(1)
firstname CHARACTER VARYING(25)
lastname CHARACTER VARYING(25)
subjects
subjectid CHARACTER VARYING(10)
subjectname CHARACTER VARYING(100)

This is the design for the database. The blue icons indicate foreign key. The gold icons indicate a primary key (multiple keys in one table indicate a composite key). Missing from this diagram are some changes: we created a headmaster table, we created a lock table, and we updated shortattempt and multipleattempt to have an smallint for version number.

Exam State Diagram

The behaviour of the Exam is complex so we include a State diagram to understand it:


Create Exam
Save Exam
Save Exam [no student taking or submitted]
Save Exam [no student taking or submitted]
Unpublished
published and closed
published and open
[time in exam range]
[time not in exam range]
Manually close
Delete Exam
Delete Exam [no student taking or submitted]
Delete Exam [no student taking or submitted]

Exam State Diagram. The condition “time in exam range” means the server time is within the range of the exam’s start and end time.

Features

We attempted to implement as many of the features as possible. However, due to time constraints, some features were not implemented. Here is a list of the missing features (to the best of our knowledge):

Feature	Status
Check that student has not taken exam already	Not implemented
Display one question at a time as student takes exam	Not implemented, instead all questions are displayed
Display multiple choice questions with 2 or more possible answers	Only allows 4 possible answers

Handle pessimistic concurrency	Unable to release the lock if the user hits back before completing transaction, or if browser crashes. Otherwise implemented.
Handle optimistic concurrency for marking	Logical error updating version numbers for all questions not just those that change, otherwise implemented
Handle checking time of exam	Did not implement this in database nor system

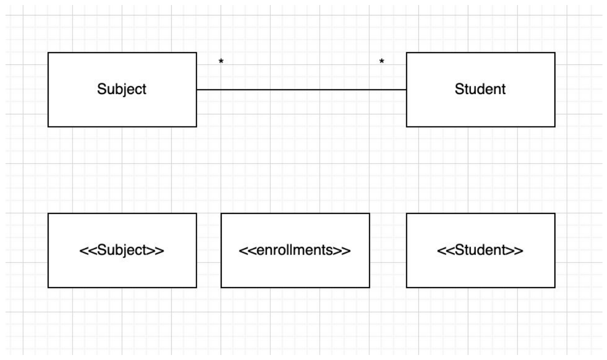
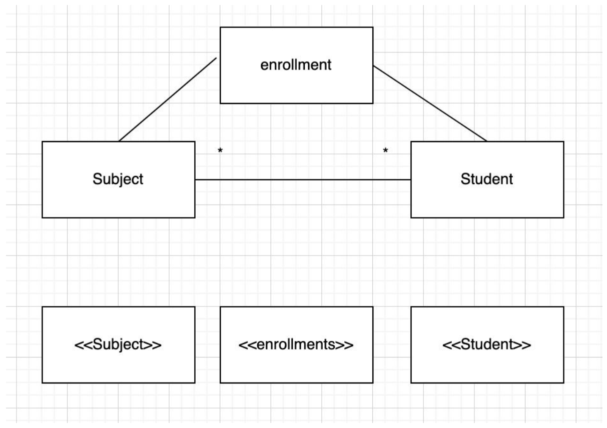
Patterns

In this section we will describe each pattern used, why it was used, how it was implemented, and a sequence diagram.

Association Table Mapping

We used the association table mapping pattern. This pattern is necessary when you have a many to many relationship between objects. This occurs with students and subjects: a student has many subjects, and a subject has many student. Also with teachers: a teacher teaches many subjects, and subjects can have more than one teacher. So in the database we create an association table for each. For the relationship between students and subjects, it was called enrollments. This table has the student ID and the subject ID as well as year and semester of enrollment. Likewise the relationship between teacher and subject has a table called appointments, that includes the teacher's ID and the subject ID as well as year and semester. By including year and semester, we give the system the possibility of showing only subjects a student is currently enrolled in.

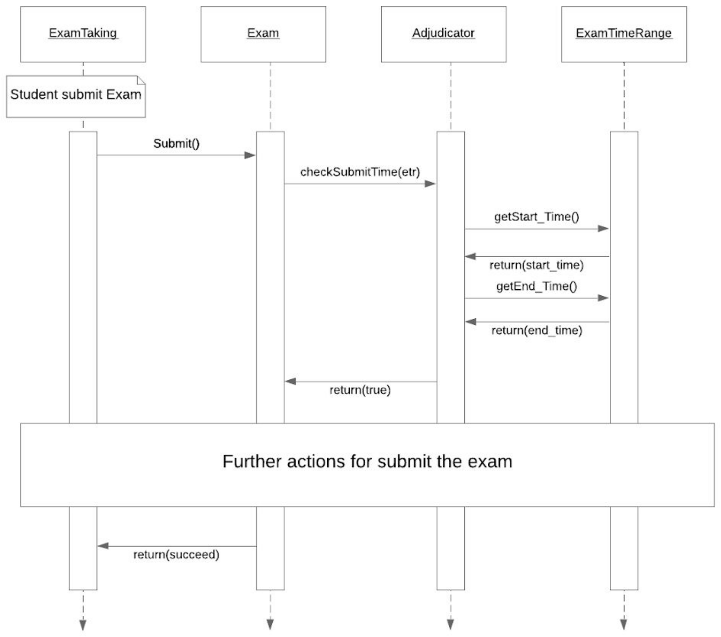
While a relational database cannot handle a many to many relationship, the domain model can. We can just have for say one subject, a linked list of student objects. We don't need an object for each association. However, this is only true in isolation. In practice, Unit of Work requires objects. So if the use cases included say the Admin changing the enrolment of a student from one subject to another, for the unit of work object to handle this, we'd need an object for each association which we can mark as created or deleted. Likewise if the association table appointments contained information such as salary, then we would need an object to mark as dirty if that changed.

No unit of work	With unit of work
 Subject * * Student <<Subject>> <<enrollments>> <<Student>>	 enrollment Subject * * Student <<Subject>> <<enrollments>> <<Student>>

Embedded Value

An embedded value refers to the situation where something exists as a separate object in the domain but is represented in the database as part of an owner table. The separate object is loaded and saved at the same time as the owner table. In this application, this occurs for the start and end time of the exam. We created a separate object for this range that the adjudicator object accepts when the function canStartExam is called.

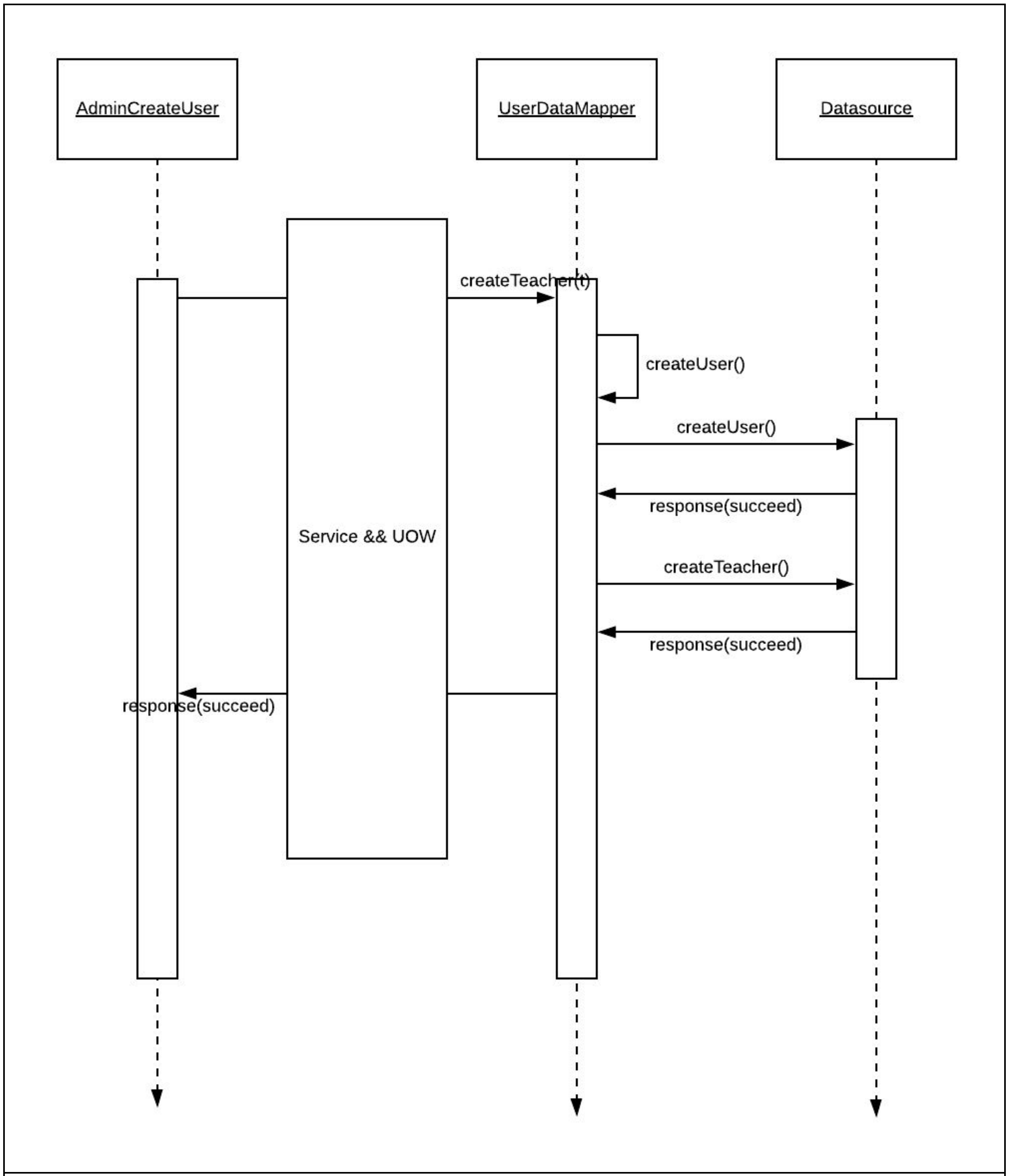
Rather than record start and end time for each exam as a seperate table, it is more efficient to record it as two values of the owner table Exams. We create separate objects for the start and end time due to there being other information that might need to be recorded, such as how many minutes reading time, time zone etc. It’s cleaner to pass an object rather than a large number of values to the adjudicator object, and more decoupled too.



Sequence Diagram for Adjudicator and ExamTimeRange handle students who submit an exam.

Domain Model Pattern

Domain model refers to the pattern of creating an object orientated model. This means we create objects for each student, teacher, exam etc. The alternative patterns are the Transaction script or Table module. Why use the Domain model for this application? If the system was simple, such as simply recording the enrolments for students (and changes to that), then it would make sense to use the Transaction Script pattern. However, the domain for the Exam application is quite complex. We can take advantage of polymorphism, such as having Students, Teachers and Admins be types of Users. This means we can reuse methods. A second advantage is the flexibility it provides dealing with new behaviour.



Sequence Diagram for Create a teacher by admin

Data Mapper

How do we get the data out of the database and into the domain? An important feature is the complexity of the domain. For example, the exam object links to information from both the exam table, the questions tables, the scriptbook table, the attempt tables. There isn't a simple one to one match of domain objects to tables. Therefore the appropriate pattern to choose is the Data Mapper pattern. Our application has SubjectDataMapper, the UserDataMapper and the ExamDataMapper.

The job of these mappers is not just to load data, but to update tables, delete records and create new records.

The datamapper interacts with the service objects, unit of work, identity map and even objects that use lazy loading.

Having a datamapper allows us to decouple the database and domain objects, which makes changing behaviour of the system easier to implement.

The sequence diagram of Unit of Work shows how the system uses the Data Mapper.

Identity Field

How do we say update a student's record? We need a unique key in the student database, otherwise how would the data mapper object know which row to update. In the case of students, we use the studentNumber as the primary key. This is an example of a meaningful key. An example of a table with a meaningless key is the user. We used a serial key (a made up number) for this table because it was possible a system would need teachers that could also be students. For example, say Dumbledore had an Admin account and a Teacher account, but the school only gave him one ID. In that case, we couldn't use a single ID as a key, we would need a compound key of say ID and Type.

We use a lot of compound keys in the database. For example, for a student's attempt at a multiple choice question, the key is made from the student's ID, the year, type and semester of the Exam, the subject ID and the question number. Why use such a messy composite key? Well we can't really trust say teachers to create a unique Exam id themselves, so it is better to make one out of information that when combined should be unique, such as there should be one exam of the same type per subject per semester.

Foreign Key Mapping

Objects relate to other objects: an exam has a number of scriptbooks associated with it, and each scriptbook has a number of attempts, which are themselves related to questions, which are related to exams. How do we tie all this information together in the database? Using foreign key mapping.

See the database diagram for the tables that use foreign key mapping.

Class Table Inheritance (Inheritance Pattern)

Objects implement polymorphism using declarations like extends. This relationship is represented in the database using inheritance patterns. In the case of users, we use class table inheritance. That is, part of a student's information will be stored in the student table, and part of it (username, password etc) will be stored in the user table. Why did we use this pattern instead of Concrete Table Inheritance? We would want the task of users to login to be as quick as possible. So creating a table that just has username, type and password (and necessary foreign keys) is more efficient to load.

Concrete Table Inheritance (Inheritance Pattern)

In the case of questions, there is no need to store information about a single question over separate tables. So in this situation we have a table for multiple choice questions, table for short answer questions, but no table for the generic object question.

Template View (Presentation Pattern)

If we wanted to implement a fancy looking user interface, it might be necessary to use a Transform View pattern. Instead we chose to create a bare bones interface, and this the suitable pattern for this is the Template View.

The disadvantage of the Template View is the code is quite messy. There is an interleaving of static data (HTML) and Java that generates dynamic content.

Page Controller (Inheritance Pattern)

We used the Page Controller pattern. That is, for every view there is a controller that handles the input behaviour.

The reason for using this Page Controller pattern over say the Front Controller is we weren't sure we could implement the ability to store the type of user in a session.

We were going to use a Service layer, that is, a facade that decouples the Presentation layer from the Domain. However, this was not fully implemented.

Unit of Work

How often should data be written to the database? The more frequent, the less efficient the system. Should the whole table be updated, or just the data that has changed? How do we keep track of what has changed?

The Unit of Work object will keep track of what objects are new, what have been deleted, and what are modified. Changes are made in the domain objects only, and then after a trigger (commit) then they are written to the database.

When a commit is made, the unit of work will go through the lists of new, dirty and deleted objects and make the appropriate changes to the database.

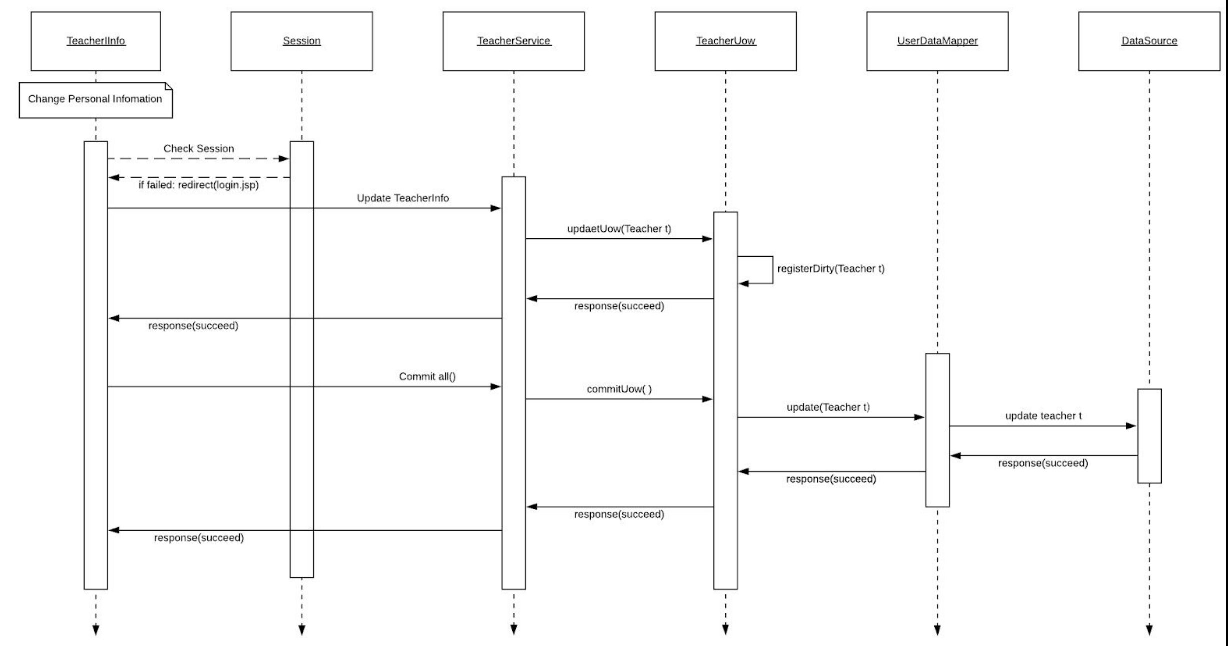
We did not make the unit of work a static class. There is a new unit of work for each session. The reason for this is due to issues stemming from concurrency.

