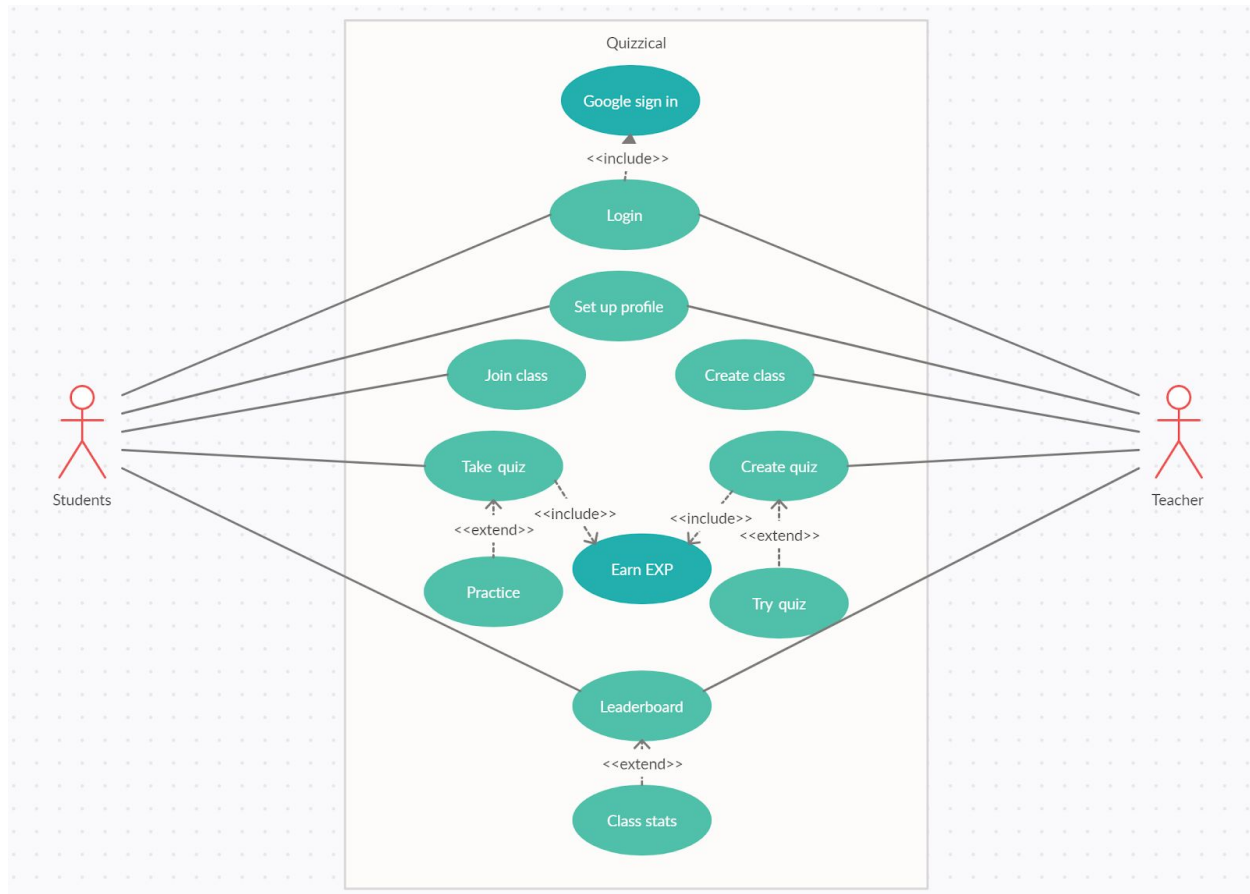


# 1. Requirements

## 1.1 Uses Cases

- Users should be able to use Google sign in to create an account. When the user logs in again with the same Google account, the user info should be retrieved from the server. (**Google sign in API**)
- Users can change their profile picture, username, user email and push notification rate
- Teachers can create multiple class (specifying class name, class grade and course categories), multiple quiz modules in each class and quizzes
- When teacher creates a class, he/she will receive an email notification containing the class code to distribute using **Gmail API**
- When a teacher updates quiz modules in a class or delete a class, all his/her students receive a push notification (**firebase push notification**)
- Teachers can earn EXP through creating quizzes and receiving likes from students (**non trivial logic**)
- Students can join multiple classes and do quizzes to earn EXP. The EXP earned will be based on the correctness in the quiz. (**non trivial logic**)
- Students can review which question they did wrong in each quiz, and see the class average and highest score
- Teachers can see the overall class statistics for each quiz, as well as the overall performance of a single student
- Leaderboard for both students and teachers should be updated based on EXP
- Users can choose to receive push notification about leaderboard update every day/week/month (node scheduler)

## 1.2 Use Case Diagram



## 1.3 Non-functional Requirements

- The users will be informed with alert dialog when clicking on each button so that they can have a chance to decide whether they want to proceed or not to avoid unintentional behavior. We identify it as a user experience related non-functional requirement, because all the buttons already inform the user what function the button has. With this in mind, we don't actually need to inform the user of every button click. This is just a safeguard for the user in case they accidentally click on some button and cause unintentional behaviors.
- The teacher should be able to use only 1 picture (load/take it one time) with a cropping function to set up one question with all its choices so that the quiz setup process can be easier and faster. We identify it as a user experience related non-functional requirement. The user can click fewer times and don't need to take many pictures to finish creating a question. Also, this is not mandatory, we could have the user take multiple pictures for one question, but this implementation can greatly reduce the complexity and time used to create a quiz.
- Users identities should not be revealed (security, using UID as main identifier instead of any personal info)

## 1.4 Necessary user accounts/Details needed for Teaching Staff to test app

There are not too many restrictions on what accounts or details that are needed to test the app. One teaching staff can create an account as an instructor (when logging in first time, tick the instructor checkbox to be an instructor), and create a class and quizzes. Others can create accounts as students and join the corresponding class to take the quizzes. If you want to use a class that is already created, you can enter class code 35607 as a student and take the simple quizzes we have made in the class.

Note: if you want to build the app in your own environment (e.g. when doing frontend UI test which you might need to log in first), you need to replace the client id with your own google client id at line 108 in **CPEN321-Quizzical\Front-end\app\src\main\res\values\strings.xml**

## 2. Design

### List of all backend components

Class: for getting class information such as modules in a class. Also handles deleting a class

emailSending: contains helper functions for sending emails.

firebasePush: for sending Firebase push notifications, with scheduling.

Index: for getting home page. (not related to app usage)

instructor Leaderboard: for getting information to display the Instructor Leaderboard.

Quiz: for getting information related to quizzes, such as quiz questions, quiz score (average, highest, user scores), wrong question lists etc.

studentLeaderboard: for getting information to display the student Leaderboard it will return both score and EXP.

Upload: for uploading all data.

Users: for getting user information such as username, email, firebase push notification frequency, user's class list, etc.

