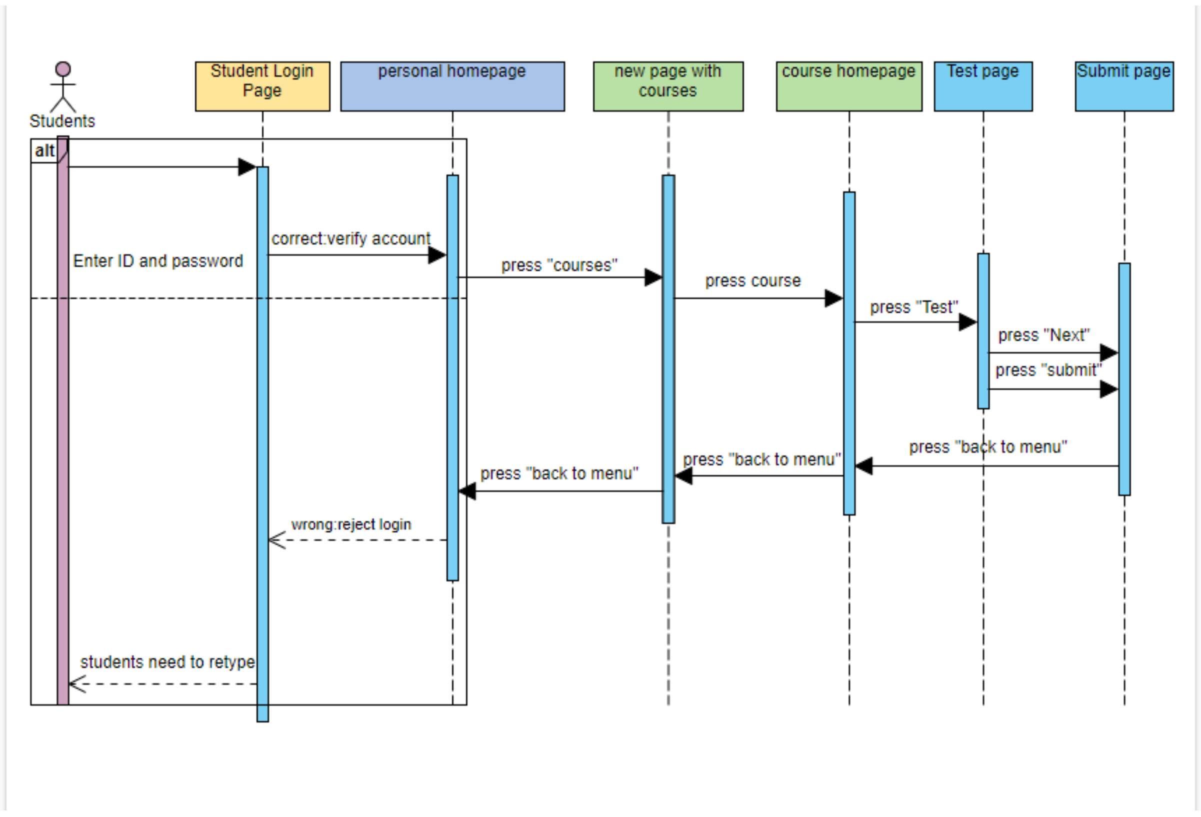
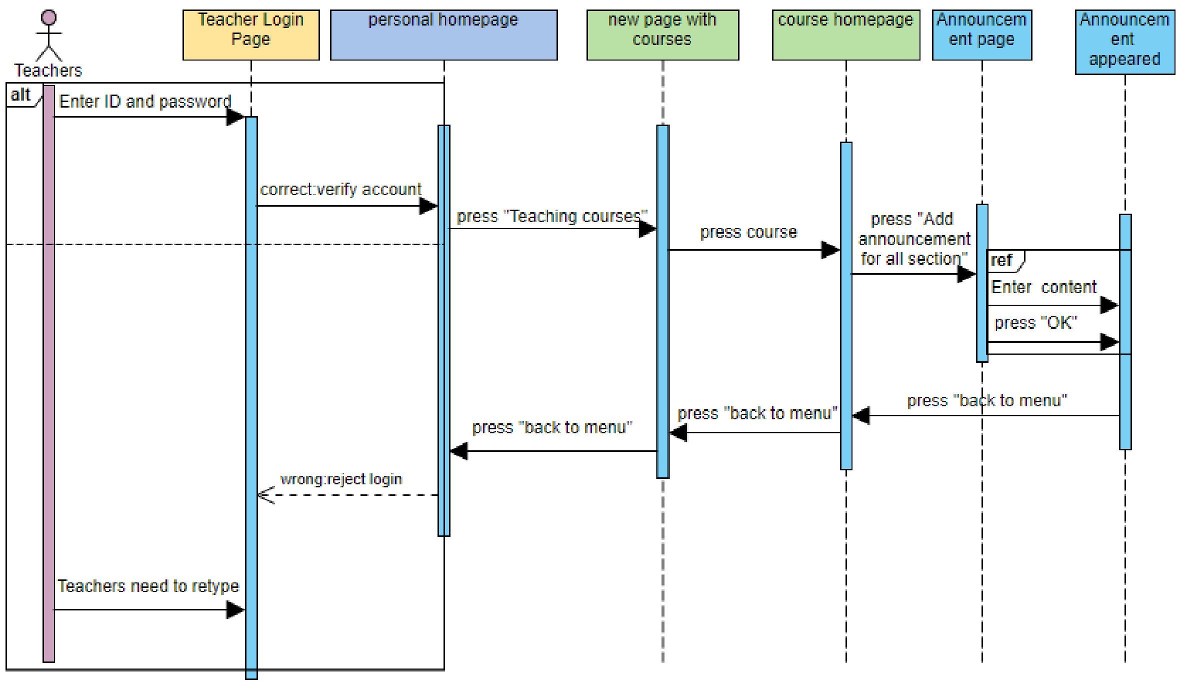
Interim Release

