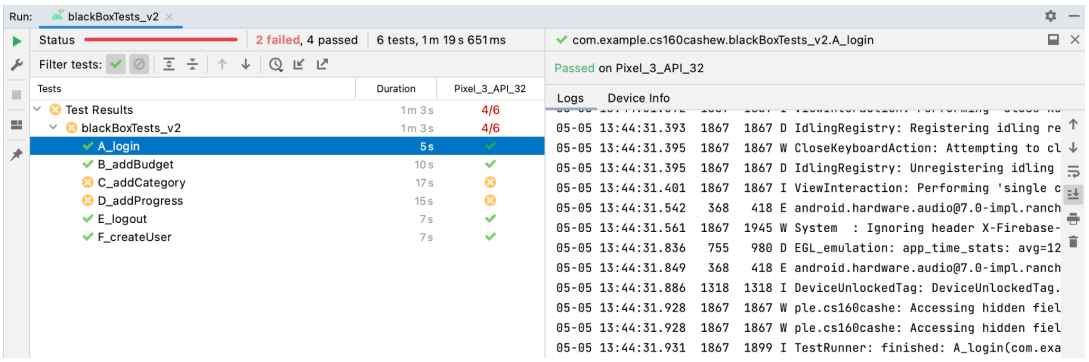
CS160 Testing Document

1. Project Title and Authors
   List of all team members (Names and SJSU ID)
   Jason Hammar: jason.hammar@sjsu.edu
   Student ID: 012568517
   Min-yuan Lee: min-yuan.lee@sjsu.edu
   Student ID: 014247051
   Aidan Kormanik: aidan.kormanik@sjsu.edu
   Student ID: 012719694
   Bernard Tan: bernard.tan@sjsu.edu
   Student ID: 015215317
   Minh Hung Nguyen: minhhung.nguyen@sjsu.edu
   Student ID: 014949649
   Andrei Titoruk: andrei.titoruk@sjsu.edu
   Student ID: 015193958
   Margaret Li: margaret.li01@sjsu.edu
   Student ID: 014443221

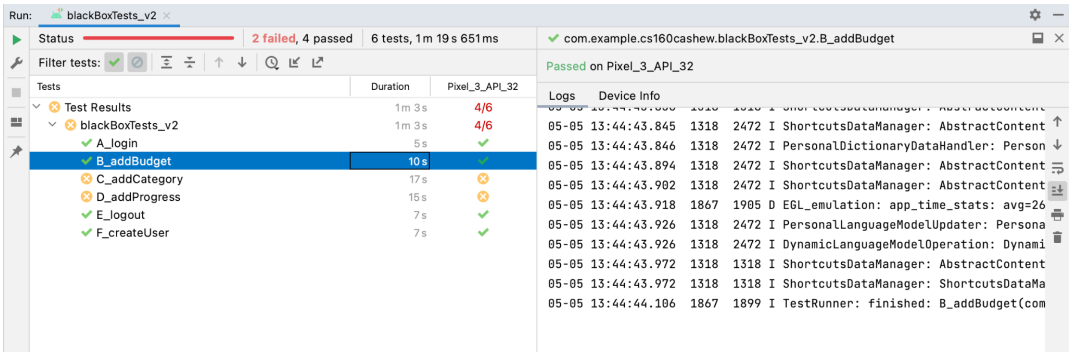
2. Preface
   The purpose of CASHew is an Android Budget Management App that helps users manage their spending habits. Everyone struggles with how to spend their money, and what to spend their money on. This app will help change that while making it interactive and fun for the user.