## 3. Testing

### Automated frontend tests

Note: if you want to run the tests on your own machine, you may need to replace the client id with your own google client id at line 108 in

**CPEN321-Quizzical\Front-end\app\src\main\res\values\strings.xml** and sign in the app properly with either student or instructor account, details in the comments in each test files.

In

**CPEN321-Quizzical\Front-end\app\src\androidTest\java\com\cpen321\quizzical\CreateQuizTest.java** - 9 tests (2 are back button test listed in the non-functional requirement section):

1. testChangingModules: We change the module name in the CreateQuizActivity. The test will pass if the toast message shown after we changed the module corresponds to the quiz module we have chosen. And fails if the toast message is not correct.
2. testSubmitNoQuestions: We click Finish directly after we get into the CreateQuizActivity. The test will pass if a warning of no question input is shown, and fails otherwise. If we click on OK, we should remain in createQuizActivity. And the test will fail if we go to any other activities.
3. testTooManyQuestions: we have a limit of 20 questions. If the user adds more than 20 questions, there will be a warning message shown on the screen indicating that the number of questions has exceeded the limit. The test will pass if the message is shown and there is never a 21st question in the question list.
4. testTooManyChoices: similar to the test above. We have a limit of 6 choices per question. If the user adds more than 6 choices to the question, there will be a warning

message. The test tries to add more than 6 choices and check the warning message is correctly shown.

5. testNoQuestionInput: we need to make sure that every question submitted has a question field (either text or image, the image test will be shown manually). If a user submits a quiz with some questions having no question field. A warning will be shown and the quiz will not be submitted. The test will pass if we correctly shown the warning and remain in the CreateQuizActivity(quiz not submitted)
6. testTooFewChoices: similar to the test 5 above, but we need to make sure that the user cannot submit quiz with some question having less than 2 choices
7. testNoCorrectAnswer: similar to the test 5 above, but we need to make sure that the user cannot submit a quiz with some question having no correct answer.

**Note: InitActivityTest.java suspended, because most of the test needs the backend disabled to work.**

In

**CPEN321-Quizzical/Front-end/app/src/androidTest/java/com/cpen321/quizzical/Instructor CreateClassTest.java - 6 tests:**

1. checkClassNameInvalid: similar to check name invalid, we put an invalid class name (a single character "a") to the class name field when creating a class. The test passes if when we click on the "submit" button in create class, a "class invalid" message is shown.
2. addNewClassTest: We create a class with class name "CLASSINTEST", this is a valid class name. The new class should be added to the user's class list. And the test will check that a new class created message is shown in the dialog. We call deleteClass after we have added the class. The test fails if deleteClass cannot find the class in our class list. (The class is not created, thus not deletable)
3. addDuplicateClassTest: We create two classes with the same class name in a row, without deleting the existing one. The test passes if "class name invalid" message shows up when creating the second one and the second one is not added to the class list. If the second class is created successfully, the test fails.
4. addQuizNoModuleTest: we firstly create a class, then try to create a quiz directly without setting up a quiz module. The test fails if the app goes into createQuizActivity directly, and passes if a create module dialog pops up.
5. addModuleTest: we create a class and add a module to the class. The test passes if there is a new module set up in the UI, with all (quiz, notes, wrong questions, stats) field giving no data.
6. addQuizAfterCreatingModule: this is similar to test4, but this time, since we have added a quiz module already, the test will fail if it again asks us to create a module, and the the test will pass if it goes into createQuizActivity directly.

In

**CPEN321-Quizzical/Front-end/app/src/androidTest/java/com/cpen321/quizzical/Instructor StatisticTest.java - 2 tests**

The tests are very simple, we just check if the UI is correctly set up when we click the button that switches between student's stats and teacher's leaderboard. If the switch is correct (in the teacher's leaderboard, there should be no score value, and the button should change text to "Back to statistics". In the student's statistics board, there should be a score field, representing the students' scores, the button should have text: "Teachers leaderboard"), then the tests pass.

In

**CPEN321-Quizzical/Front-end/app/src/androidTest/java/com/cpen321/quizzical/ProfileFragmentTest.java** - 7 tests:

1. `testValidUsernameChange`: we used "Valhalla" as the valid username. The test will click on the change username button and enter the username and press submit. After that the test will check if the username field in the Profile fragment changed to the new valid username. If it changes, then the test passes.
2. `testInvalidUsernameChange`, similar to above, but we are using an invalid string "va", which is less than 3 characters. The expected output is: username invalid message shown on the screen and username will not be changed. The test will pass if the expected output is shown.
3. `testValidEmailChange()`: similar to the change email tests described above in the `InitActivityTest`. Also, we will check that the email field in the profile is set to the new valid email. The test will pass if the profile email field is correctly updated.
4. Similar to above, but with an invalid email, and the profile email field should not be changed and there should be an email invalid message showing up. The test will pass if both criteria are met.
5. `testLogoutInfo`: this is a non-functional requirement test and will be explained in that part.
6. `testChangeProfileImage`: The test checks whether an alert dialog asking whether the user wants to change the image actually pops up. If the user chooses "no", the app should not modify the user profile image. The test passes if nothing is changed. If the user chooses "yes", the app will check if the permission for reading storage has been given, and prompt the user to choose an image from the phone storage. If the image is the same as before, nothing should be done by the app, and The automated part only deals with "no" selected. We will test the "yes" part manually, because we have trouble setting Espresso to choose images from storage.
7. `testChangeNotificationFrequency`: in this test, we iterate through the notification frequency list, and choose the corresponding values in the push notification spinner. The test passes if after every change, the value of notification frequency in the user's shared preference exactly matches the choice. If there is one mismatch, the test fails.

In

**CPEN321-Quizzical/Front-end/app/src/androidTest/java/com/cpen321/quizzical/InstructorQuizFragment test** - 2 tests:

In this test, similar to the test above, but we are testing with a class with real data. We check whether the class stat and wrong question list are showing the data in the correct format. If the

data are in the correct format as shown in the picture, then the test passes, otherwise, the test fails.

In

**CPEN321-Quizzical/Front-end/app/src/androidTest/java/com/cpen321/quizzical/QuizActivityTest.java** - 5 tests (2 are back button test listed in the non-functional requirement section)

(Note in the tests here, since the rendering speed of Latex (3rd party code) in simulator is slow, the choice texts are not shown in the tests):

1. **testNoResponseSelected:** this test checks that we have handled the selection reading correctly. The test will not make any choices for the question and press submit directly. The test will pass with a warning "Please answer the question before submission." shown in the UI. The test will fail if we can go to the next question directly.
2. **testResponseAllCorrect:** This test checks the flow of UI change during quiz activity. The test will choose the correct answer based on the ID set to the choices UI. And it will detect the button text changes through the end. After the quiz is finished, it will also check that the **quizFinishedActivity** is correctly set up with the student results. The test will pass if every button press leads to the correct button text change and correct number text change, and if after the final question, we correctly go to the **QuizFinishedActivity**. The test will fail if there is any text mismatch or logic mismatch.
3. **testResponseOneWrong:** Similar to the second test, but have one question answered wrongly. We check that the correct number count text for the wrong question and the following questions as well as the text in the **quizFinishedActivity** are set up correctly. The test will fail if one of the texts is not correct.