Sequence Diagrams:

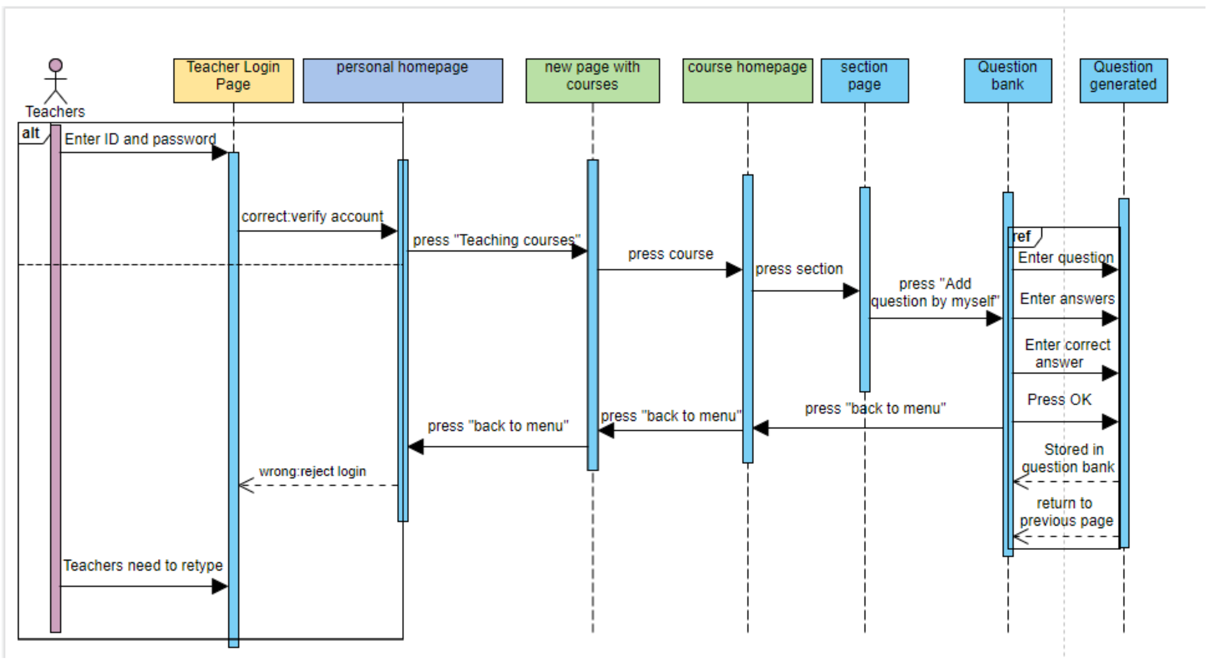
Use case1: Take A Test