We didn’t keep a list of clean objects instead that can be inferred from the other lists.

We used the caller registration patter, but since the system does not handle requests from multiple users in this stage, the caller registration patten has not been implemented. Lists are going to be saved as a static variable in UOW class.

The pattern is used for teachers, students, subjects, questions and exams.

Commit is called after each transaction or the user presses the button to commit changes. This was not fully implemented.

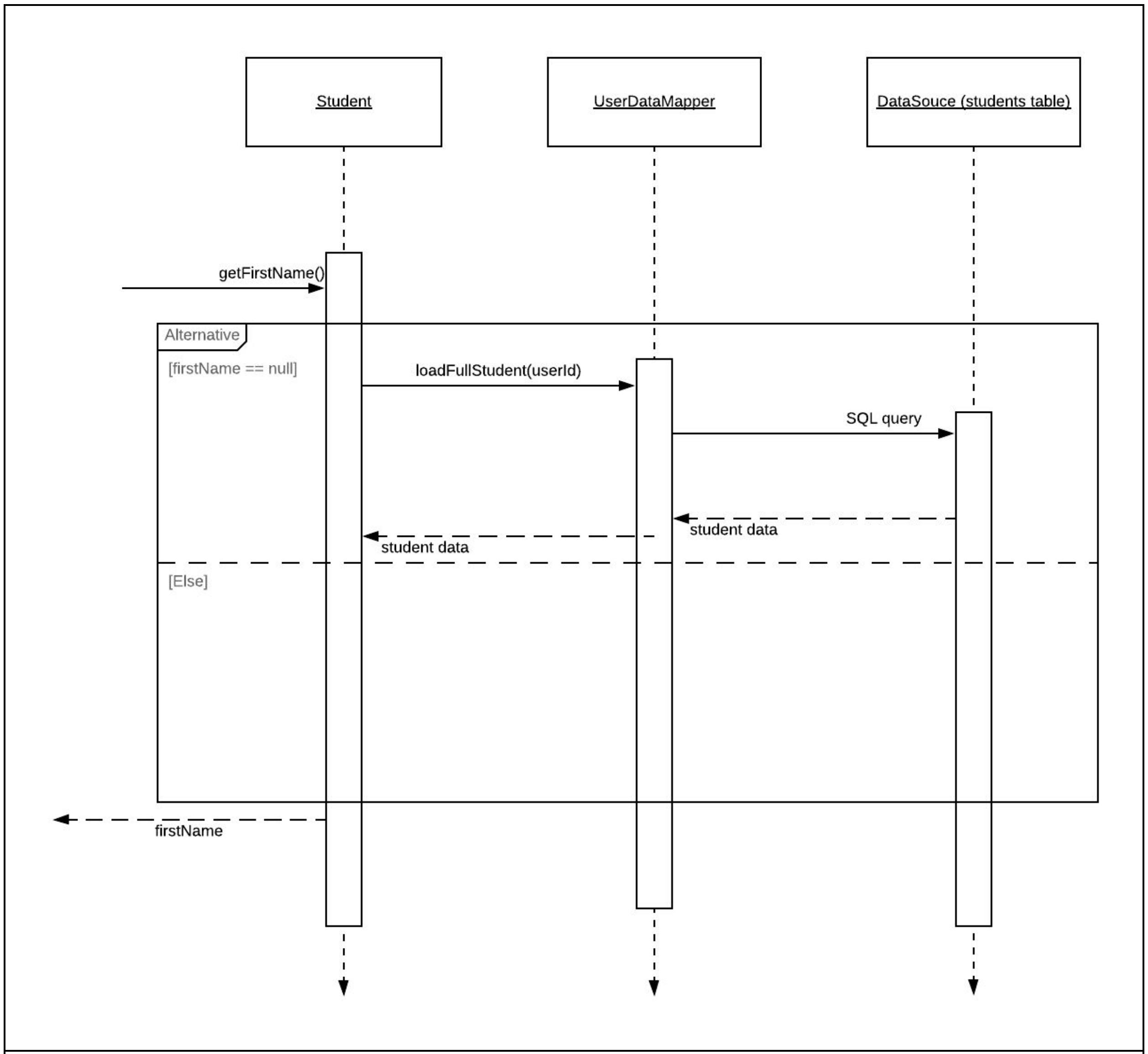


UoW for the Teacher updates personal information. Uow would be the middle of the service layer and the datamapper layer.

LazyLoad

We want the login page to load fast. Would it make sense then to load all the student's information, such as their contact information in case of emergency, or their results history etc? No, so we use lazy loading. The specific pattern we use is Lazy Initialisation. This means we set the fields like first name, last name etc. to null. Only when say the getFirstName() method is called will that method check if the first name is null, and in that case, call a getInfo() method that will call a method in the Data Mapper to retrieve that information from the database.

We were going to use lazy loading for the exams.



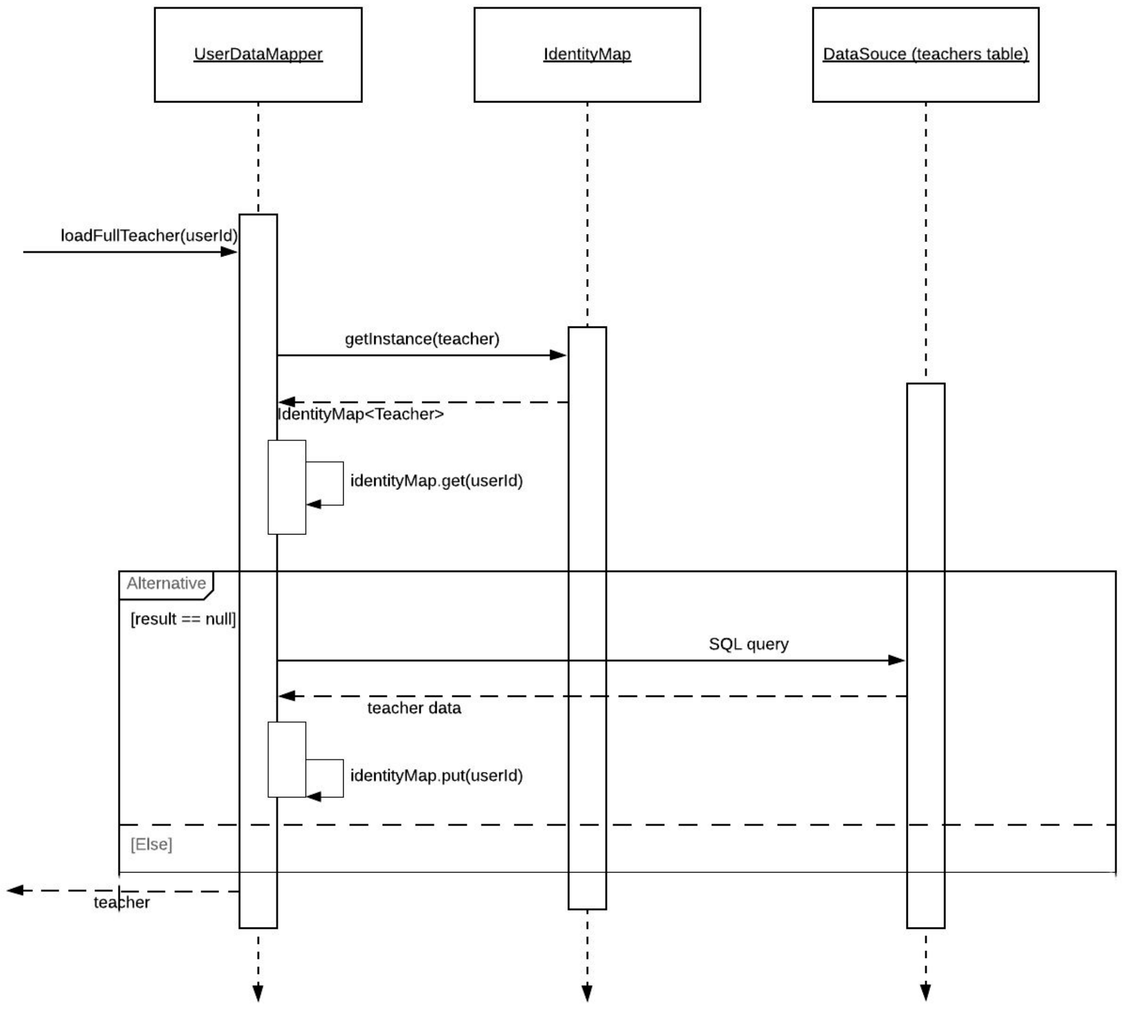
A sequence diagram showing the use of the lazy initialisation. When the method getFirstName is called, it checks if the variable is null, in which case, it calls the User Data Mapper object to fetch the data from the database.

Identity Map

This pattern is used to make sure we don't load the data more than once. This ensures a single source of truth. It is also more efficient.

The identity map gets called by the Data Mapper objects. So for example, let's say the system allows a teacher to see the first and last name of any other instructors teaching the same subject (for example, when viewing who created an exam). Let’s say in a session that teacher A examines a subject B taught by teacher C, and then A later examines subject D which is again taught by C. The UserDataMapper will be able to get from a HashMap using a key of the teacherId the teacher’s object, without reloading from the datasource.

We did not implement this fully yet. While putting information to the identity map is done, checking and getting is not done.



A sequence diagram showing the use of the identity map pattern. Only if the identityMap for teachers does not contain the teacher should the UserDataMapper access the database. This has not been fully implemented.

We needed to update the identity map so it is only used by the unit of work. Otherwise, it operates as a second database, and would muck up concurrency.

However, because we didn't load questions one at a time, unit of work never got used, so identity map never got used on data that wasn't immutable.

Authentication

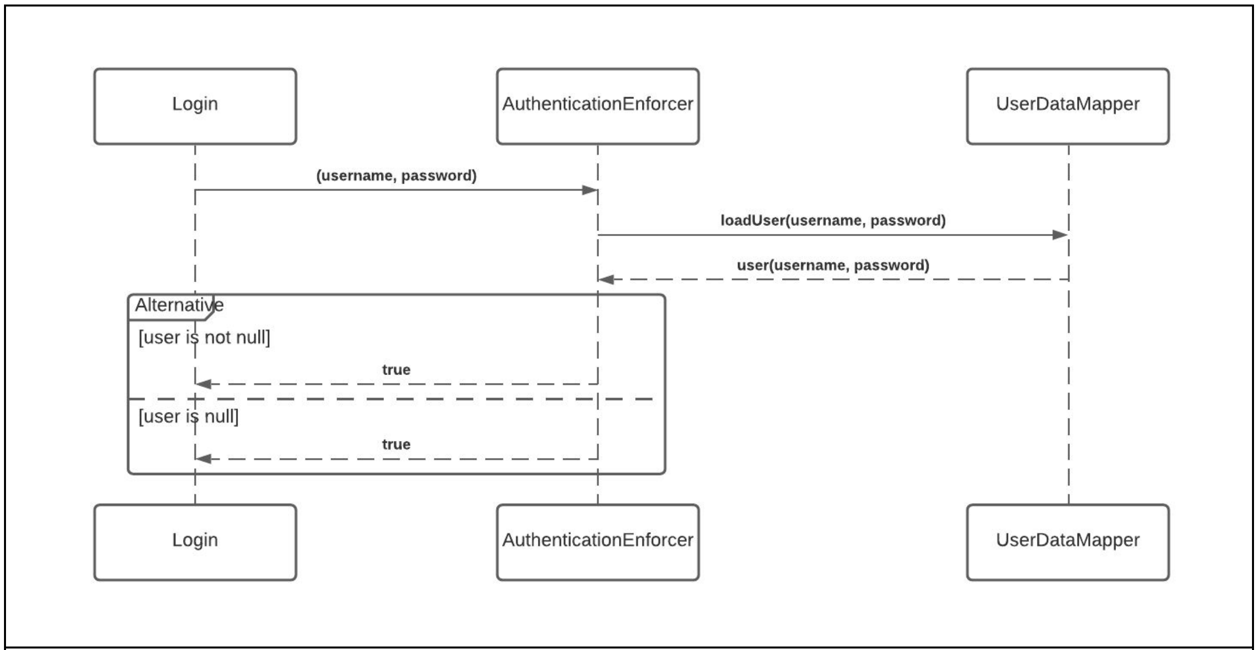
The authentication is handled by the Authentication enforcer in the Security package. When the user tries to login, it checks the database for an existing combination of login/password. The authentication uses the UserDataMapper to get this information. It checks the details for every class of user: teacher, student, headmaster and admin. If the username and password match an entry in the database, the userid is stored on the session using a library (javax.servlet.http.HttpSession). The authentication enforcer then returns true.

If however, they don't match, or the user doesn't exist, it returns false. The login.jsp then displays an alerting notifying the user of the problem.

When authentication is checked later, rather than accessing the database, it simply checks if the userid is stored in the session.

The authentication enforcer by the authorization enforcer, which ensures that each operation is completed by the currently authenticated user. For example, when the user is not authenticated tries to attempt to access the page which requires authentication (by typing in a url), the authentication enforcer will check and notify the user that he/she needs to authenticate first.

The reason for using this pattern is it centralises the code. This means you only have to update the code in one place (eg. if two factor authentication were implemented) when there are multiple points to login.



A sequence diagram showing the Authentication Enforcer when the user is trying to login into the system.