In

**CPEN321-Quizzical/Front-end/app/src/androidTest/java/com/cpen321/quizzical/StudentClassTests.java** - 5 tests

1. **joinClassTest:** Make sure that given a correct class code, we can successfully join the class when we enter it. The test will pass if we get a toast message that we succeeded.
2. **joinClassAlreadyTest:** Make sure if we try to join a class we joined already and enter the same code it will give us a toast message that we cannot join. The test will pass when it gets the toast.
3. **invalidClassTest:** Make sure we can't join with invalid class code. Tests pass if we get the error on the dialog that complains.
4. **invalidLongClassTest:** Sometimes if we enter too many numbers, it has an exception. I fixed it, so the test should pass when the dialog complains it is an invalid code.
5. **swapClassTest:** Make sure after joining two classes, we can switch classes as well. Passes when the page has a view that correctly reads the current class, even after switching.

Note: some of the frontend UI test actually tests the connection between frontend and backend as well. For example, the test for stat, quiz, wrong questions and creating/deleting/joining a class can only pass if the backend sends the data correctly.

## Backend unit test

In **CPEN321-Quizzical/Back-end/test/classMocked.test.js** - 3 test suites, 9 tests:

1. Get general class info: This is a GET /classes test with a valid class code 2 that is in the database, we expect the server to respond with status code 200 to indicate success and it should reply with the corresponding class info  

```
("[{\"classCode\":2,\"uid\":\"2\",\"category\":\"English\",\"className\":\"testClass2\",\"instructorUID\":\"2\"}]")
```
2. Get general class info 2: This is a GET /classes test with a valid class code 1 that is in the database, we expect the server to respond with status code 200 to indicate success and it should reply with the corresponding class info  

```
("[{\"classCode\":1,\"uid\":\"1\",\"category\":\"Math\",\"className\":\"testClass1\",\"instructorUID\":\"1\"}]")
```
3. Get general class info with class code not exist: This is a GET /classes test with a class code 11111 that is not in the database, we are still expecting status code 200 to indicate that the server does not crash with an invalid input and it should reply with the following message ("[]") indicating that there is no such a class.
4. Get class 2 quiz module: This is a GET /classes test with a valid class code 2 that is in the database, we expect the server to respond with status code 200 to indicate success and it should reply with the corresponding quizModule list  

```
("{\"category\":\"English\",\"classCode\":2,\"id\":0,\"moduleName\":\"module1\"}")
```
5. Get class 1 quiz module: This is a GET /classes test with a valid class code 1 that is in the database, we expect the server to respond with status code 200 to indicate success and it should reply with the corresponding quizModule list  

```
("{\"category\":\"Math\",\"classCode\":1,\"id\":0,\"moduleName\":\"module1\"}")
```
6. Get class quiz module with class undefined: This is a GET /classes test with a class code 11111 that is not in the database, we are still expecting status code 200 to indicate that the server does not crash with an invalid input and it should reply with an empty string indicating that there is no data.
7. Student delete a class: This is a DELETE /classes/delete test with a valid classCode, type=deleteClass, uid=student uid, isInstructor=false test. We expect the server to delete the student from the class and respond with 204.
8. Teacher delete a class: This is a DELETE /classes/delete test with a valid classCode, type=deleteClass, uid=instructor uid, isInstructor=true test. We expect the server to delete the class from the class database and respond with 204.
9. Teacher delete a quiz module: This is a DELETE /classes/delete test with a valid classCode, type=deleteQuiz, uid=instructor uid, quizModule test. We expect the server to delete the class from the class database and respond with 204.
10. Invalid get request: This is a GET /classes test with invalid type parameter. We expect the server to reply with "invalid request".



## Results

```
npx jest --forceExit test/classMocked.test.js
```

```
PASS test/classMocked.test.js
```

```
  class test
```

```
    ✓ get general class info (28 ms)
```

```
    ✓ get general class info 2 (4 ms)
```

```
    ✓ get general class info with class undefined (4 ms)
```

```
  class quiz module test
```

```
    ✓ get class 2 quiz module (4 ms)
```

```
    ✓ get class 1 quiz module (4 ms)
```

```
    ✓ get class quiz module info with class undefined (2 ms)
```

```
  delete test
```

```
    ✓ student delete a class (5 ms)
```

```
    ✓ teacher delete a class (4 ms)
```

```
    ✓ teacher delete a quiz (9 ms)
```

```
  test invalid gets
```

```
    ✓ invalid get (3 ms)
```

## Coverage

```
npx jest --forceExit --coverage test/classMocked.test.js
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	74.78	64.71	61.9	74.78	
class.js	97.4	84.62	100	97.4	51,84
firebasePush.js	28.95	0	11.11	28.95	22-43,49-56,62-69,75-82

We have mocked firebasePush.js in the test. The remaining 28.95% lines of firebasePush.js that are triggered are just some basic database and firebaseAdmin setup. None of the actual functions are triggered.

Line 51, 84 in class.js are error handling code that should never be triggered as long as the database is set up correctly.

In **CPEN321-Quizzical/Back-end/test/quizMocked.test.js** - 5 test suites, 13 tests:

1. fetchDataForTeachers case of router.get("/") with one student class: This is a test with a valid classCode 2, quizCode 0, type score, userId 4 and isInstructor=true parameters trying to get from /quiz. We expect the server to respond with status code 200 to indicate success and it should reply with the following message  
(["[{\"username\":\"student1\",\"quiz0score\":100}],100,100"])
2. fetchDataForTeachers case of router.get("/") with class undefined: This is a test with a valid quizCode 0, type score, userId 4 and isInstructor=true parameters, but invalid classCode 3 trying to get from /quiz. We expect the server to respond with status code

200 to indicate that the server does not crash and it should reply with the following message (`"[[[],null,-1]"`) indicating that there is no data based on the information provided.

3. `fetchDataForTeachers` case of `router.get("/")` with a class with more than 1 student: This is a test with a valid `classCode 1`, `quizCode 0`, type `score`, `userId 4` and `isInstructor=true` parameters trying to get from `/quiz`. We expect the server to respond with status code 200 to indicate success and it should reply with all the students ranked by score with average and highest score  
`("[[{"username\":\"student1\",\"quiz0score\":100},{\"username\":\"student2\",\"quiz0score\":75},{\"username\":\"student3\",\"quiz0score\":80}],85,100)")`
4. `fetchDataForTeachers` case of students wrong counts: This is a test with a valid `classCode 0`, `quizCode 0` parameters trying to get from `/quiz/studentWrongCounts`. We expect the server to respond with status code 200 to indicate success and since there is no person got wrong on the quiz, we expect the message to be empty string.
5. `fetchDataForStudents` case of `router.get("/")` with student highest: This is a test with a valid `classCode 1`, `quizCode 0`, type `score`, `userId 1` and `isInstructor=false` parameters trying to get from `/quiz`. We expect the server to respond with status code 200 to indicate success and it should reply with a list of average, highest and student's score  
`("[85,100,100]")`
6. `fetchDataForStudents` case of `router.get("/")` with some other student: This is a test with a valid `classCode 1`, `quizCode 0`, type `score`, `userId 1` and `isInstructor=false` parameters trying to get from `/quiz`. We expect the server to respond with status code 200 to indicate success and it should reply with a list of average, highest and student's score  
`("[85,100,75]")`
7. `fetchDataForStudents` case of `router.get("/")` with class undefined: This is a test with a valid `classCode 3`, `quizCode 0`, type `score`, `userId 1` and `isInstructor=false` parameters trying to get from `/quiz`. We expect the server to respond with status code 200 and it should reply with (`"[null,-1,null]"`) to indicate that the data based on the query cannot be found.
8. `fetch an existing quiz`: This is a test with a valid `classCode 1` and `quizCode 1` to get from `/quiz`. We expect the server to respond with status code 200 to indicate success and it should reply the corresponding quiz info  
`("[{\"classCode\":1,\"moduleName\":\"module2\",\"uid\":\"1\",\"courseCategory\":\"Math\",\"instructorUID\":\"1\",\"questionList\": [{\"hasPic\":false,\"category\":\"Math\",\"choices\": [{\"isPic\":false,\"str\":\"5\"},{\"isPic\":false,\"str\":\"6\"}],\"correctAnsNum\":1,\"index\":1,\"picSrc\":\"\",\"question\":\"2+3=?\",\"questionType\":\"MC\"}],\"quizCode\":1}]")`
9. `fetch an non-existing quiz` (quiz module does not exist): This is a test with a valid `classCode 1`, but invalid `quizCode 0` to get from `/quiz`. We expect the server to respond with status code 200 to indicate the server does not crash and it should reply with (`"[]"`) to indicate that there is no such a quiz in the class.

10. fetch an non-existing quiz (class not exist): This is a test with an invalid classCode 3 and quizCode 0 to get from /quiz. We expect the server to respond with status code 200 to indicate the server does not crash and it should reply with ("[]") to indicate that there is no such a quiz.
11. get wrong questions: This is a test trying to GET student wrong question counts, wrong question list for teachers, and wrong question list for students. For the student wrong question counts, it should return ("[1]") since there is only one question and only one person got wrong on the question as set up in the database. For the question list for teachers and wrong question list for the student, it should return (