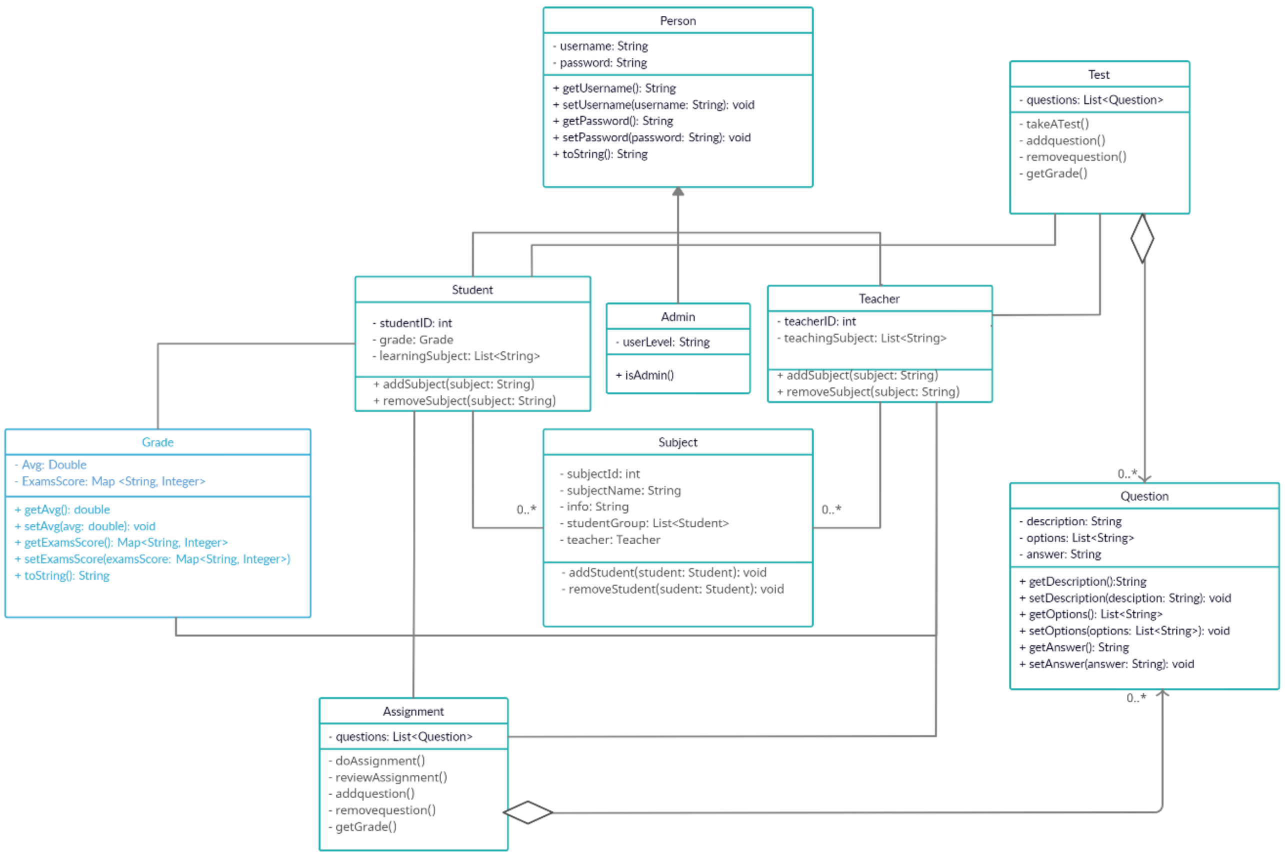
Use case2: Release announcement