Authorisation

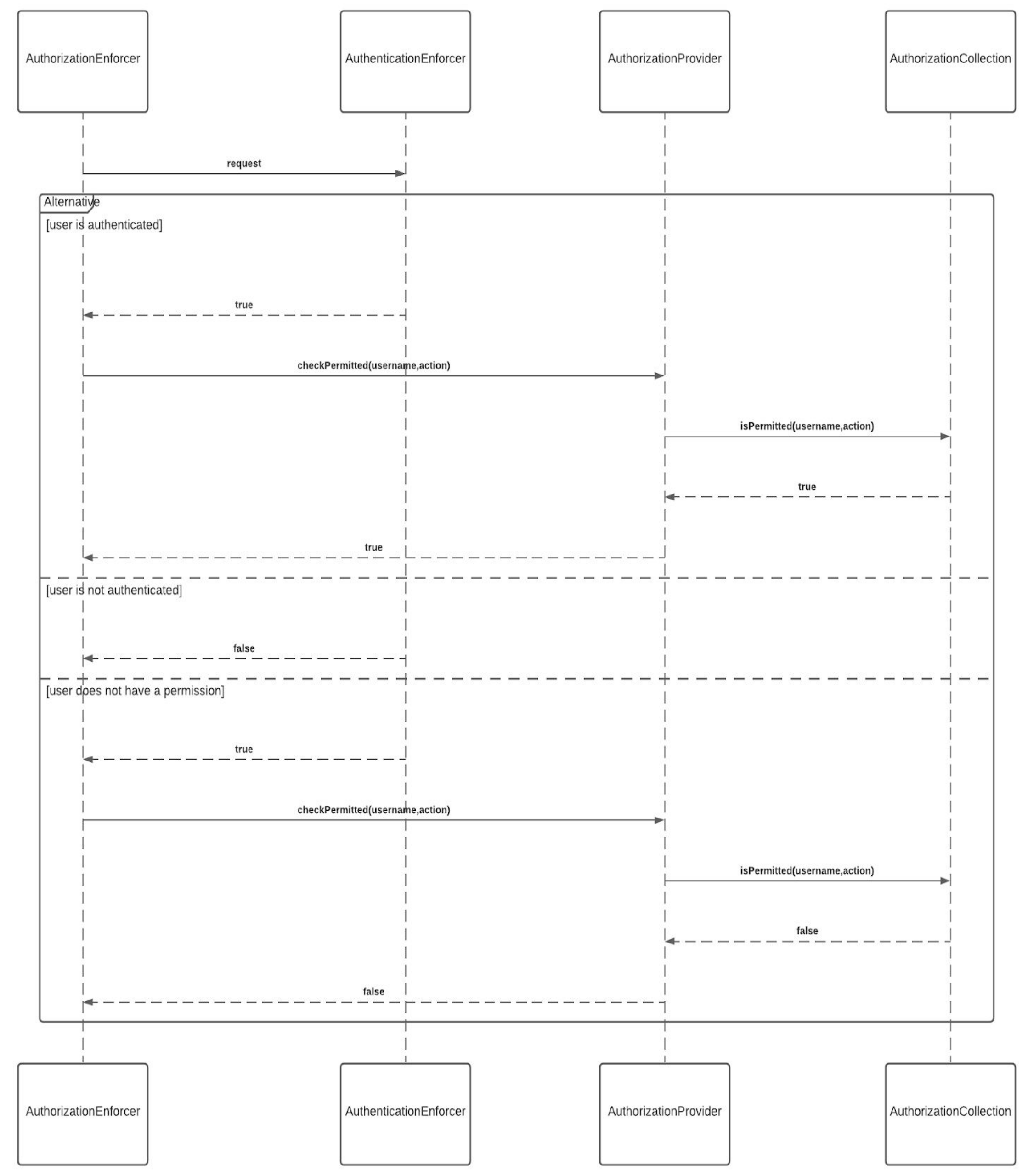
The authorisation is handled by the authorisation enforcer. It checks whether the current user is who they are, and then whether they are permitted to do what they're asking to do.

The permissions collection identifies the operations which are available for each user type. If the user does not have a permission to do a certain operation, it will return *boolean false*.

The Authorisation Provider is the link between the Authorisation Enforcer and Permissions collection. When the Authorisation enforcer tries to identify the authorised user, it sends an information to the Authorisation provider, which identifies if the user has an access to perform the operation.

The Authorisation Enforcer ensures that each operation performed by a user is an authorised one. First, it checks if the user is Authenticated using Authentication enforcer, then it sends the username and requested action to the Authorisation Provider which returns *true* if the user has a permission and *false* otherwise.

The example for the authorization enforcer is the following: the system has only one administrator with username “Dobby”. He is able to create a new subject and see the subjects as per administrator requirements. However, the other user is Dumbledore, whose user type is ‘head master’. As a headmaster, Dumbledoor is able to see the subject's information, but cannot create a new subject like an administrator.

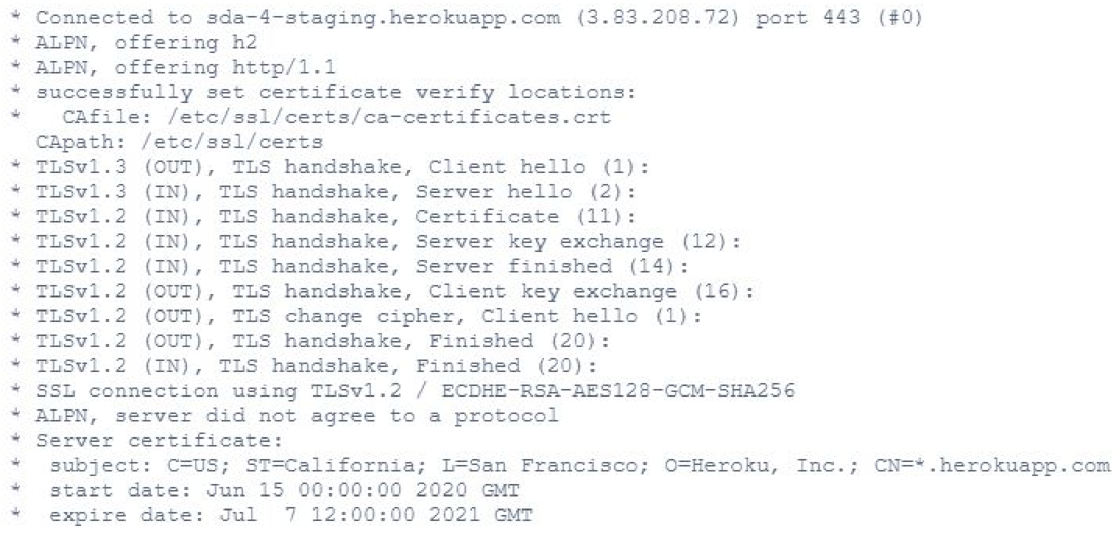


A sequence diagram showing the use of the Authorisation Enforcer. It handles the cases when the user is not logged in, and when the user has / does not have a permission to use the operation

The advantage of this pattern is that it centralises the permissions data in one location. This means if you wanted to changed Dumbledore’s permissions so that he could now create subjects, then you would only have to add “Create Subject” to the headmaster permissions List in PermissionsCollection,java. It decouples authorisation from the presentation layer, allowing for easier updates, but also allowing different users to share the same screen layouts.

Security Pipe

We have to include the TLS protocol to ensure the protection of the data sent over an unsecured network. This protocol allows us to achieve secrecy, proper authentication and integrity. The TLS protocol used for the deployed application is provided by Heroku. It uses Heroku SSL protocol to provide the security pipe which is an extension of the TSL, and comes for all the deployed applications on Heroku.
The security pipe can be tested using Heroku command line:
curl -vI https://sda-4-staging.herokuapp.com/
The output is the following:



```
* Connected to sda-4-staging.herokuapp.com (3.83.208.72) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/certs/ca-certificates.crt
  CApath: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* ALPN, server did not agree to a protocol
* Server certificate:
*  subject: C=US; ST=California; L=San Francisco; O=Heroku, Inc.; CN=*.herokuapp.com
*  start date: Jun 15 00:00:00 2020 GMT
*  expire date: Jul  7 12:00:00 2021 GMT
```

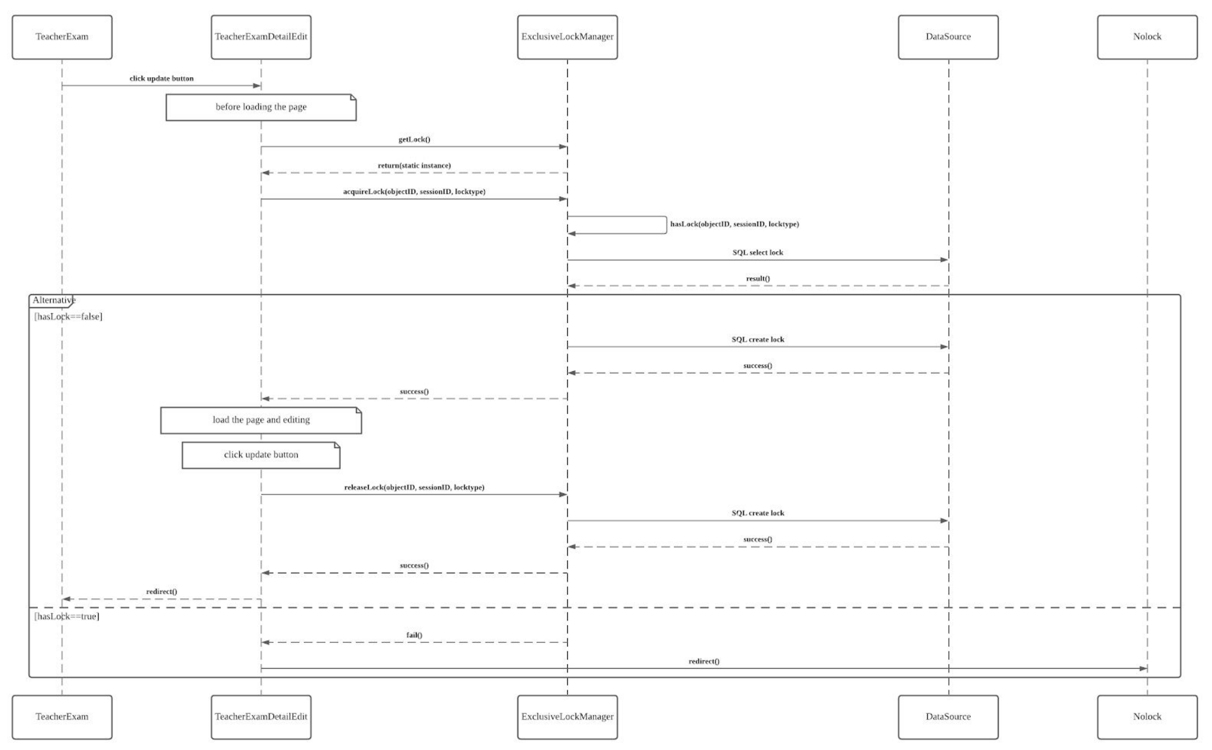
Concurrency

We adopted the Pessimistic Offline Lock to handle concurrency except for marking scriptbooks.

Pessimistic Offline Lock is easy to understand and use. It works by allowing only one person to access the data at a time, which avoids issues like lost updates. For example, we implement it to prevent two teachers from editing the exam details at the same time. In this way, it is an exclusive write lock, because students and teachers can still see those details. Unfortunately, this means there can still be inconsistent reads.

The teachers acquire the locks when they access deleting, updating or editing questions of exams. They release the lock when they finish the transactions, such as clicking to confirm delete. See the files Delete.jsp, TeacherExamQuestions.jsp, TeacherExamDetailEdit.jsp for acquiring the locks, and DeleteExam.java, UpdateExam.java and DoneEditingExamQuestions.java for releasing the locks.

We also release the locks when the user logs out of a session.

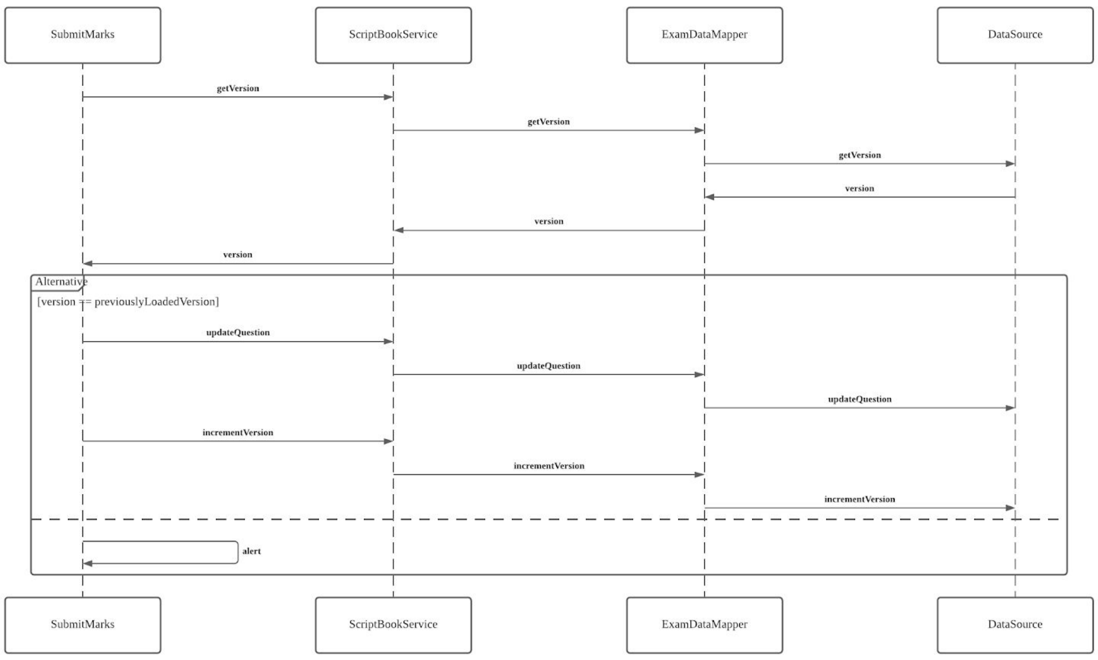


A sequence diagram showing the use of the Pessimistic Offline Lock in when the Instructor is trying to update details of the exam.

The pessimistic lock has liveness issues. In other words, if you imagine that there were a number of teachers who divided up the marking of exams, such as Snape marking the short answer, and another teacher marking the multiple choice questions. It would be frustrating if there were a read/write lock. Instead, given conflict should be unlikely, we should use optimistic lock.

Optimistic lock requires keeping a version number for each attempt in the database. When a teacher updates a question, the version number should be incremented. Then when another teacher, who acquired his information before the increment, goes to update, he will be informed of the conflict, and in our system, prevented from updating that data.

While we implemented most of this pattern we did not have enough time to successfully finish it. The main issue is we weren't keeping track of which questions the teacher had changed, so we were incrementing to many version numbers, which effectively made it an exclusive write lock.



A diagram showing optimistic lock. The database entry only gets updated if the version number of the question when loaded at the beginning of the transaction matches the version when trying to update. If they match, then the update to database occurs, and the version number is also incremented. Otherwise, an alert is sent to the user informing them of the conflict.

Testing

We used manual tests. Tests are included in the Appendix. We only tested functional requirements, not performance of the system.

Features not fully implemented

The Identity Map pattern was not fully implemented, only putting information on the identity map and not checking before loading.

The use cases related to creating questions and marking questions were not fully implemented.

The unit of work was not fully implemented.

Lazy loading was only used for users and not fully implemented for exams.

Not immigrate all the newest changes and make them work.

Known Bugs

Can’t login to administrator account even with correct details.