```
("[{\"HasPic\":false,\"category\":\"Math\",\"choices\":[{\"isPic\":false,\"str\":\"2\"},{\"isPic\":false,\"str\":\"3\"}],\"correctAnsNum\":1,\"index\":1,\"picSrc\":\"\",\"question\":\"1+1=?\",\"questionType\":\"MC\"}]")
```

) The question list containing that specific question.
12. Calculate Average function test: This is a function test testing **function calculateAverageScore** in **quiz.js**, we feed a list of scores for two quizzes as the first input into the function, and the function is expected to return the average of each quiz respectively based on the second selection input.
13. Find Max Score function test: This is a function test testing **function findMaxScore** in **quiz.js**, we feed a list of scores for two quizzes as the first input into the function, and the function is expected to return the max score of each quiz respectively based on the second selection input.
14. Find Student Score function test: This is a function test testing **function findMaxScore** in **quiz.js**, we feed a list of data with student uid, math, chemistry and english scores as the first input into the function, and the function is expected to return the student's score for a specific subject based on student uid and quiz name (math, chemistry, english).

## Results

```
npx jest --forceExit test/quizMocked.test.js
PASS test/quizMocked.test.js
  fetchDataForTeachers
    ✓ fetchDataForTeachers case of router.get("/") with one student class (26 ms)
    ✓ fetchDataForTeachers case of router.get("/") with class undefined (4 ms)
    ✓ fetchDataForTeachers case of router.get("/") with a class with more than 1 student (4 ms)
    ✓ fetchDataForTeachers case of students wrong counts (3 ms)
  fetchDataForStudents
    ✓ fetchDataForStudents case of router.get("/") with student highest (4 ms)
    ✓ fetchDataForStudents case of router.get("/") with some other student (3 ms)
```

- ✓ fetchDataForStudents case of router.get("/") with class undefined (4 ms)
- fetch quiz
  - ✓ fetch an existing quiz (3 ms)
  - ✓ fetch an non-existing quiz (quiz module does not exist) (4 ms)
- ✓ fetch an non-existing quiz (class not exist) (3 ms)
- wrong question test
  - ✓ get wrong questions (12 ms)
- Calculate Average function
  - ✓ it should calculate the average score of quizScoreField values from the input data array (1 ms)
- Find Max Score function
  - ✓ it should find the highest quizScoreField value in the input data array (1 ms)
- Find Student Score function
  - ✓ it should check through the input data array for an element with matching studentID and return its quizScoreField value (1 ms)

## Coverage

```
npx jest --forceExit --coverage test/quizMocked.test.js
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	95.96	85.71	100	95.92	
quiz.js	95.96	85.71	100	95.92	56,70,100,114

The uncovered lines are error handling for toArray functions which should never be triggered.

In **CPEN321-Quizzical/Back-end/test/studentLeaderboardMocked.test.js** - 3 test suites, 6 tests:

- get leaderboard with student highest: This is a test with valid classCode, userId and isInstructor=false parameters trying to get from /studentLeaderboard. We are trying to get the leaderboard for the student in the first place. We expect the server to respond with status code 200 to indicate success and it should reply with the following message
 

```
(["uid\":\"1\", \"username\":\"student1\", \"EXP\":10, \"score\":100}, {"uid\":\"2\", \"username\":\"student2\", \"EXP\":9, \"score\":100}, {"uid\":\"3\", \"username\":\"student3\", \"EXP\":8, \"score\":100}, {"uid\":\"4\", \"username\":\"student4\", \"EXP\":7, \"score\":100}, {"uid\":\"5\", \"username\":\"student5\", \"EXP\":6, \"score\":100}, {"uid\":\"6\", \"username\":\"student6\", \"EXP\":5, \"score\":100}, {"uid\":\"7\", \"username\":\"student7\", \"EXP\":4, \"score\":100}, {"uid\":\"8\", \"username\":\"student8\", \"EXP\":3, \"score\":100}, {"uid\":\"9\", \"username\":\"student9\", \"EXP\":2, \"score\":100}, {"uid\":\"10\", \"username\":\"student10\", \"EXP\":1, \"score\":100}])
```

"8\\",\\"username\\":\\"student8\\",\\"EXP\\":3,\\"score\\":100},{\\"uid\\":\\"9\\",\\"username\\":\\"student9\\",\\"EXP\\":2,\\"score\\":100},{\\"uid\\":\\"10\\",\\"username\\":\\"student10\\",\\"EXP\\":1,\\"score\\":100},1,{\\"uid\\":\\"11\\",\\"username\\":\\"student11\\",\\"EXP\\":10,\\"score\\":100}]]") with first 10 items in the list being the first 10 places ranked by EXP in the class, the 11th item being the students' place which is 1 in this case and the 12th item being the student's data.