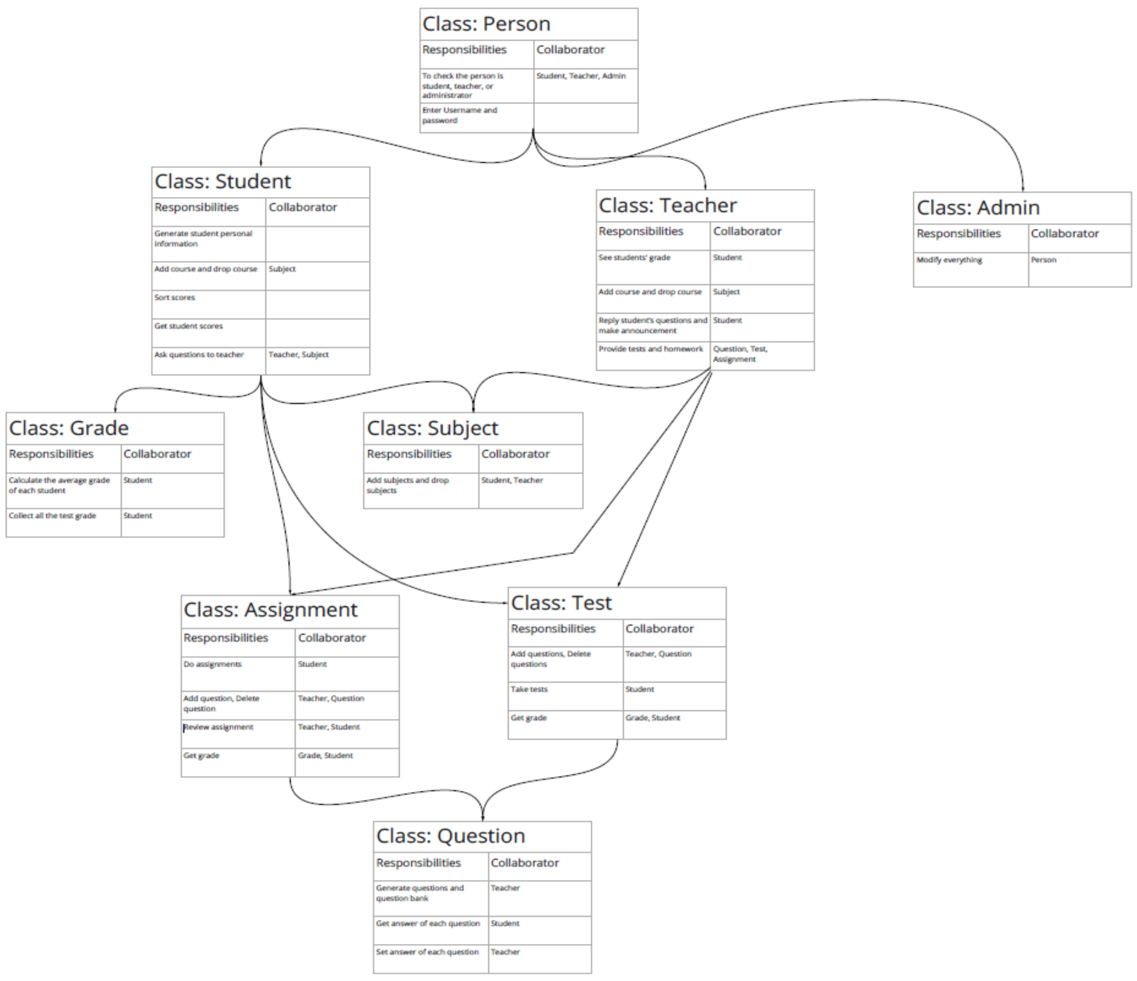
Use case: Add Question



Static Class Diagrams:


Person
- username: String
- password: String
+ getUsername(): String
+ setUsername(username: String): void
+ getPassword(): String
+ setPassword(password: String): void
+ toString(): String
Test
- questions: List<Question>
- takeATest()
- addquestion()
- removequestion()
- getGrade()
Student
- studentID: int
- grade: Grade
- learningSubject: List<String>
+ addSubject(subject: String)
+ removeSubject(subject: String)
Admin
- userLevel: String
+ isAdmin()
Teacher
- teacherID: int
- teachingSubject: List<String>
+ addSubject(subject: String)
+ removeSubject(subject: String)
Grade
- Avg: Double
- ExamsScore: Map <String, Integer>
+ getAvg(): double
+ setAvg(avg: double): void
+ getExamsScore(): Map<String, Integer>
+ setExamsScore(examsScore: Map<String, Integer>)
+ toString(): String
Subject
- subjectId: int
- subjectName: String
- info: String
- studentGroup: List<Student>
- teacher: Teacher
- addStudent(student: Student): void
- removeStudent(sudent: Student): void
0..*
0..*
0..*
Question
- description: String
- options: List<String>
- answer: String
+ getDescription():String
+ setDescription(desciption: String): void
+ getOptions(): List<String>
+ setOptions(options: List<String>): void
+ getAnswer(): String
+ setAnswer(answer: String): void
0..*
Assignment
- questions: List<Question>
- doAssignment()
- reviewAssignment()
- addquestion()
- removequestion()
- getGrade()

CRC Cards:



Class: Person	
Responsibilities	Collaborator
To check the person is student, teacher, or administrator	Student, Teacher, Admin
Enter Username and password	

Class: Student	
Responsibilities	Collaborator
Generate student personal information	
Add course and drop course	Subject
Sort scores	
Get student scores	
Ask questions to teacher	Teacher, Subject

Class: Teacher	
Responsibilities	Collaborator
See students' grade	Student
Add course and drop course	Subject
Reply student's questions and make announcement	Student
Provide tests and homework	Question, Test, Assignment

Class: Admin	
Responsibilities	Collaborator
Modify everything	Person

Class: Grade	
Responsibilities	Collaborator
Calculate the average grade of each student	Student
Collect all the test grade	Student

Class: Subject	
Responsibilities	Collaborator
Add subjects and drop subjects	Student, Teacher

Class: Assignment	
Responsibilities	Collaborator
Do assignments	Student
Add question, Delete question	Teacher, Question
Review assignment	Teacher, Student
Get grade	Grade, Student

Class: Test	
Responsibilities	Collaborator
Add questions, Delete questions	Teacher, Question
Take tests	Student
Get grade	Grade, Student

Class: Question	
Responsibilities	Collaborator
Generate questions and question bank	Teacher
Get answer of each question	Student
Set answer of each question	Teacher

Class: **Grade**	
Responsibilities	**Collaborators**
Calculate the average grade of each student	Student
Collect all the test grade	Student

Class: **Person**	
Responsibilities	**Collaborators**
To check the person is student, teacher or, administrator	Student, Teacher, Admin
Enter username and password	

Class: **Question**	
Responsibilities	**Collaborators**
Generate questions and question bank	Teacher
Get answer of each question	Student
Set answer of each question	Teacher

Class: **Student**

Responsibilities	**Collaborators**
Generate student personal information	
Add course and drop course	Subject
Sort scores	
Get student scores	
Ask questions to teacher	Teacher, Subject

Class: **Subject**	
Responsibilities	**Collaborators**
Add subjects	Teacher, Student
Drop subjects	Teacher, Student

Class: **Teacher**	
Responsibilities	**Collaborators**
Add subjects and drop subjects	Subject
See students' grade	Student
Reply student's questions and make announcement	Student
Provide tests and homework	Question, Test, Assignment

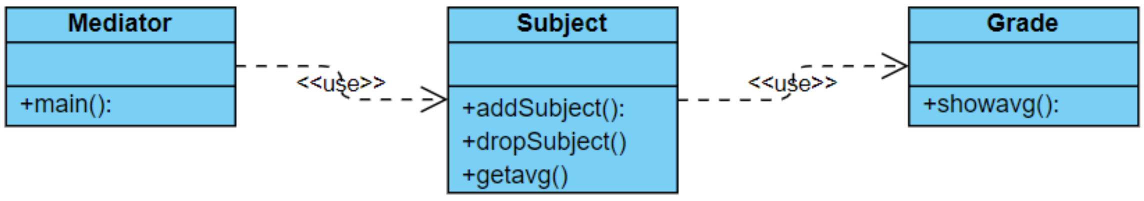
Class: **Test**	
Responsibilities	**Collaborators**
Add questions	Teacher
Take tests	Student
Delete questions	Teacher
Get grade	Grade, Student