Login

Login information so that marker can test our system themselves:

Username	Password	Account Type
HarryP	Abra	Student
HerG	Dobb	Student
RonW	Broken	Student
SSnape	ih8harry	Teacher
DracoM	ih8harry2	Student
Dobby	iluvharry	Admin
Dumb	Light	Headmaster

MM	cat	Teacher

Appendix

Title:	Lazy Initialisation 1
Test Case Description:	To test Lazy Initialisation is working, verify some of the student information is set to null.
Test Case Instructions:	Disable lazy loading. Load the page that displays the user information that is specific to the student (first name, last name etc).
Test Case Data:	
Expected result:	See null for fields first name, given name.
Actual result:	See null for fields first name, given name.
Pass/Fail:	Pass

Title:	Lazy Load Test 1
Test Case Description:	To test Lazy loading is working,
Test Case Instructions:	load the page that displays the user information that is specific to the student (first name, last name etc).
Test Case Data:	
Expected result:	See all information
Actual result:	See all information

Pass/Fail:	Pass

Title:	Data Mapper Test 1 (On fmme branch)
Test Case Description:	To test User Data Mapper loadAllUsers(), um.getAllStudents(), getAllTeachers(), loadFullStudent(String userNumber), loadFullTeacher(String userNumber), loadAllSubject() works
Test Case Instructions:	load the Admin page, which displays all the students, teachers and subjects information.
Test Case Data:	
Expected result:	See all information
Actual result:	See all information
Pass/Fail:	Pass

Title:	Data Mapper Test 2
Test Case Description:	To test User Data Mapper loadSubjectByStudent(Student s) is working
Test Case Instructions:	Login student account, subjects that the student enrolled should be shown.
Test Case Data:	
Expected result:	See all information
Actual result:	See all information
Pass/Fail:	Pass

Summary:

Passed Test	Failed Test
1,2,3,4,5,6,7,8,9,10,12,13,14,15,16,18	11,17

Individual Test:

TestNumber	1
Test Name	Login Test for Student
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Login using student username(HarryP) 2.Login using student password
Expect result	Login Successful and could browse all the subjects the student has enrolled.
Actual result	Login Successful and could browse all the subjects the student has enrolled.
Status	Pass

TestNumber	2
Test Name	Login Test for Teacher
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Login using teacher username(SSnape) 2.Login using teacher password
Expect result	Login Successful and could browse all the subjects the teacher has appointed.
Actual result	Login Successful and could browse all the subjects the teacher has appointed.
Status	Pass

TestNumber	3
Test Name	Login Test for Administrator
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Login using Administrator username(Dobby) 2.Login using administrator password
Expect result	Login Successful and could browse all the Students, Teachers and Subject information.
Actual result	Login Successful and could browse all the Students, Teachers and Subject information.
Status	Pass

TestNumber	4
Test Name	Student Inspect Exam
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Logged in as student HarryP 2.Click the exams button after the subject Defense against the Dark Arts.
Expect result	Student could inspect all the exams have been published
Actual result	Student could inspect all the exams have been published
Status	Pass

TestNumber	5
Test Name	Student take the exam

Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Logged in as student HarryP and click the exams button after the subject Defense against the Dark Arts. 2.Click Attempt button after the exam Defense Against the Dark Arts End of Magic End of Semester Exam. 3.Student select all multiple questions and answers all short answer questions, and then clicks submit. 4.Student click submit in the alert box.
Expect result	The exam is successfully submit
Actual result	The exam is successfully submit
Status	Pass

TestNumber	6
Test Name	Teacher Create Exam
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Logged in as Teacher SSnape 2.Click the exams button after the subject Defense against the Dark Arts. 3.Click ‘add’ button 4.Input Year(2020), Semester(2), Exam Name(Quiz), total mark(5), and select Exam type(M) for the new exam. 5.Click ‘add exam’ button
Expect result	The exam is created and shows in the exam

	inspect page.
Actual result	The exam is created and shows in the exam inspect page.
Status	Pass

TestNumber	7
Test Name	Teacher add questions to the exam
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Logged in as Teacher SSnape 2.Click the exams button after the subject Defense against the Dark Arts. 3.Click ‘edit’ button in the Quiz exam, questions attribute 4.Click the ‘add’ button after Multiple Choice questions. 5.Input Question Text (What is the most common way to defend dark arts?), Answer A (Physical defence), Answer B(Block it by Magic shield), Answer C(Use Light magic to hit back), Answer D(Try to avoid encounters Dark Arts), Marks (1) and select the answer (D). 6. Click the submit button. 7.Click the ‘add’ button after Short Questions. 8.Input Question Text (List at least three Dark Arts.) and marks (2). 9. Click the submit button. 10.Click Done button to save the exam.
Expect result	The questions are created and shown in the Questions page.

Actual result	The questions are created and shown in the Questions page.
Status	Pass

TestNumber	8
Test Name	Teacher edit the question
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Click the edit button in the short questions List at least three Dark Arts. 2.Change the mark to 3. 3.Click submit button
Expect result	The mark for the question is changed
Actual result	The mark for the question is changed
Status	Pass

TestNumber	9
Test Name	Teacher delete question
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Click the delete button in the short questions List at least three Dark Arts.
Expect result	The question is deleted and would not be shown in the question page.
Actual result	The question is deleted and would not be shown in the question page.
Status	Pass

TestNumber	10
Test Name	Teacher update exam details
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Click ‘Update’ button in the Quiz exam, Details attribute. 2.Change the Total Marks to 15. 3.Click the ‘update exam details’ button
Expect result	The mark has changed.
Actual result	The mark has changed.
Status	Pass

TestNumber	11
Test Name	Teacher delete exam
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Click ‘delete’ button in the Quiz exam, Details attribute. 2.Click ‘delete exam’ exam in Delete Exam Information page
Expect result	The exam is deleted and would not be shown in the exam inspect page.
Actual result	The exam is deleted but has not been redirected to the exam inspect page. http://sda-4-staging.herokuapp.com/DeleteExam
Status	Fail

TestNumber	12
Test Name	Teacher marking the exam
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Click ‘scriptbooks’ button in the Defense Against the Dark Arts End of Magic End of Semester Exam exam, Marking attribute. 2.Click ‘load’ exam in Student S10, Load attribute. 3.Give marks for all the questions. 4. Click ‘Submit’ button 5.Click submit in the alert box. 6.Click set to give final marks 7.Type in the value at the summed total field.
Expect result	The marks have been given and shown in the marking page.
Actual result	The marks have been given and shown in the marking page.
Status	Pass

TestNumber	13
Test Name	Administrator create subject
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Click ‘create subject’’ button. 2.Input Code(DDM110), Title(Defend Dark

	Magic), Year(2020), semester(2) for the subject and select harryP and one SSnape. 3.Click Add Subject button.
Expect result	The subject is created and shown.
Actual result	The subject is created and shown.
Status	Pass

TestNumber	14
Test Name	Administrator view subject
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Click ‘View’’ button in the subject DEF101.
Expect result	The teachers, students and exams could be viewed.
Actual result	The teachers, students and exams could be viewed.
Status	Passed

TestNumber	15
Test Name	Two student take the exam at the same time
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Logged in as student HarryP and clicked the exams button after the subject Defense against the Dark Arts. 2.HarryP clicks the Attempt button after the exam Defense Against the Dark Arts End of Magic End of Semester Exam.

	3.Logged in as student HerG and clicked the exams button after the subject Defense against the Dark Arts. 4.HerG clicks the Attempt button after the exam Defense Against the Dark Arts End of Magic End of Semester Exam. 5.HerG answers all the questions and clicks submit. 6.HarryP answers all the questions and clicks submit. 7.HarryP click submit in the alert box. 8.HerG click submit in the alert box.
Expect result	The exams for HarryP and HerG are successfully submit.
Actual result	The exams for HarryP and HerG are successfully submit
Status	Pass

TestNumber	16
Test Name	Two teachers update the exam at the same time
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Logged in as teacher SSnape and clicked the exams button after the subject Defense against the Dark Arts. 2.SSnape clicks the update button after the exam Defense Against the Dark Arts End of Magic End of Semester Exam. 3.Logged in as teacher MM and clicked the exams button after the subject Defense against the Dark Arts.

	4.MM clicks the update button after the exam Defense Against the Dark Arts End of Magic End of Semester Exam. 5.MM see the conflict page. 6.SSnape changes the mark to 10. 7.SSnape clicks submit. 8.MM clicks update button again 9.MM could access the update page.
Expect result	In the 4th step MM is redirected to the lock page. The mark has been changed by SSnape. And MM can view it when she gets access to the update page.
Actual result	In the 4th step MM is redirected to the lock page. The mark has been changed by SSnape. And MM can view it when she gets access to the update page.
Status	Pass

TestNumber	17
Test Name	Two teachers marking the same exam
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Logged in as teacher SSnape and clicked the exams button after the subject History of Magic. 2.SSnape clicks the update button after the exam History of Magic End of Semester Exam. 3.Logged in as teacher MM and clicked the exams button after the subject History of Magic.

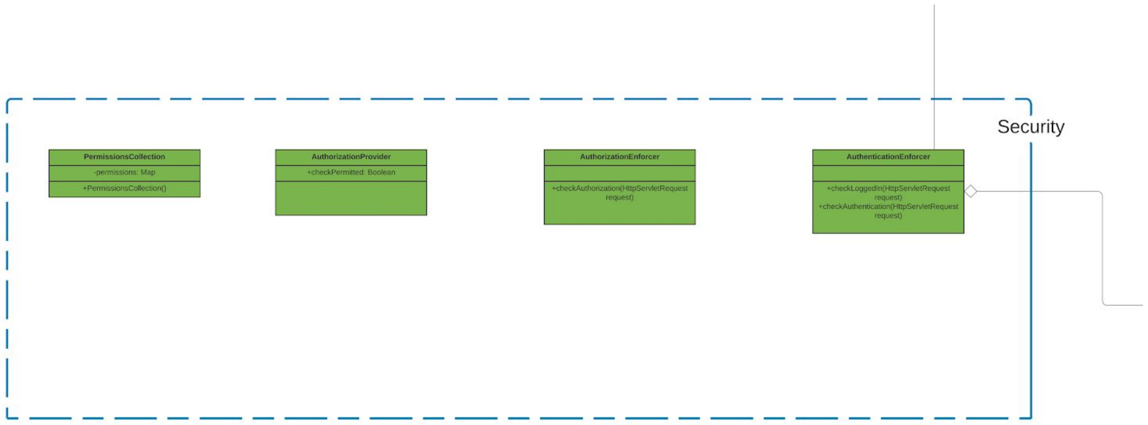
	4.MM clicks the update button after the exam History of Magic End of Semester Exam. 5.SSnape clicks the load button to load the exam done by S12. 6.MM clicks the load button to load the exam done by S12 7.SSnape marking all the questions and clicking submit. 8.MM marking all the questions and click submit. 9.SSnape click submit in the alert box to save the mark. 10.SSnape will be redirected to the conflict page. 11.SSnape logout. 12.MM click submit in the alert box to save the mark.
Expect result	Only the mark MM changed is saved.
Actual result	MM has also been blocked.
Status	Fail

TestNumber	18
Test Name	Student try to view Administrator page
Who conducted the test	Fanyu
Date conducted	1/11/2020
Steps	1.Login as HarryP. 2.Input

	https://sda-4-staging.herokuapp.com/Admin.jsp on the bowser 3.Log out HarrP and log in with MM. 4.Input https://sda-4-staging.herokuapp.com/Admin.jsp on the bowser
Expect result	Harry and MM cannot view the Administrator page and be redirect to the permission information page.
Actual result	Harry and MM cannot view the Administrator page and be redirect to the permission information page.
Status	Pass