2. get leaderboard with student lowest (out of 10): This is a test with valid classCode, userId and isInstructor=false parameters trying to get from /studentLeaderboard. We are trying to get the leaderboard for the student outside the first 10 places. We expect the server to respond with status code 200 to indicate success and it should reply with the following message

("["{\\"uid\\":\\"1\\",\\"username\\":\\"student1\\",\\"EXP\\":10,\\"score\\":100},{\\"uid\\":\\"2\\",\\"username\\":\\"student2\\",\\"EXP\\":9,\\"score\\":100},{\\"uid\\":\\"3\\",\\"username\\":\\"student3\\",\\"EXP\\":8,\\"score\\":100},{\\"uid\\":\\"4\\",\\"username\\":\\"student4\\",\\"EXP\\":7,\\"score\\":100},{\\"uid\\":\\"5\\",\\"username\\":\\"student5\\",\\"EXP\\":6,\\"score\\":100},{\\"uid\\":\\"6\\",\\"username\\":\\"student6\\",\\"EXP\\":5,\\"score\\":100},{\\"uid\\":\\"7\\",\\"username\\":\\"student7\\",\\"EXP\\":4,\\"score\\":100},{\\"uid\\":\\"8\\",\\"username\\":\\"student8\\",\\"EXP\\":3,\\"score\\":100},{\\"uid\\":\\"9\\",\\"username\\":\\"student9\\",\\"EXP\\":2,\\"score\\":100},{\\"uid\\":\\"10\\",\\"username\\":\\"student10\\",\\"EXP\\":1,\\"score\\":100},11,{\\"uid\\":\\"11\\",\\"username\\":\\"student11\\",\\"EXP\\":0,\\"score\\":100}]]") with first 10 items in the list being the first 10 places ranked by EXP in the class, the 11th item being the students' place which is 11 in this case and the 12th item being the student's data.

3. get leaderboard with student somewhere in between: This is a test with valid classCode, userId and isInstructor=false parameters trying to get from /studentLeaderboard. We are trying to get the leaderboard for the student somewhere between 1-10 places. We expect the server to respond with status code 200 to indicate success and it should reply with the following message

("["{\\"uid\\":\\"1\\",\\"username\\":\\"student1\\",\\"EXP\\":10,\\"score\\":100},{\\"uid\\":\\"2\\",\\"username\\":\\"student2\\",\\"EXP\\":9,\\"score\\":100},{\\"uid\\":\\"3\\",\\"username\\":\\"student3\\",\\"EXP\\":8,\\"score\\":100},{\\"uid\\":\\"4\\",\\"username\\":\\"student4\\",\\"EXP\\":7,\\"score\\":100},{\\"uid\\":\\"5\\",\\"username\\":\\"student5\\",\\"EXP\\":6,\\"score\\":100},{\\"uid\\":\\"6\\",\\"username\\":\\"student6\\",\\"EXP\\":5,\\"score\\":100},{\\"uid\\":\\"7\\",\\"username\\":\\"student7\\",\\"EXP\\":4,\\"score\\":100},{\\"uid\\":\\"8\\",\\"username\\":\\"student8\\",\\"EXP\\":3,\\"score\\":100},{\\"uid\\":\\"9\\",\\"username\\":\\"student9\\",\\"EXP\\":2,\\"score\\":100},{\\"uid\\":\\"10\\",\\"username\\":\\"student10\\",\\"EXP\\":1,\\"score\\":100},5,{\\"uid\\":\\"5\\",\\"username\\":\\"student5\\",\\"EXP\\":6,\\"score\\":100}]]") with first 10 items in the list being the first 10 places ranked by EXP in the class, the 11th item being the students' place which is 5 in this case and the 12th item being the student's data.

4. get student leaderboard as a teacher: This is a test with valid classCode, userId and isInstructor=true parameters trying to get from /studentLeaderboard. We are expected to get all the students ranked by EXP as follows:

```
"[{\"uid\": \"1\", \"username\": \"student1\", \"EXP\": 10, \"score\": 100},
{ \"uid\": \"2\", \"username\": \"student2\", \"EXP\": 9, \"score\": 100}, {
 \"uid\": \"3\", \"username\": \"student3\", \"EXP\": 8, \"score\": 100}, { \"
uid\": \"4\", \"username\": \"student4\", \"EXP\": 7, \"score\": 100}, { \"ui
d\": \"5\", \"username\": \"student5\", \"EXP\": 6, \"score\": 100}, { \"uid\
\": \"6\", \"username\": \"student6\", \"EXP\": 5, \"score\": 100}, { \"uid\":
 \"7\", \"username\": \"student7\", \"EXP\": 4, \"score\": 100}, { \"uid\": \"
8\", \"username\": \"student8\", \"EXP\": 3, \"score\": 100}, { \"uid\": \"9\
\", \"username\": \"student9\", \"EXP\": 2, \"score\": 100}, { \"uid\": \"10\
\", \"username\": \"student10\", \"EXP\": 1, \"score\": 100}, { \"uid\": \"11\
\", \"username\": \"student11\", \"EXP\": 0, \"score\": 100}]]".
```

5. get student leaderboard with less than or equal to 10 students in total: in this test, we drop a student and get the student leaderboard again. This test is similar to the tests above, just to increase branch coverage.
6. User Position function test: This is a function test testing **function getUserPosition** in **studentLeaderboard.js**, we feed a list of data with student uid into the function and the function is expected to return the position of a specific uid in the list together with its uid.
7. Refactor Data function test: This is a function test testing **function refactorData** in **studentLeaderboard.js**, we feed a list of data with more than 10 student uids and the function is expected to return the first 10 elements in the list together with the position of the given uid in the list and its corresponding element.

## Results

```
npx jest --forceExit test/studentLeaderboardMocked.test.js
```

```
PASS test/studentLeaderboardMocked.test.js
```

```
test student leaderboard
```

- ✓ get leaderboard with student highest (58 ms)
- ✓ get leaderboard with student lowest (out of 10) (6 ms)
- ✓ get leaderboard with student somewhere in between (5 ms)
- ✓ get student leaderboard as a teacher (4 ms)
- ✓ get student leaderboard with less than or equal to 10

```
students in total (8 ms)
```

```
User Position function
```

- ✓ it should return position of matching uid user in input data array (1 ms)

```
Refactor Data function
```

- ✓ it should return an array with up to the first 10 elements of the input array, with the output of getUserPosition appended (1 ms)