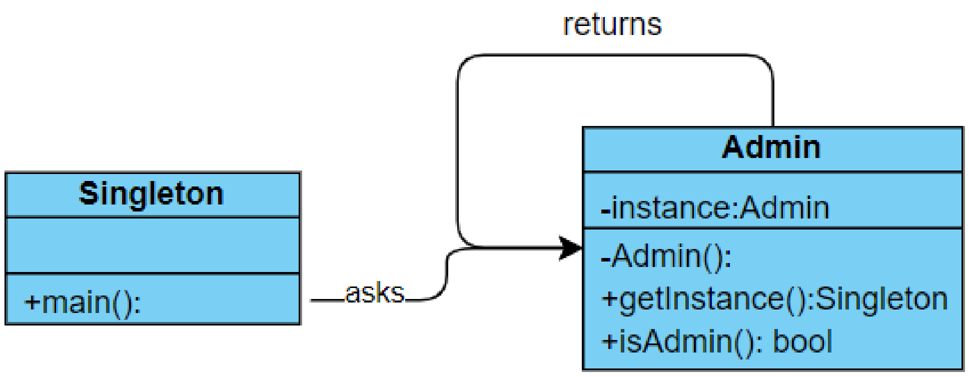
Class: **Assignment**	
Responsibilities	**Collaborators**
Do assignments	Student
Add question, Delete question	Teacher, Question
Review assignment	Student, Teacher
Get grade	Grade, Student

Class: **Admin**	
Responsibilities	**Collaborators**
Modify everything	Person

Design Approach:

For the class diagram shown above, we have “Grade”, “Person”, “Question”, “Student”, “Subject”, “Teacher”, and “Admin” classes. “Person” class is our superclass, which has two variables “username” and “password”. Specifically, when users would like to log in to our system, they need these two variables. Then “Student” and “Teacher” classes are the subclass of “Person” class. They inheritance these two variables from their parent class “Person”. They also have their variables since we will not break the Single Responsibility Principle here. That is to say, a class should have only one reason to change. In the “Student” class, we have “studentID” to record student users’ ID number, “grade”, which is a “Grade” object, to record all their grades, and “learningSubject”, which is a list of String, to store the subjects they learn for this semester. In the “Teacher” class, we have “teacherID” and “teachingSubject”, which are similar to those of the “Student” class but with different properties. In this way, we follow the Open-Closed Principle, that “Person” class is open for extension but closed for modification. We follow the Liskov Substitution Principle as well, which requires that subtypes must be substitutable for their base types. We define both “Student” class and “Teacher” class as the true subtypes, which are replaceable for the “Person” class. And we design this “Grade” class used to store all the grades of a specific “Student” object, which includes the quiz grades, the exam grades, as well as the average grade. For convenience, we just use Map to keep all quiz and exam scores together. Then we can use a specific quiz name or exam name to get the score value. For the “Subject” class, we have “subjectID” for a specific course number, “subjectName” for the name of a subject, “info” to store the curriculum information like prerequisite for a specific course, “studentGroup”, which is a list to store “Student” object, “teacher” for “Teacher” object, and “questionList”, which is a list to store “Question” object. By designing “Subject” class in this way, we can use Mediator design pattern to reduce communication complexity between “Subject” class and “Grade” class, which provides easy maintenance of the code by loose coupling. We can therefore get the average score of a specific subject by calling “getavg()” method. For the “Admin” class, since we can ensure that this class only has a single instance, we could use the Singleton design pattern here. With Singleton, global access to this instance is provided, which is convenient for us to test out code. And the Singleton object is initialized only when it is required for the first time.