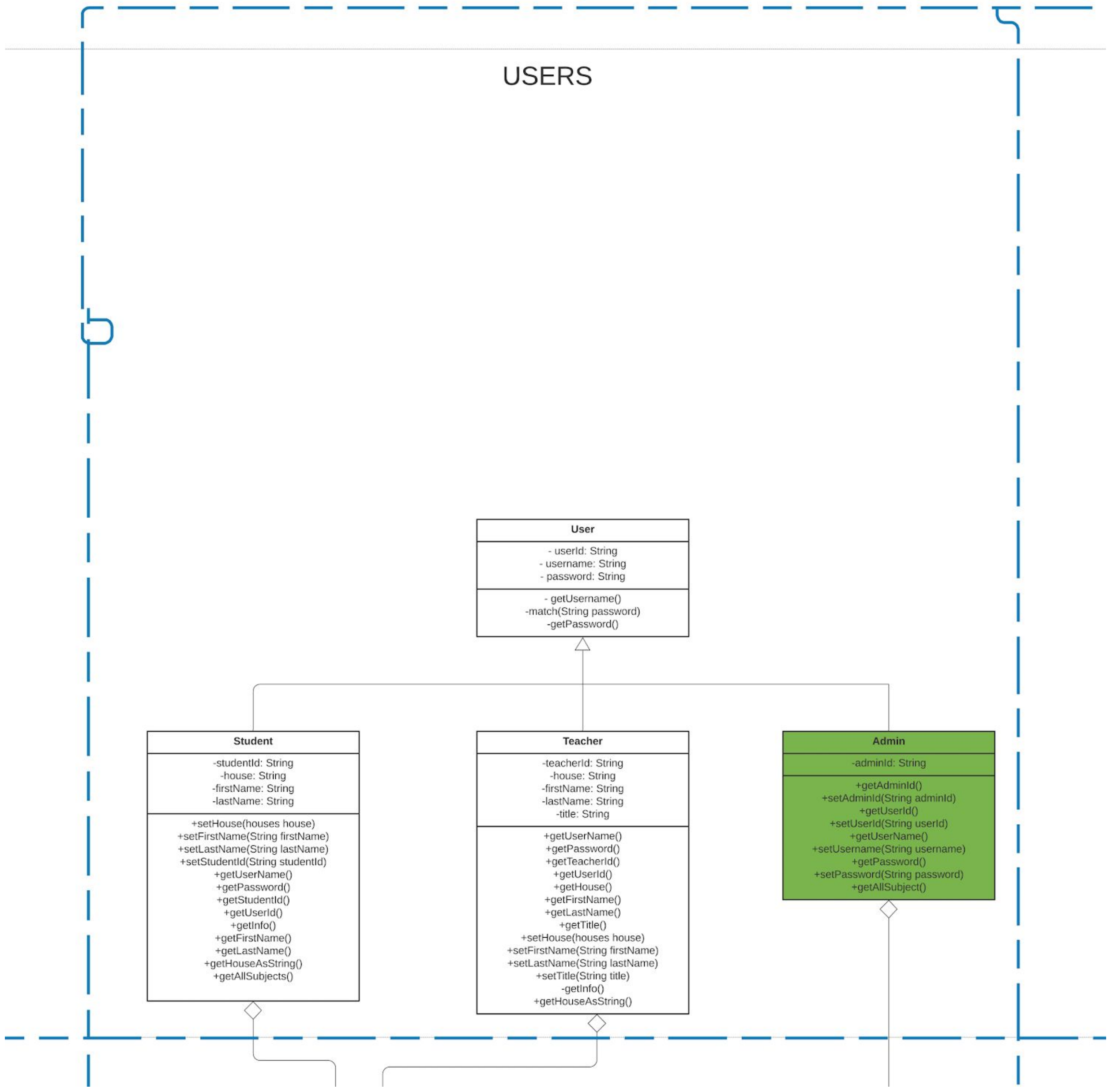
Class Diagram Individual

Security level

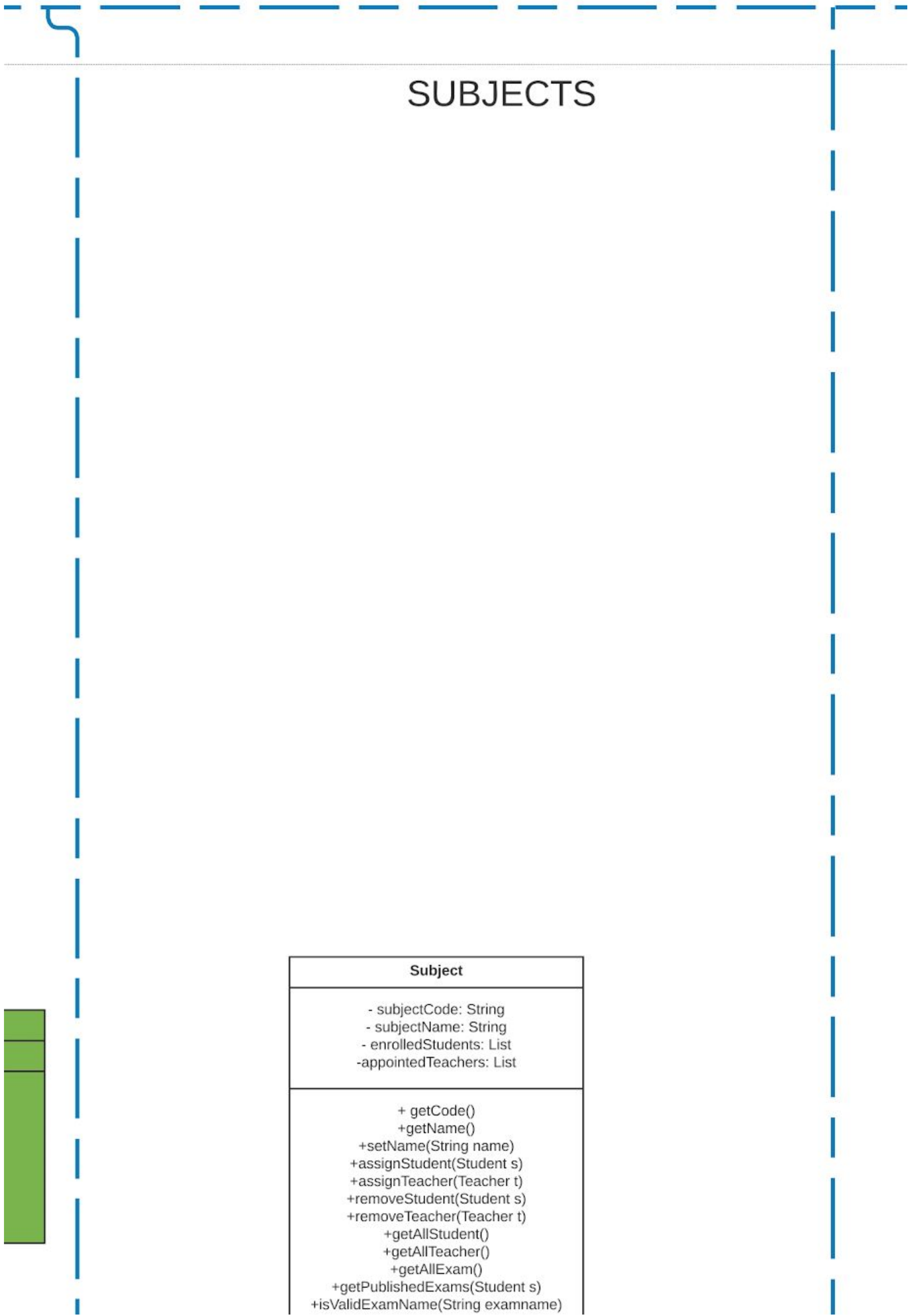


Domain level

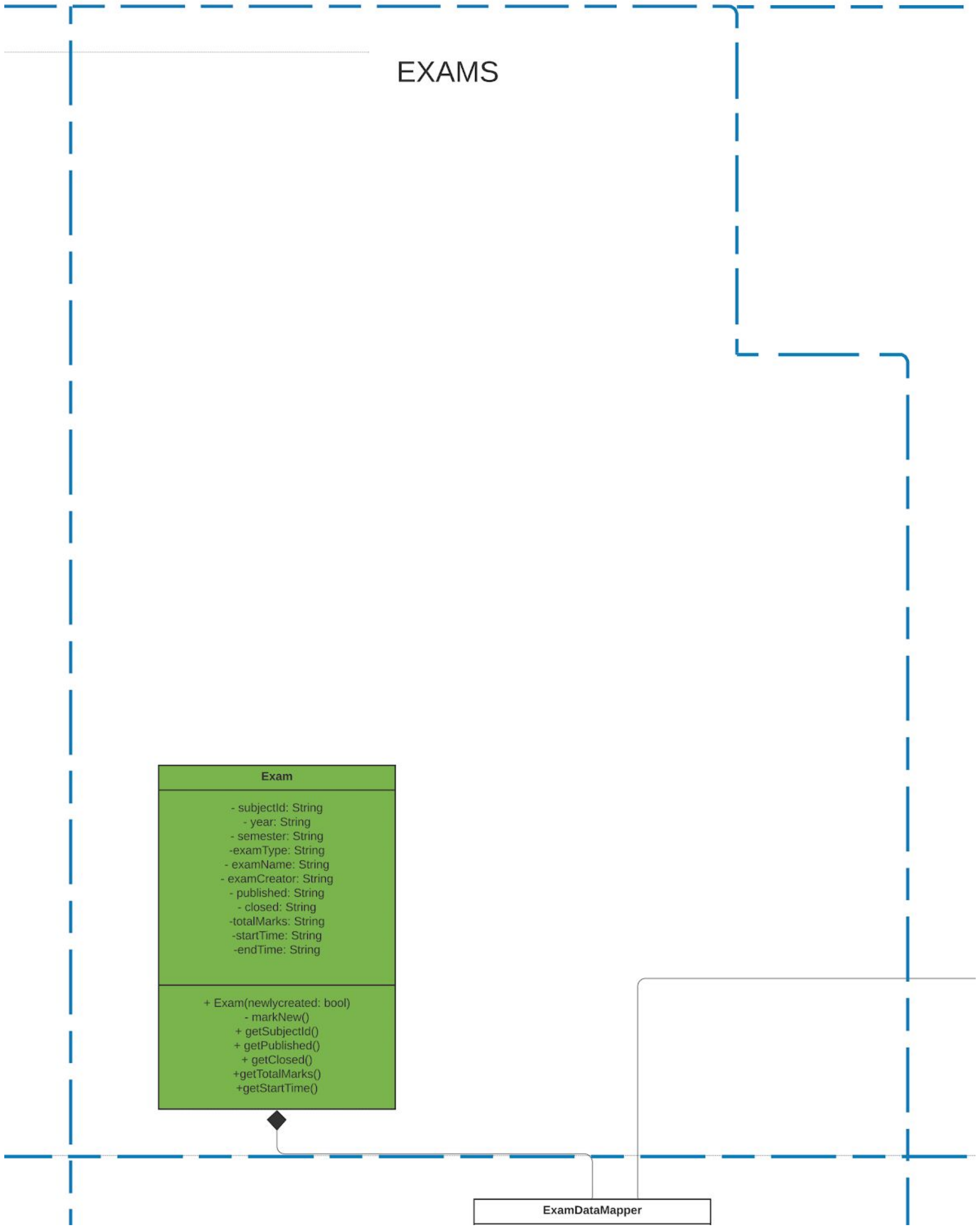
1. User Domain



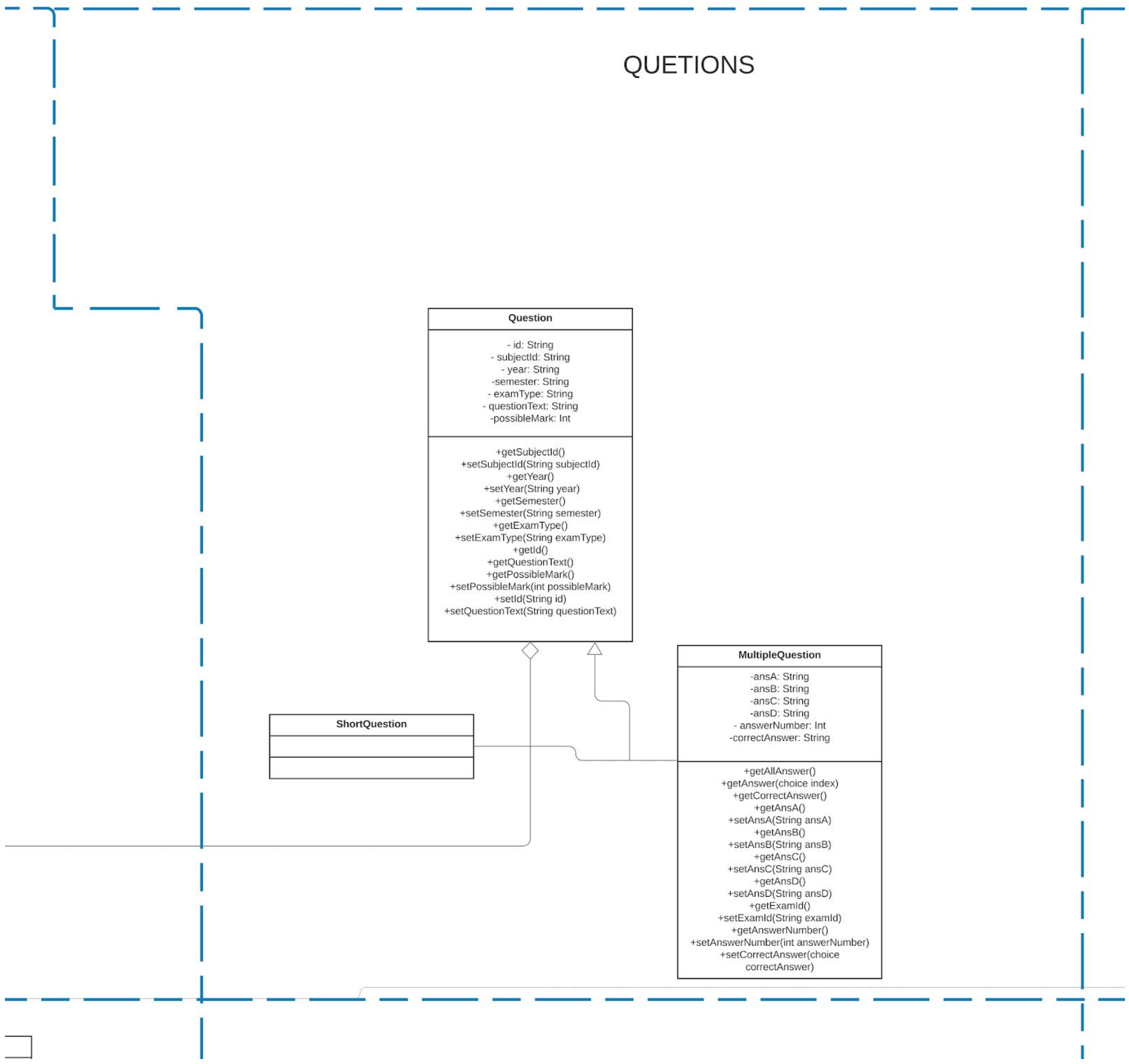
2. Subject Domain


SUBJECTS
Subject
- subjectCode: String
- subjectName: String
- enrolledStudents: List
-appointedTeachers: List
+ getCode()
+getName()
+setName(String name)
+assignStudent(Student s)
+assignTeacher(Teacher t)
+removeStudent(Student s)
+removeTeacher(Teacher t)
+getAllStudent()
+getAllTeacher()
+getAllExam()
+getPublishedExams(Student s)
+isValidExamName(String examname)

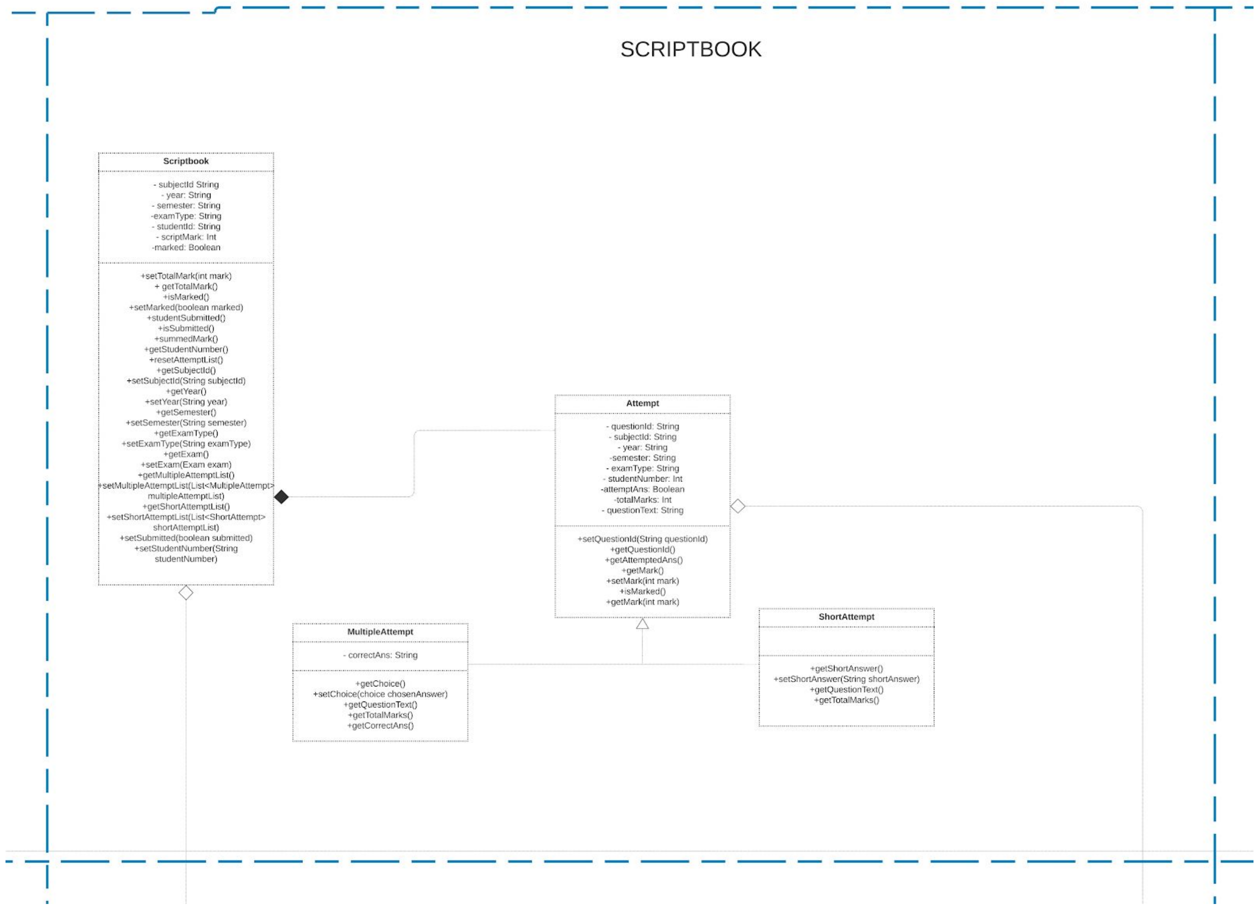
3. Exam Domain


EXAMS
Exam
- subjectId: String
- year: String
- semester: String
-examType: String
- examName: String
- examCreator: String
- published: String
- closed: String
-totalMarks: String
-startTime: String
-endTime: String
+ Exam(newlycreated: bool)
- markNew()
+ getSubjectId()
+ getPublished()
+ getClosed()
+getTotalMarks()
+getStartTime()
ExamDataMapper