## Coverage

```
npx jest --forceExit --coverage
test/studentLeaderboardMocked.test.js
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
studentLeaderboard.js	100	100	100	100	

## Backend integration test

Note: as writing profile images requires long encoded strings in the test file and reading/writing files on the travis server which causes troubles. There is no test related to the profile image handling at the backend. Also some of the errors will never be triggered if we set up the database correctly, and this will always happen, so the if (err) branches are never triggered. We also have some scheduled functions (in **firebasePush.js**) for timed (daily, weekly, monthly) push notifications, and these functions are not testable. The test coverage is together with all the unit tests above without mock (in the files with no mocked in filenames, identical to the files above, but integrated with other modules).

In **CPEN321-Quizzical/Back-end/test/integration.test.js** - 2 test suites, 8 tests:

1. test create account and get user data (username, email, EXP, user quiz count): Firstly make a POST request to create an account with the following body: { uid: "1", type: "userInfo", data: {username: "student1", email: "student1@ubc.ca", isInstructor: false, userQuizCount: 0, EXP: 0}}. And try to get each of the username, email, isInstructor, userQuizCount, EXP fields. The expected status codes are 200 and the expected outputs should exactly match the values in the POST request.
2. Test invalid get request: trying to GET with invalid type in the users endpoint. They should reply with "request invalid"
3. Test change username: firstly make a POST request with { uid: "1", type: "username", data: "newUserName1"} to change the username. Then make a username GET request. We expect the username replied to be newUserName1.
4. Test change email: firstly make a POST request with { uid: "1", type: "Email", data: "cpen321@ece.ubc.ca"} to change the user email. Then make an email GET request. We expect the email replied to be cpen321@ece.ubc.ca.
5. Test class list: firstly make a POST request with {"category\\":\\"Math\\",\\"classCode\\":35608,\\"className\\":\\"test2\\",\\"instructorUID\\":\\"2\\"} to update user's class list. Then make a class list GET request. We expect the class list replied to be {"category\\":\\"Math\\",\\"classCode\\":35608,\\"className\\":\\"test2\\",\\"instructorUID\\":\\"2\\"}.



6. Test notification frequency post and get: firstly make a POST request with `{ uid: "1", type: "notificationFrequency", data: {notificationFrequency: 0, firebaseToken: "aaa"} }` to create the field of notification frequency for the user. Then make a GET request. We expect the value replied to be 0 (notification frequency).
7. Test notification frequency update: firstly make a POST request with `{ uid: "1", type: "notificationFrequency", data: {notificationFrequency: 1, firebaseToken: "aaa"} }` to change the notification frequency value of the user to 1. Then make a GET request. We expect the value replied to be 1 (the new notification frequency).
8. Test invalid notification frequency get request: we try to GET from the notification endpoint with type username. We expect the string replied to be "request invalid".
9. create class and a student join the class: firstly a user create a class using POST request with `{"uid": "1", "type": "createClass", "data": {"category": "Math", "classCode": 1, "className": "test3", "instructorUID": "1"} }`. Then a student joins the class and gets the info of the class. We expect the output the student get to be `"[{ \"category\": \"Math\", \"classCode\": 1, \"className\": \"testClass1\", \"instructorUID\": \"1\" } ]"`.
10. Join a class that does not exist: a student join a class with class code that does not exist in the database. The server should respond with `[]`.
11. Test delete class: similar to delete class in the class test.
12. get leaderboard with teacher lowest (out of 10): this test is similar to the student leaderboard. But now we are trying to get the leaderboard for teachers with `userId 14`. It should screen out the student and reply with the following message  

```
(["[{ \"uid\": \"1\", \"EXP\": 10, \"username\": \"instructor1\" }, { \"uid\": \"2\", \"EXP\": 9, \"username\": \"instructor2\" }, { \"uid\": \"3\", \"EXP\": 8, \"username\": \"instructor3\" }, { \"uid\": \"4\", \"EXP\": 7, \"username\": \"instructor4\" }, { \"uid\": \"5\", \"EXP\": 6, \"username\": \"instructor5\" }, { \"uid\": \"6\", \"EXP\": 5, \"username\": \"instructor6\" }, { \"uid\": \"7\", \"EXP\": 4, \"username\": \"instructor7\" }, { \"uid\": \"8\", \"EXP\": 3, \"username\": \"instructor8\" }, { \"uid\": \"9\", \"EXP\": 2, \"username\": \"instructor9\" }, { \"uid\": \"10\", \"EXP\": 1, \"username\": \"instructor10\" }, 11, { \"uid\": \"14\", \"username\": \"instructor11\", \"EXP\": 0 } ]"])
```
13. get leaderboard with teacher somewhere in between: this test is similar to the student leaderboard. But now we are trying to get the leaderboard for teachers with `userId 5`. It should screen out the student and reply with the following message  

```
(["[{ \"uid\": \"1\", \"EXP\": 10, \"username\": \"instructor1\" }, { \"uid\": \"2\", \"EXP\": 9, \"username\": \"instructor2\" }, { \"uid\": \"3\", \"EXP\": 8, \"username\": \"instructor3\" }, { \"uid\": \"4\", \"EXP\": 7, \"username\": \"instructor4\" }, { \"uid\": \"5\", \"EXP\": 6, \"username\": \"instructor5\" }, { \"uid\": \"6\", \"EXP\": 5, \"username\": \"instructor6\" }, { \"uid\": \"7\", \"EXP\": 4, \"username\": \"instructor7\" }, { \"uid\": \"8\", \"EXP\": 3, \"username\": \"instructor8\" }, { \"uid\": \"9\", \"EXP\": 2, \"username\": \"instructor9\" }, { \"uid\": \"10\", \"EXP\": 1, \"username\": \"instructor10\" }, 11, { \"uid\": \"5\", \"username\": \"instructor11\", \"EXP\": 0 } ]"])
```



```
ame\":"instructor9\"},",{"uid\":"10\","EXP\:1,\"username\":"instructor10\"},5,{"uid\":"5\","username\":"instructor5\","EXP\:6}
]")
```

14. get leaderboard with less than or equal to 10 teachers: similar to above, but increase test coverage for the situation where there are less than or equal to 10 teachers in total.
15. Create quiz test: simulate a user creating a quiz. The front end will upload quiz data, and updated user experience and quiz count. We expect the server to respond 200 and when we get the EXP we should see a 10 in increase, and when we get the user quiz count we should see a 1 in increase.
16. Student do quiz test: simulate a student doing a quiz. The student will GET the quiz created in test 14. The test results will be POST to the server and a like to the quiz will also be POSTed. We expect the server to respond 200 for all the operations. We should also see an increase of 15 in student's EXP, 5 in teacher's EXP and 1 in student's user quiz count.
17. Get wrong questions: now it's an integrated version of get wrong questions. Firstly, GET the wrong question (counts) for teachers. There is one question in the test and only one student did the question wrong. So the reply is expected to be (number of people who got wrong on each question: "[1]" and question list:
 

```
"[{"HasPic\:false,\"category\":"Math\","choices\":[{"isPic\:false,\"str\":"2\"},{\"isPic\:false,\"str\":"3\"}],\"correctAnsNum\:1,\"index\:1,\"picSrc\":"\"\",\"question\":"1+1=?\",\"questionType\":"MC\"}]")
```

 Then, we fetch the wrong question list for each student. If the student got the question wrong, the expected output is (question list:
 