returns
Singleton
+main():
asks
Admin
-instance:Admin
-Admin():
+getInstance():Singleton
+isAdmin(): bool

Mock up Test Plan:

Tasks:

Tasks	**Number of tests**	**Duration**	**Start date**	**End date**
Test creating a duplicate account	10	30 minutes	7.26	7.26
Test creating a new account with empty username	10	30 minutes	7.26	7.26
Test creating a new account with invalid password	10	30 minutes	7.26	7.26
Test logging in with the correct account password	5	15 minutes	7.26	7.26
Test logging in with the incorrect account password	5	15 minutes	7.26	7.26
Test logging in simultaneously with 100 users	5	30 minutes	7.27	7.27
Test logging in simultaneously with 200 users	5	1 hour	7.27	7.27
Test the respond speed of logging in	20	30 minutes	7.27	7.27
Test calculating the due date	5	30 minutes	7.28	7.28
Test calculating the remaining questions	5	30 minutes	7.28	7.28
Test showing the basic information	3	10 minutes	7.28	7.28
Test taking the quiz	10	1 hour	7.28	7.28
Test taking the exam	10	2 hours	7.28	7.28
Test taking the exam simultaneously with 100 users	5	4 hours	7.29	7.29
Test the respond	20	3.5 hours	7.29	7.29

speed of showing the grade after a quiz or an exam				
Test calculating class average	10	30 minutes	7.29	7.29
Test matching color with grade	10	30 minutes	7.29	7.29
Test adding tags	5	1 hour	7.30	7.30
Test filtering questions based on subject	10	1 hour	7.30	7.30
Test filtering questions based on difficulty	10	1 hour	7.30	7.30
Test filtering questions based on subject	10	1 hour	7.30	7.30
Test releasing announcements	5	30 minutes	7.30	7.30
Test UI with different modes	5	15 minutes	7.30	7.30

Questions about the interface:

- Do you meet any lagging issues with our interface?
- What button do you think is the most unnecessary for its design?
- Could you describe the worst experience when you use our UI?
- Could you give some suggestion on how we can improve our UI?
- Do all buttons work as you desire?
- What colors and modes would you prefer in our UI?
- Do you think our interface can provide great convenience for you?
- What is the part of our UI interface that you satisfy the most?

Goals:

Testers will write several test suits to cover all tasks mentioned above with Java JUnit Test. All the tests should pass the Java JUnit Test. After Junit Test, testers should first check with the return status, and check whether there exists a mistake. Then testers have to check the updates of the database. To check whether some quiz scores or some exam scores were added to the database is necessary. And testers will check whether the data transmissions between Java and MySQL are successful. Also, testers will check the grade diagram generated according to the database.

Others:

Testers should test all the tasks listed above step by step and prepare for the test inputs. Testers should try some corner cases, which are uncommon but possible. If one of the tests fails, testers should first check with the test suits, and make sure there is no bug there. If testers find any bugs in the test suits, testers should modify them first. Otherwise, the error occurs in our project. Then we should find it and modify it immediately.

Contribution Summary:

Rongzhe Cui	Participate in weekly discussion. Prepare for the demo presentation. Write contribution summary Write status report
Honghao Huang	Participate in weekly discussion. Write Design Approach Write mockup test plan
Han Ling	Participate in weekly discussion. Write CRC cards Draw sequence diagrams
Lifu Zhang	Participate in weekly discussion. Prepare for the demo presentation.
Yiduo Jiang	Participate in weekly discussion. Draw static class diagrams

Status Report:

Sprint 3 deliverable

We have finished discussing and writing interim deliverables last week. We are still preparing for the first demo next weekly meeting.

Previous Weeks' development

We implemented functions in order to fulfill features that we promised for interim in sprint 3 schedule

We worked on student's and teacher's part and made debugs

Blockers Encountered
These two weeks we have plenty of assignments to be completed. We have to manage time properly
Code issue: we met several bugs of the functions for teacher users like students cannot see the released tests and assignments

Next week's assignment:
We are going to develop UI next week.
We will try to perform an excellent presentation (first demo) next meeting