4. Question Domain

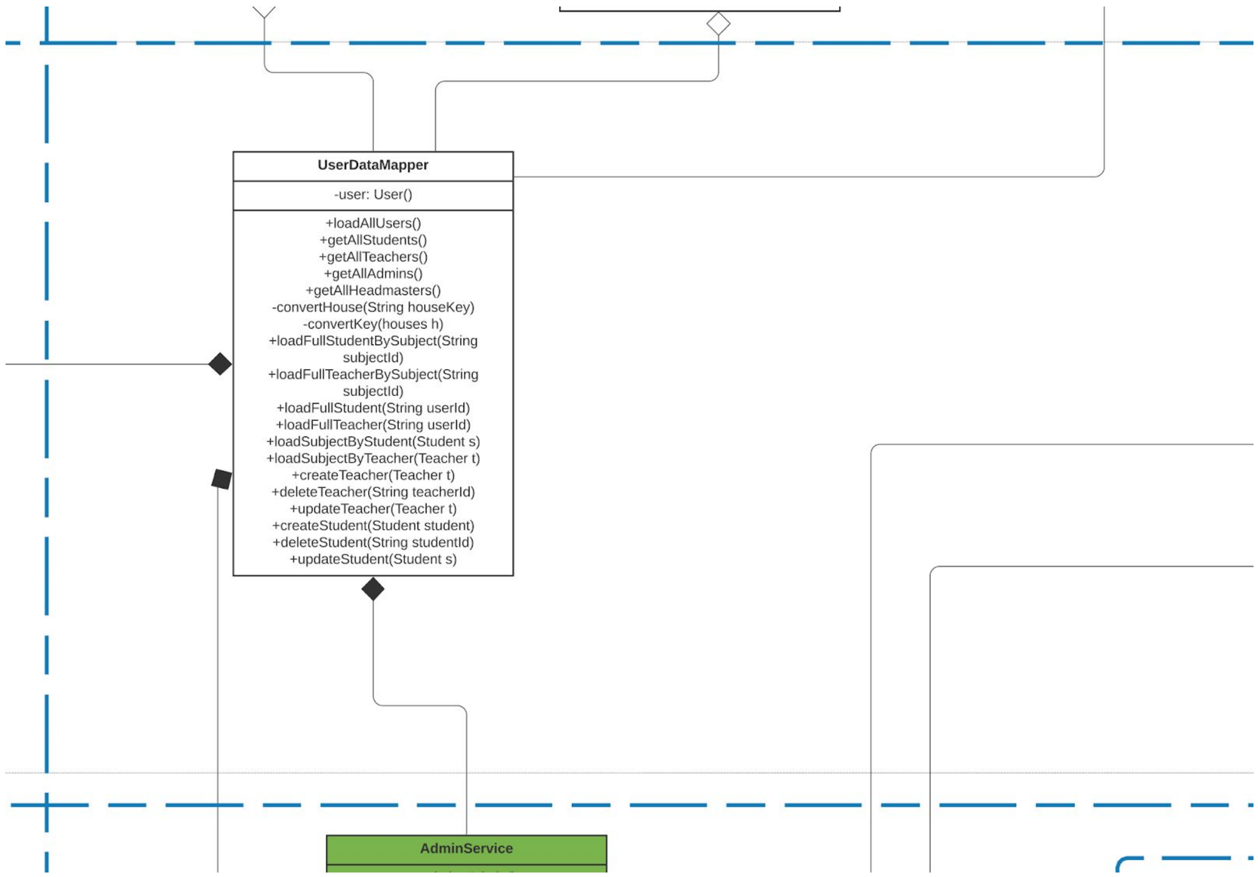


5. Scriptbook Domain

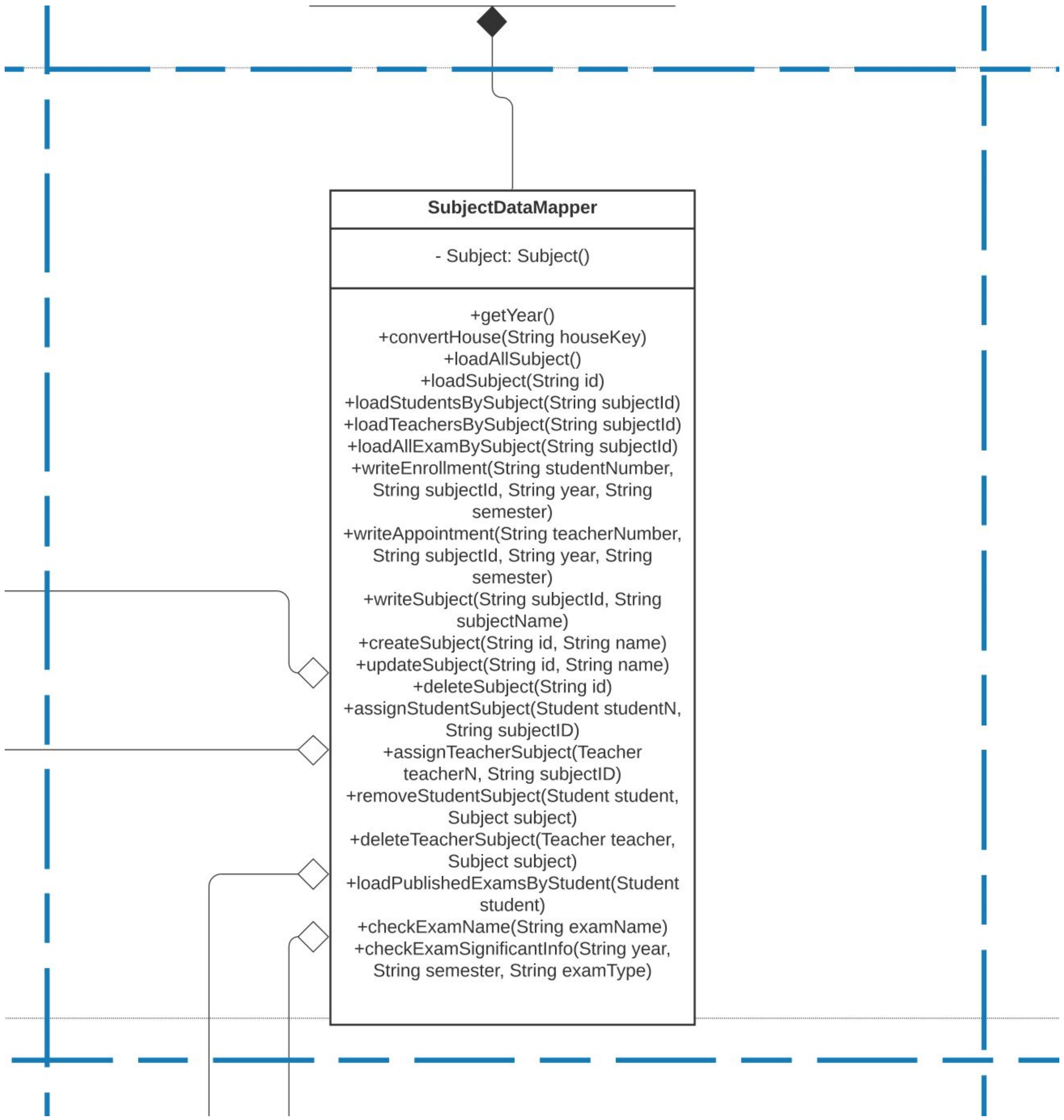


Data Source level

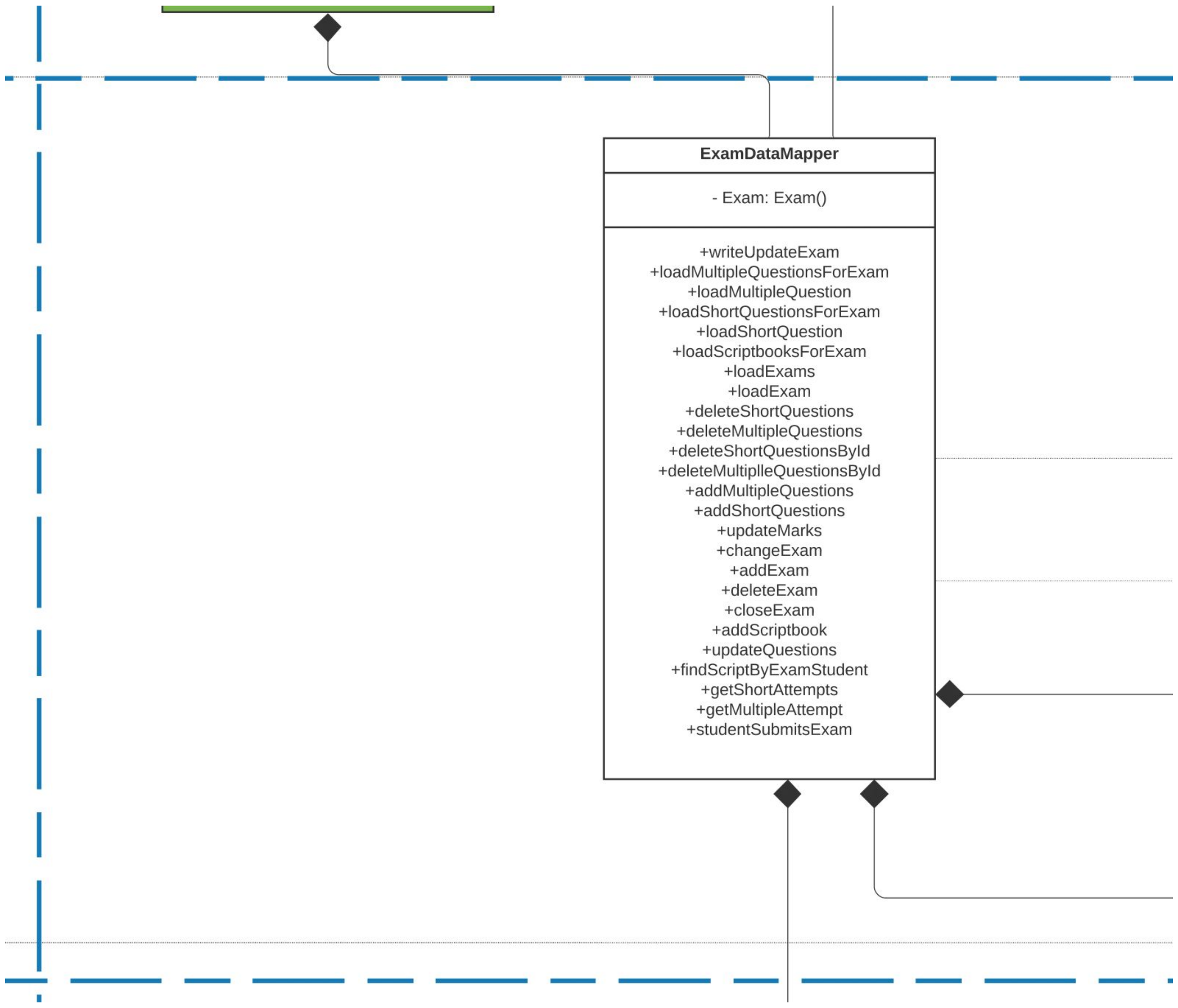
1. User Datamapper


UserDataMapper
-user: User()
+loadAllUsers()
+getAllStudents()
+getAllTeachers()
+getAllAdmins()
+getAllHeadmasters()
-convertHouse(String houseKey)
-convertKey(houses h)
+loadFullStudentBySubject(String subjectId)
+loadFullTeacherBySubject(String subjectId)
+loadFullStudent(String userId)
+loadFullTeacher(String userId)
+loadSubjectByStudent(Student s)
+loadSubjectByTeacher(Teacher t)
+createTeacher(Teacher t)
+deleteTeacher(String teacherId)
+updateTeacher(Teacher t)
+createStudent(Student student)
+deleteStudent(String studentId)
+updateStudent(Student s)
AdminService

2. Subject Datamapper

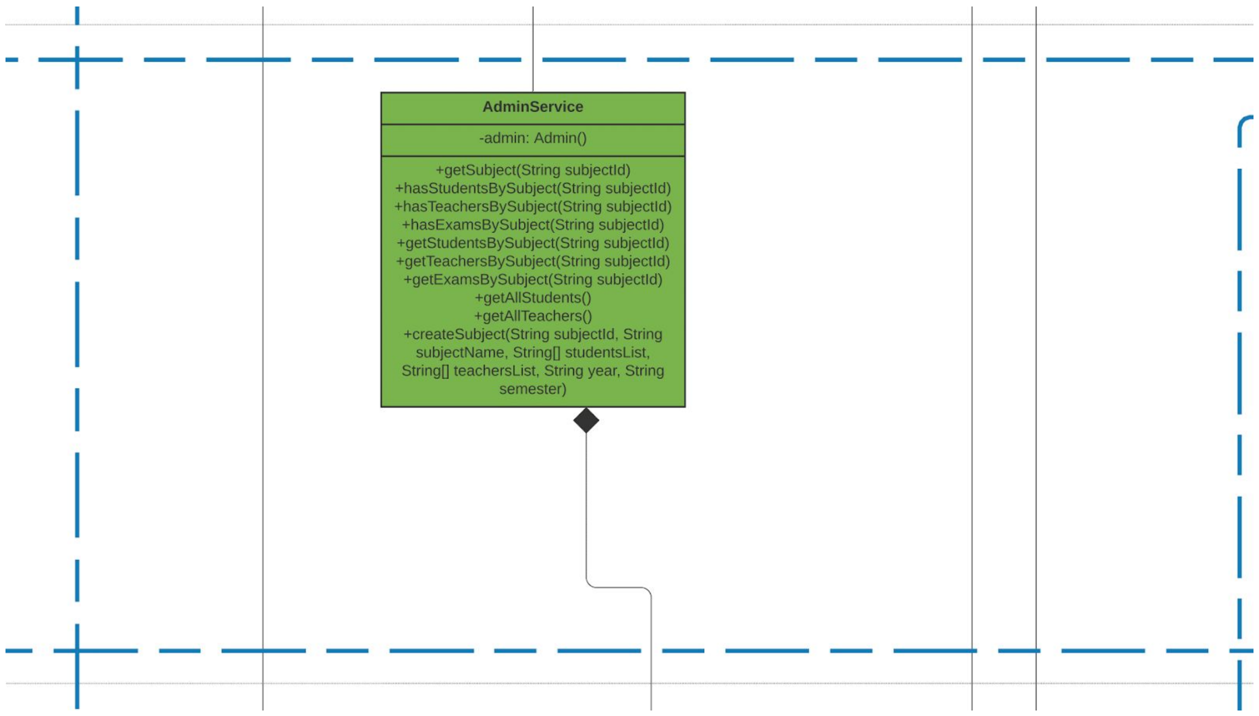

SubjectDataMapper
- Subject: Subject()
+getYear()
+convertHouse(String houseKey)
+loadAllSubject()
+loadSubject(String id)
+loadStudentsBySubject(String subjectId)
+loadTeachersBySubject(String subjectId)
+loadAllExamBySubject(String subjectId)
+writeEnrollment(String studentNumber, String subjectId, String year, String semester)
+writeAppointment(String teacherNumber, String subjectId, String year, String semester)
+writeSubject(String subjectId, String subjectName)
+createSubject(String id, String name)
+updateSubject(String id, String name)
+deleteSubject(String id)
+assignStudentSubject(Student studentN, String subjectID)
+assignTeacherSubject(Teacher teacherN, String subjectID)
+removeStudentSubject(Student student, Subject subject)
+deleteTeacherSubject(Teacher teacher, Subject subject)
+loadPublishedExamsByStudent(Student student)
+checkExamName(String examName)
+checkExamSignificantInfo(String year, String semester, String examType)