```
"[{"HasPic\:false,\"category\":"Math\","choices\":[{"isPic\:false,\"str\":"2\"},{\"isPic\:false,\"str\":"3\"}],\"correctAnsNum\:1,\"index\:1,\"picSrc\":"\"\",\"question\":"1+1=?\",\"questionType\":"MC\"}]")
```

 Otherwise, the expected output is an empty string indicating the student got all correct. All the response code should be 200.
18. Upload invalid quiz type: simulates user trying to upload with an invalid type, nothing should be done. Just return 200 indicating the server does not crash.
19. Upload invalid status type: similar to above
20. Get initial page: to achieve full coverage for index.js
21. Get error page: to achieve full coverage for app.js

## Results

```
npx jest --forceExit test/integration.test.js
```

```
PASS test/integration.test.js
```

```
test account related post/get requests
```

- ✓ test create account and get user data (username, email, EXP, user quiz count) (64 ms)
- ✓ test invalid get request (7 ms)
- ✓ test change username (8 ms)
- ✓ test change email (5 ms)

- ✓ test class list (5 ms)

test notification related post/get

- ✓ test notification frequency post and get (10 ms)
- ✓ test notification frequency update (5 ms)
- ✓ test invalid notification frequency get request (4 ms)

test create/join class, create quiz modules

- ✓ create class and a student join the class (19 ms)
- ✓ joining a class that does not exist (22 ms)
- ✓ test delete class (4 ms)

test instructor leader board

- ✓ get leaderboard with teacher lowest (out of 10) (4 ms)
- ✓ get leaderboard with teacher somewhere in between (4 ms)
- ✓ get leaderboard with less than or equal to 10 teachers (7

ms)

some useless testing for index.js and app.js (just to increase coverage)

- ✓ get initial page (224 ms)
- ✓ get error (11 ms)

quiz integration test

- ✓ create quiz (18 ms)
- ✓ student do quiz (23 ms)
- ✓ get wrong questions (16 ms)

upload invalid type

- ✓ quiz invalid type (2 ms)
- ✓ instructor status invalid type (3 ms)

```

-----|-----|-----|-----|-----|-----
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----
All files |    92.09 |    73.26 |    91.74 |    92.08 |
Back-end  |    100   |    50    |    100   |    100   |
  app.js   |    100   |    50    |    100   |    100   | 43-46
Back-end/routes |    91.57 |    73.81 |    91.59 |    91.56 |
  class.js |    97.4  |    88.46 |    100   |    97.4  | 51,84
  emailSending.js |    100   |    50    |    100   |    100   | 26
  firebasePush.js |    57.89 |    75    |    33.33 |    57.89 | 24,49-56,62-69,75-82
  index.js |    100   |    100   |    100   |    100   |
  instructorLeaderboard.js |    100   |    100   |    100   |    100   |
  quiz.js  |    96.97 |    89.29 |    100   |    96.94 | 56,100,114
  studentLeaderboard.js |    100   |    100   |    100   |    100   |
  upload.js |    92.22 |    58.11 |    95    |    92.22 | 47,73,86-101,141-142,297
  users.js |    87.21 |    80    |    94.12 |    87.21 | 136-152
-----|-----|-----|-----|-----|-----

Test Suites: 8 passed, 8 total
Tests:       84 passed, 84 total
Snapshots:   0 total
Time:        7.594 s
Ran all test suites.
The command "npm test" exited with 0.
store build cache