3. Exam Datamapper

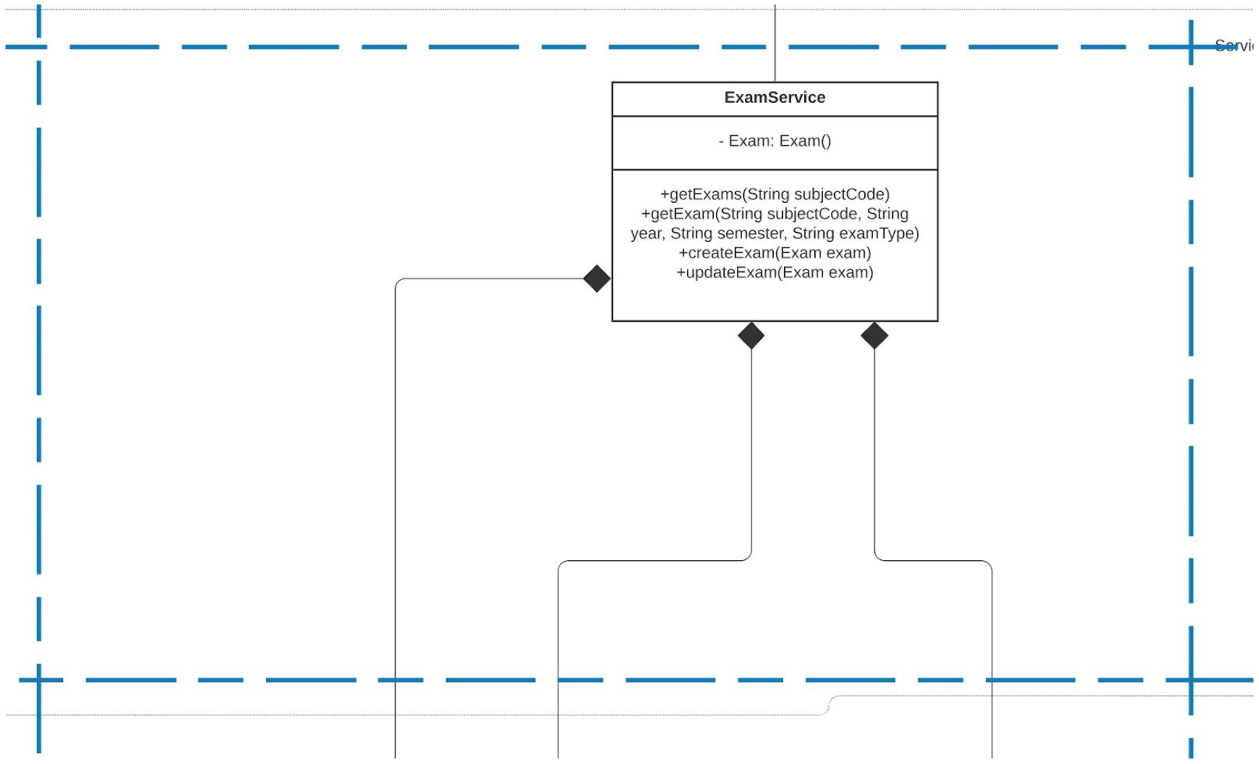


Service level

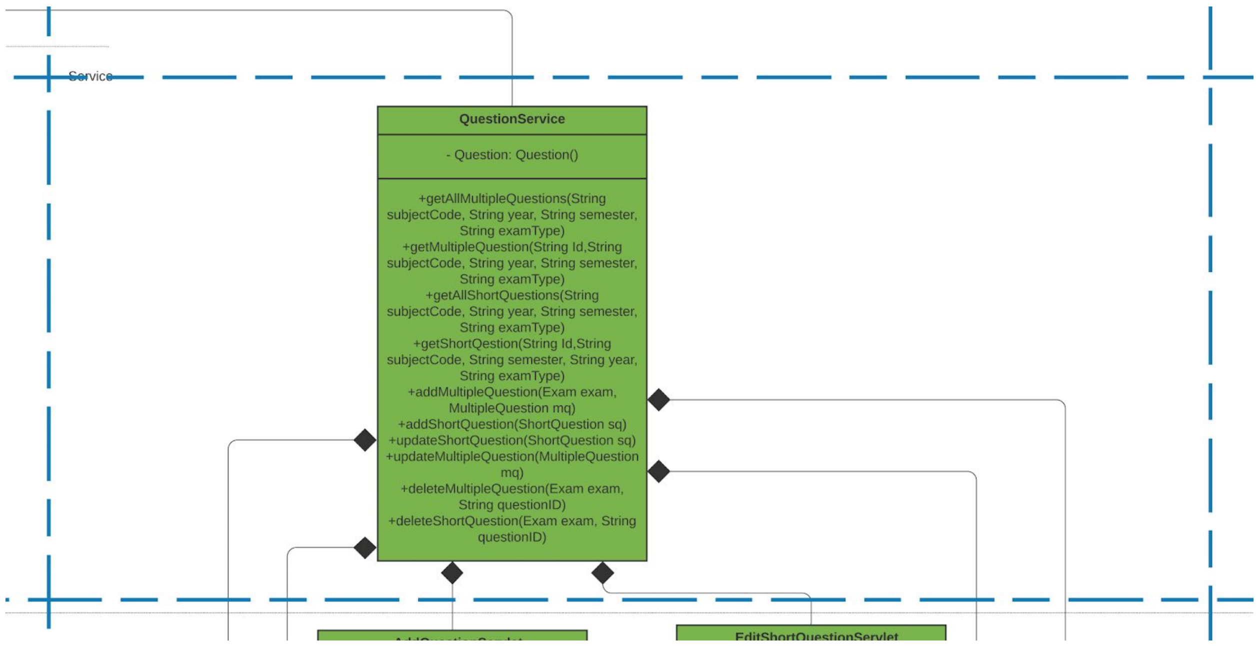
1. Admin Service


AdminService
-admin: Admin()
+getSubject(String subjectId)
+hasStudentsBySubject(String subjectId)
+hasTeachersBySubject(String subjectId)
+hasExamsBySubject(String subjectId)
+getStudentsBySubject(String subjectId)
+getTeachersBySubject(String subjectId)
+getExamsBySubject(String subjectId)
+getAllStudents()
+getAllTeachers()
+createSubject(String subjectId, String subjectName, String[] studentsList, String[] teachersList, String year, String semester)

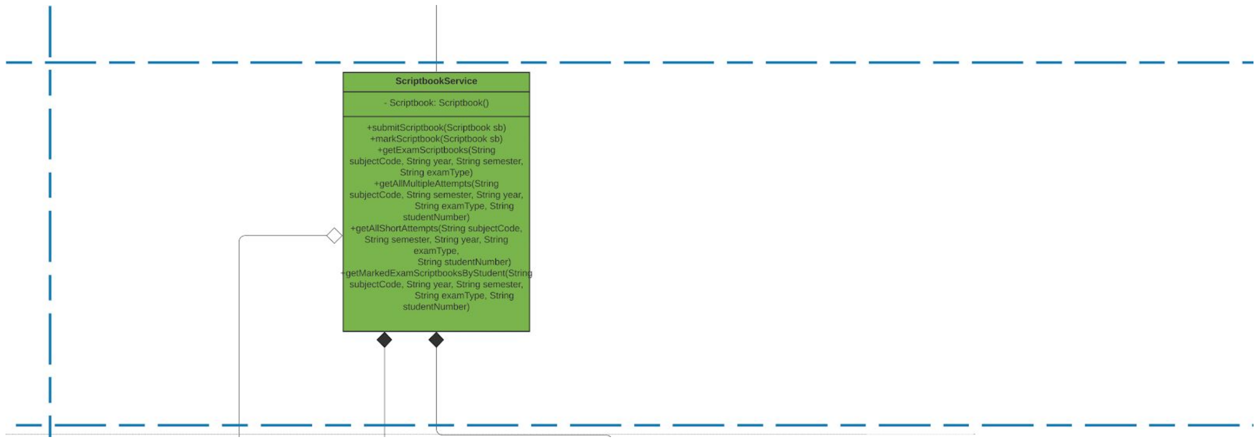
2. Exam Service


ExamService
- Exam: Exam()
+getExams(String subjectCode)
+getExam(String subjectCode, String year, String semester, String examType)
+createExam(Exam exam)
+updateExam(Exam exam)

3. Question Service

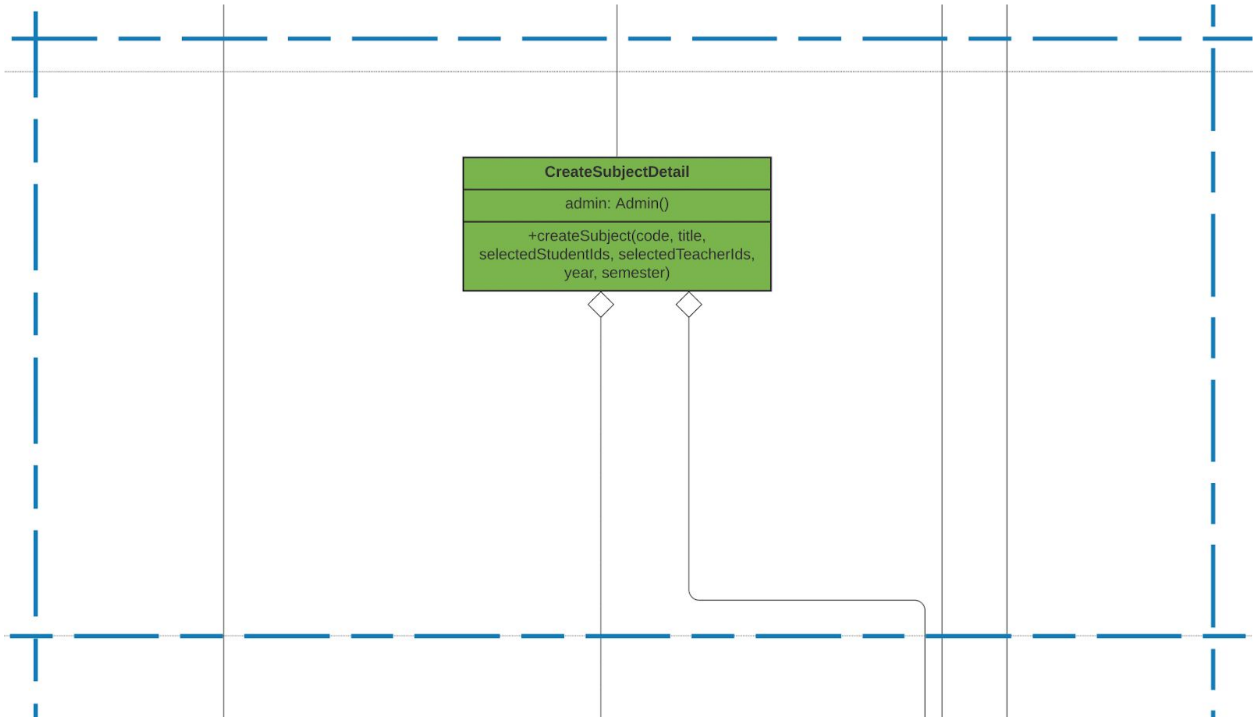

Service
QuestionService
- Question: Question()
+getAllMultipleQuestions(String subjectCode, String year, String semester, String examType)
+getMultipleQuestion(String Id,String subjectCode, String year, String semester, String examType)
+getAllShortQuestions(String subjectCode, String year, String semester, String examType)
+getShortQestion(String Id,String subjectCode, String semester, String year, String examType)
+addMultipleQuestion(Exam exam, MultipleQuestion mq)
+addShortQuestion(ShortQuestion sq)
+updateShortQuestion(ShortQuestion sq)
+updateMultipleQuestion(MultipleQuestion mq)
+deleteMultipleQuestion(Exam exam, String questionID)
+deleteShortQuestion(Exam exam, String questionID)
EditShortQuestionServlet

4. Scriptbook Service

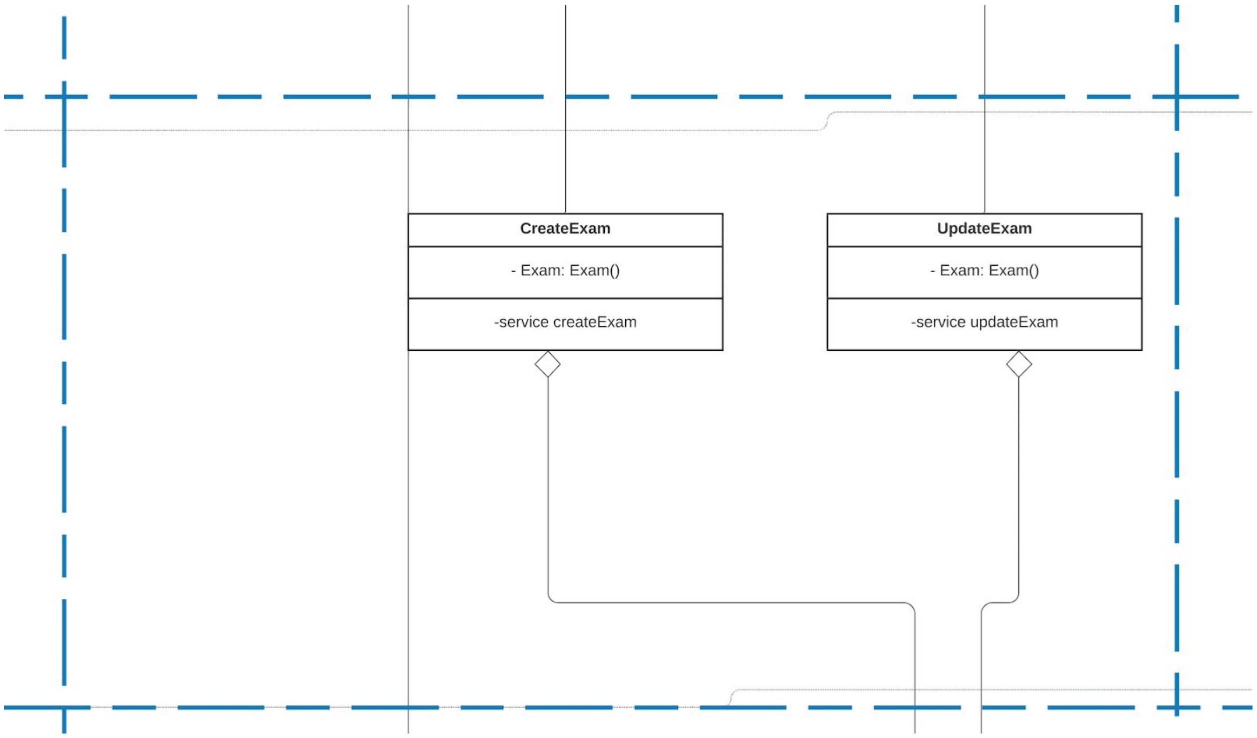

ScriptbookService
- Scriptbook: Scriptbook()
+submitScriptbook(Scriptbook sb)
+markScriptbook(Scriptbook sb)
+getExamScriptbooks(String subjectCode, String year, String semester, String examType)
+getAllMultipleAttempts(String subjectCode, String semester, String year, String examType, String studentNumber)
+getAllShortAttempts(String subjectCode, String semester, String year, String examType, String studentNumber)
+getMarkedExamScriptbooksByStudent(String subjectCode, String year, String semester, String examType, String studentNumber)

Presentation level

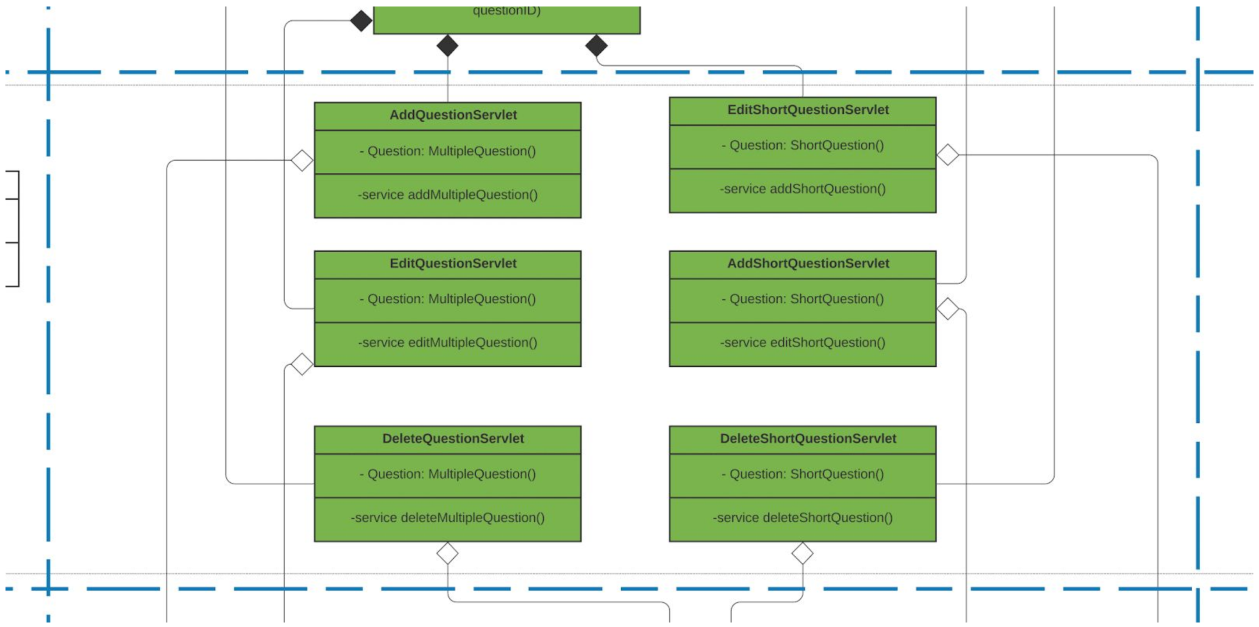
1. User Presentation


CreateSubjectDetail
admin: Admin()
+createSubject(code, title, selectedStudentIds, selectedTeacherIds, year, semester)