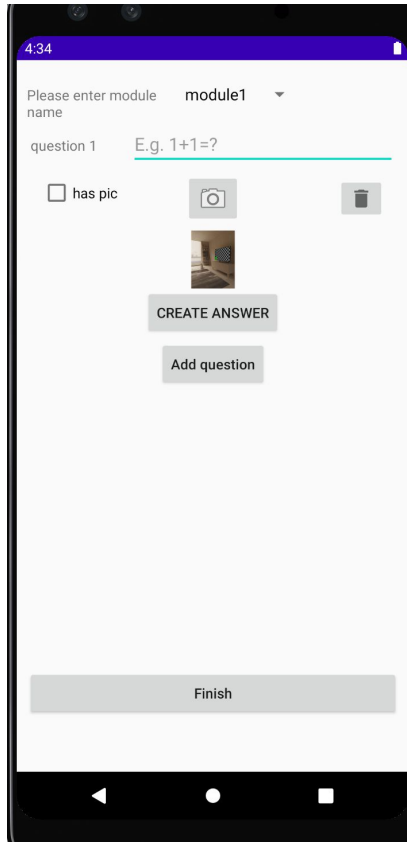
```

#### Note:

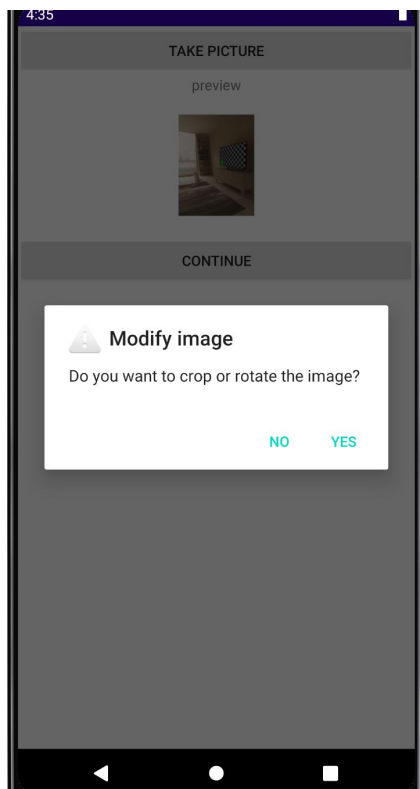
1. line 43-46 in app.js is mainly for local error handler in development and does not interact with the front end.
2. Line 51,84 in class.js are error data checking, we believe that we have had test that covering these two lines, but the uncovered line just does not go away
3. Line 26 in emailSending.js is error handling for sending emails, it is usually not triggered, because we made sure that when a user registers, his/her email address is always on the server and can be used in email sending
4. Line 49-56, 62-69, 75-82 in firebasePush.js are scheduled push for daily, weekly and monthly push notifications and can hardly be triggered by automated tests. (You need to trigger these at specific time) line 24 is an error guarding which cannot be triggered.
5. Line 56, 100, 114 in quiz.js are like line 51, 84 in class.js. We have made tests to cover these error handling, but it seems that we still lack tests for something in there/
6. Line 86-101 in upload.js and Line 136-152 in user.js are code handling reading and writing profile images to server storage. Writing tests for these on Travis causes trouble.
7. Line 47,73 in upload.js are error handling for toArray functions, which should never be triggered.
8. Line 141-142, 297 in upload.js are error handling codes which should never be triggered as long as we set up the database correctly.
9. There is showing one more test (exp.test.js) in the report which is not related to actual code testing. That is just for testing travis setup.

## Tests for non-functional requirements

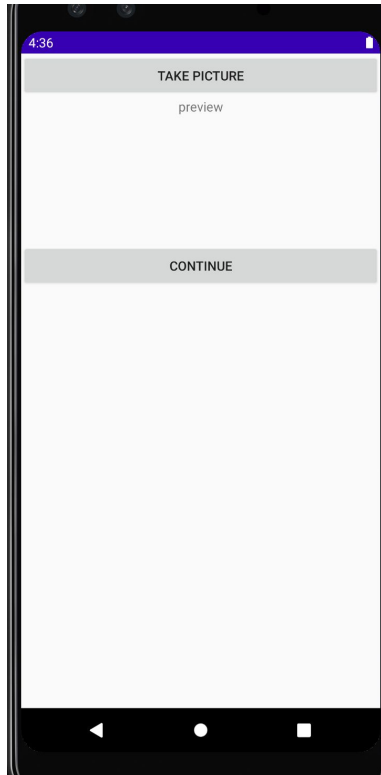
1. The user should be informed before they are trying to do something that may cause unexpected consequences. For example, logging out, quitting quiz and quitting during quiz creation. (all tests passed)
  - a. testLogoutInfo in **CPEN321-Quizzical/Front-end/app/src/androidTest/java/com/cpen321/quizzical/ProfileFragmentTest.java**: when the user clicked on the “Logout” button in the profile fragment. A warning should be shown, asking whether or not the user really wants to log out. If the user clicks “no”, we should not log the user out and keep the user in the home activity (profile fragment). We check this by confirming that the “Logout” button is still in the view.
  - b. testBackPressed in **CPEN321-Quizzical/Front-end/app/src/androidTest/java/com/cpen321/quizzical/CreateQuizTest.java** and **CPEN321-Quizzical/Front-end/app/src/androidTest/java/com/cpen321/quizzical/QuizActivityTest.java**: when user presses back button in QuizActivity or CreateQuizActivity. We need to double check that the user indeed wanted to quit. Here we showed an example of pressing the back button while creating a new quiz. The logic in the QuizActivity is the same. We check that when the user clicks on the back button, the AlertDialog is correctly set up with the correct message to be shown (in this case, asking whether the user actually wants to discard the quiz he/she is making). We made two test cases. One for the user pressing the “no” button. In this case, the app should not exit or end the activity. The other for the user pressing the “yes” button. In this case, the activity should be terminated and we check that the app goes to the correct activity (home activity with quiz layout in the view).
  - c. testDeleteModule and testDeleteClass in **CPEN321-Quizzical/Front-end/app/src/androidTest/java/com/cpen321/quizzical/InstructorCreateClassTest.java**: these two checks whether a warning is properly set up when a teacher tries to delete a quiz module or a class. The setup is similar to the tests above. We are checking whether the Alert Dialog with the proper message has been set up. And if the user presses “no”, nothing should be done to the module or the class. Otherwise, the class or the module should be deleted. These two tests are integrated into CreateClassTest as after test clean up.
2. The teacher should be able to use only one picture for the whole question. (manual testing)
  - a. We should be able to use pictures used in the question to make up choices directly without taking the picture again.



(Originally we have a picture for the question)






(when we get to take a picture for the answers, the picture is shown directly, we can modify it)





Please enter module name module1 ▼

question 1 E.g. 1+1=?



☐ has pic  



☐ correct answer  

**CREATE ANSWER**

question 2 E.g. 1+1=?

☐ has pic  

**CREATE ANSWER**

**Add question**

(if we get to a new question, then the picture will not



interfere)



- b. We should be able to use pictures used in one of the choices to make up other choices, and in the question field.

4.3.7

Please enter module name **module1**

question 1 E.g. 1+1=?

☐ has pic  

☐ correct answer  

**CREATE ANSWER**


**Add question**

**Finish**


(We create a picture for one of the choice)

**TAKE PICTURE**

preview



**CONTINUE**

 **Modify image**

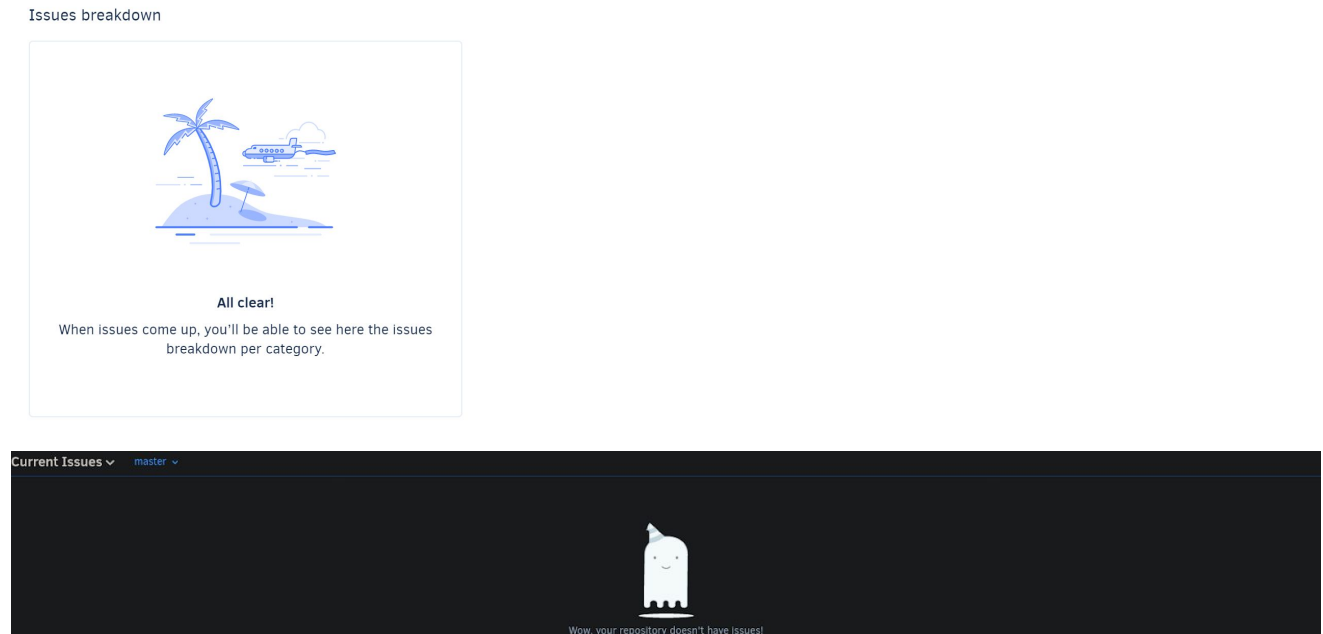
Do you want to crop or rotate the image?

**NO** **YES**

(when we try to create a new answer or add a picture to the question)

## 4. Code Review

Screenshot of code issues (default setup without disables)



Justifications for remaining issues (if any)

Nothing to see here.

## 5. “Humblebrag”

1. We beautified our app with customized round cornered buttons and round cornered rectangle boxes
2. We added some animation to the floating action buttons. When you click on the buttons, there will be some simple rotation showing.
3. We have both English and Chinese version UI for the app automatically switching based on the user's system language setting.