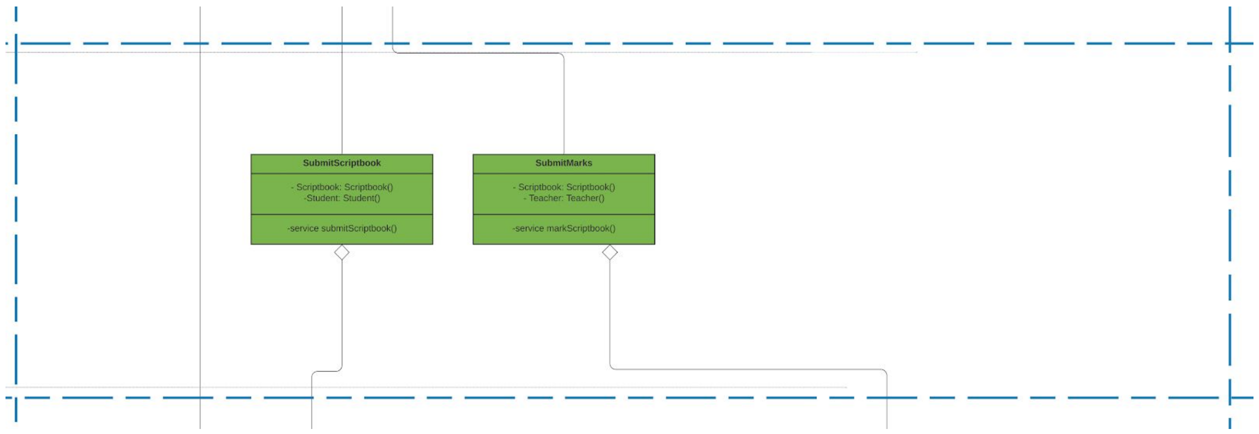
2. Exam Presentation



3. Question Presentation

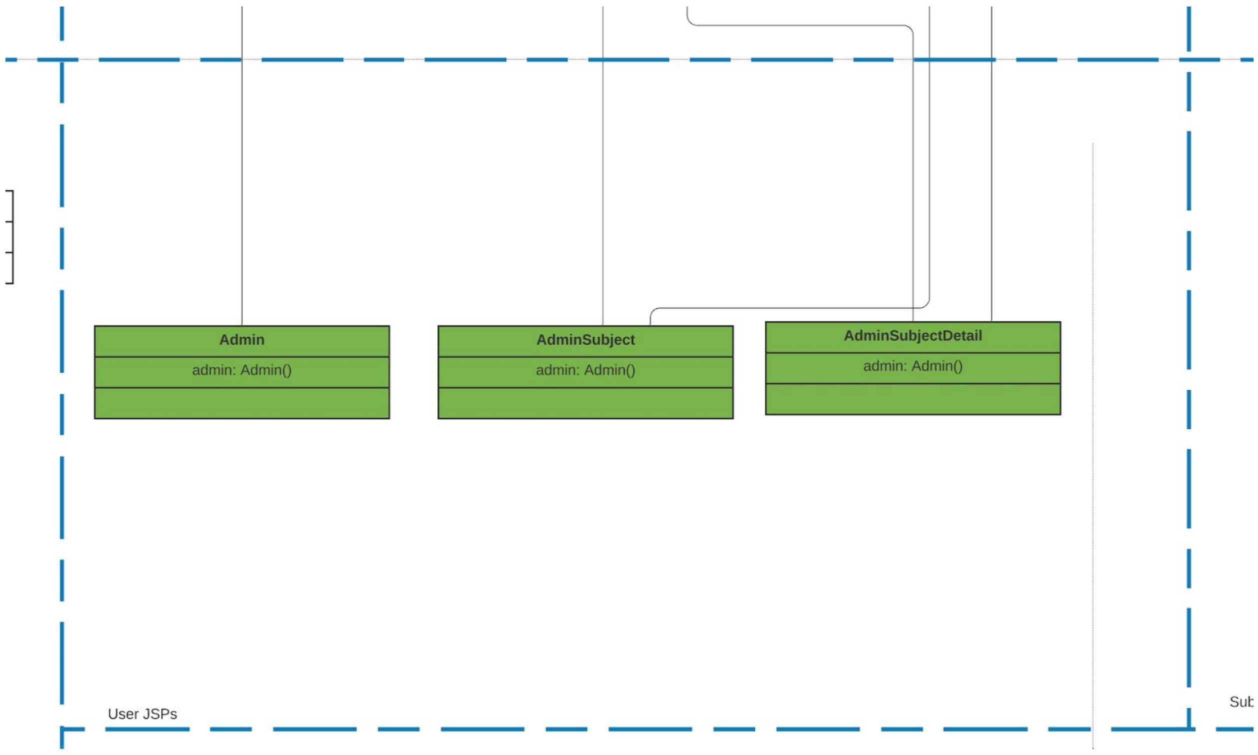

questionID)
AddQuestionServlet
- Question: MultipleQuestion()
-service addMultipleQuestion()
EditShortQuestionServlet
- Question: ShortQuestion()
-service addShortQuestion()
EditQuestionServlet
- Question: MultipleQuestion()
-service editMultipleQuestion()
AddShortQuestionServlet
- Question: ShortQuestion()
-service editShortQuestion()
DeleteQuestionServlet
- Question: MultipleQuestion()
-service deleteMultipleQuestion()
DeleteShortQuestionServlet
- Question: ShortQuestion()
-service deleteShortQuestion()

4. Scriptbook Presentation

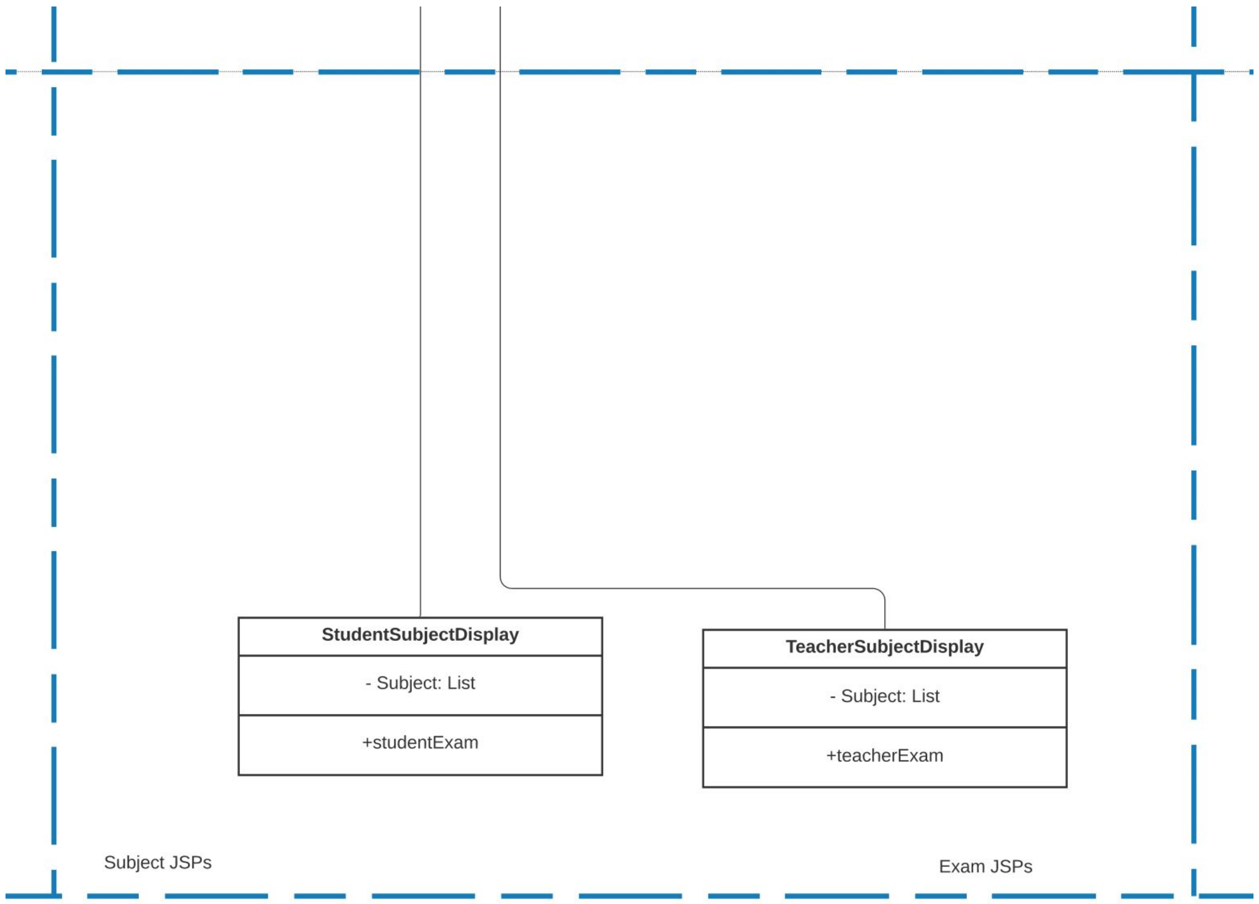

SubmitScriptbook
- Scriptbook: Scriptbook()
-Student: Student()
-service submitScriptbook()
SubmitMarks
- Scriptbook: Scriptbook()
- Teacher: Teacher()
-service markScriptbook()

JSP level

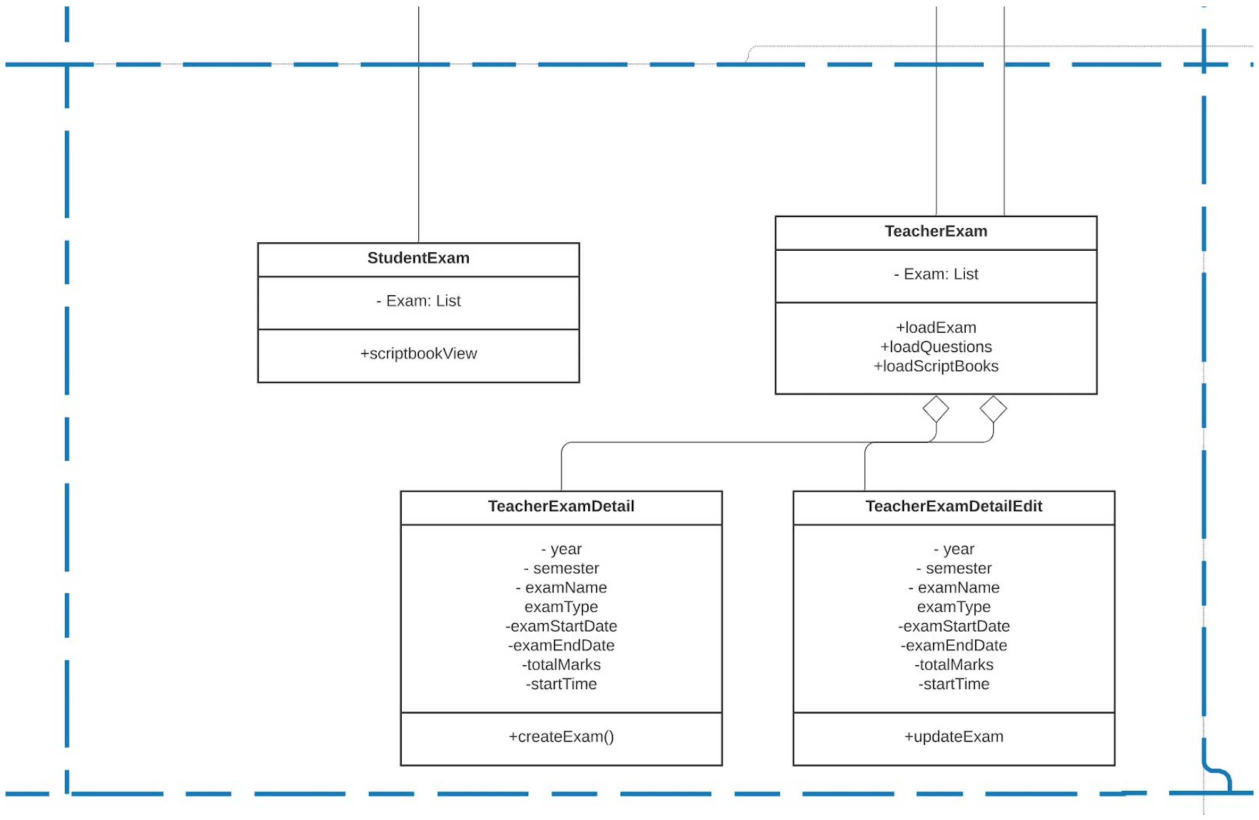
1. User JSP



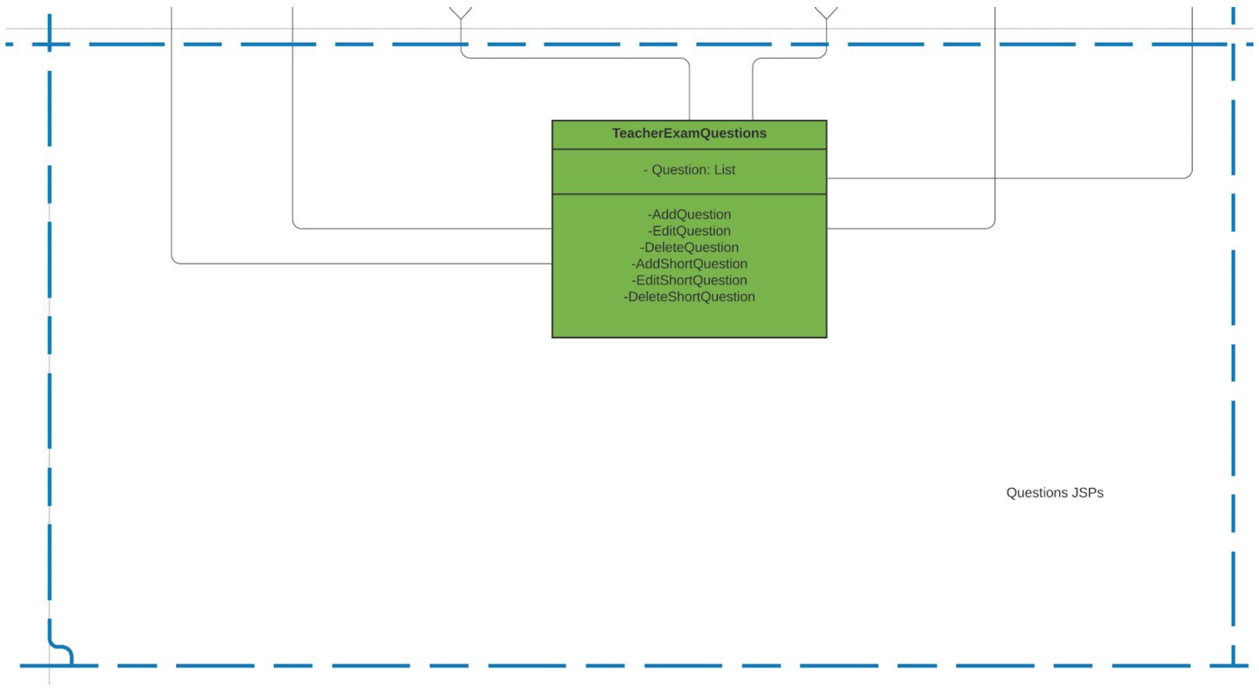
2. Subject JSP


StudentSubjectDisplay
- Subject: List
+studentExam
TeacherSubjectDisplay
- Subject: List
+teacherExam
Subject JSPs
Exam JSPs

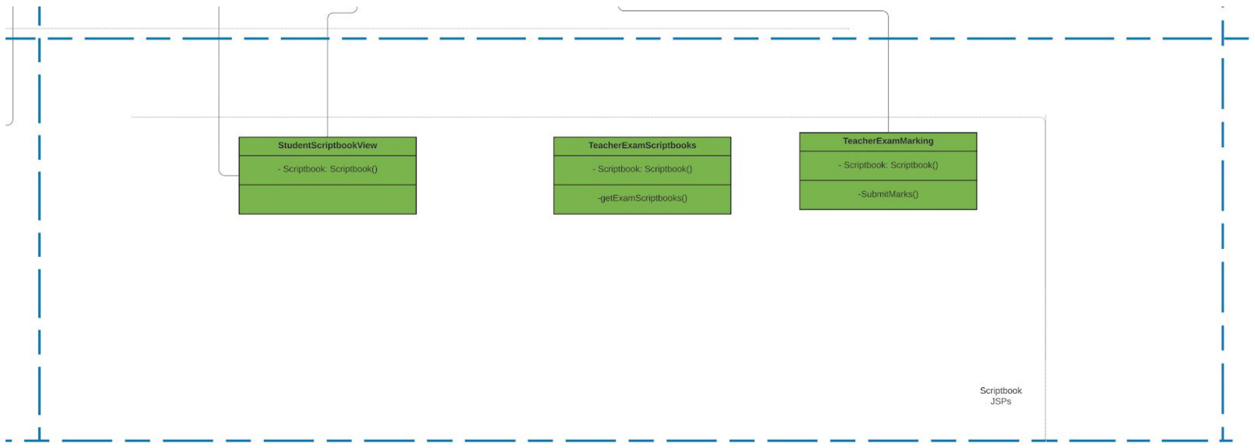
3. Exam JSP



4. Question JSP



5. Scriptbook JSP


StudentScriptbookView
- Scriptbook: Scriptbook()
TeacherExamScriptbooks
- Scriptbook: Scriptbook()
-getExamScriptbooks()
TeacherExamMarking
- Scriptbook: Scriptbook()
-SubmitMarks()
Scriptbook
JSPs