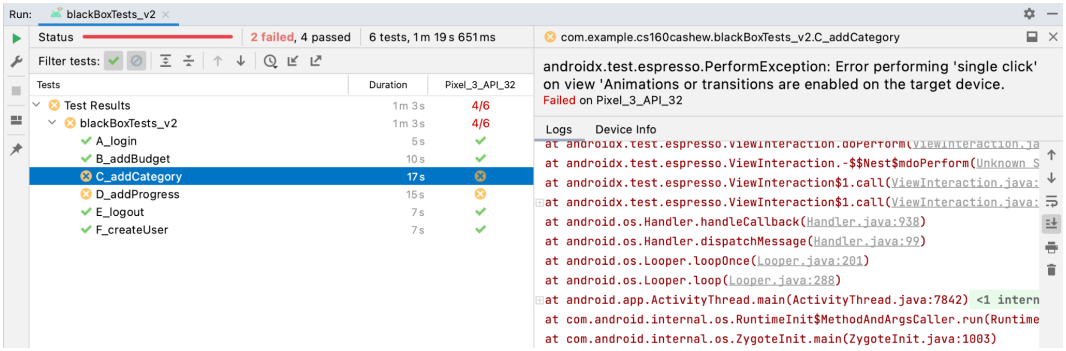
3. Instructions
   - To run the black-box tests, navigate to the blackBoxTests.java file inside the project. It's contained within the androidTest folder in the project. Or the file path is: “CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests_v2.java”. You can either run all the tests at once with the “>>” symbol next to line 38, or you can navigate to the specific test you want to run and press the “>” next to it.
   - To run the white-box tests, navigate to the test files in the project. They are contained within the androidTest folder in the project. Or the path is CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew”. There are four test suite files, BudgetTest.java, CategoryTest.java, LimitTest.java, and UserTest.java. Once you have a test file selected, you can either run all the tests in that test suite at once with the “>>” symbol next to class declaration line, or you can navigate to the specific test you want to run and press the “>” next to it.

4. Black-box Tests

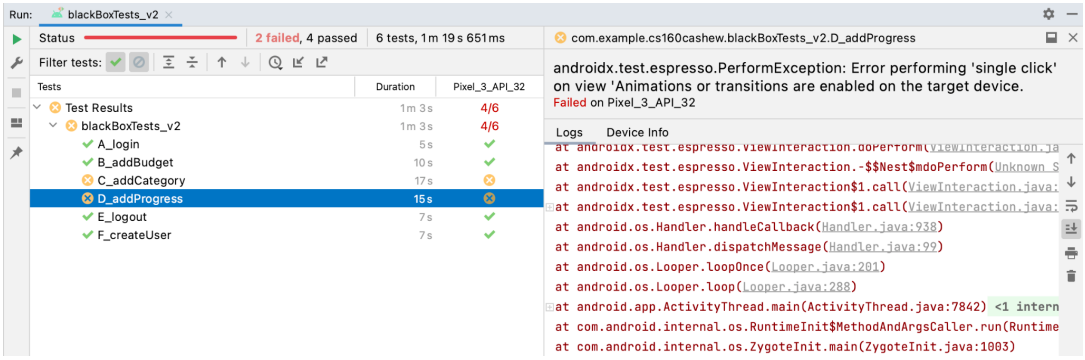
- Test 1:
    - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests_v2.java > A_login()
    - This case tests whether the app will accept a pre-existing login and successfully accept the username and password to get into the user's list of budgets. To execute the test, run the A_login() test inside the blackBoxTests_v2.java file.
    - The rationale behind this test is to use a specific, known login to make sure that it works. For the email, we used test@email.com and the password is password. This is a very basic account that will simply be used for tests. The test was generated by using Espresso and recording the interaction with the emulator so that Espresso can automatically re-run the test.
    - 

    - The test passes when the app successfully logs into the account and moves to the budget layout. If the login fails, then the test also fails.
    - Previous versions of this test did help uncover that we didn't have a uniform loading screen. Now that we have implemented that, this test passes each time.
- Test 2:
    - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests_v2.java > B_addBudget()
    - This case tests whether a user can create a new generic budget. It uses an average name and limit for the budget to test the basic functionality. To execute the test, run the B_addBudget() test inside the blackBoxTests_v2.java file.
    - The rationale behind this test is to use a general name and limit for a new budget to see if the app properly creates the new budget and displays it inside the app. The test was generated by using Espresso and recording the interaction with the emulator so that Espresso can automatically re-run the test.

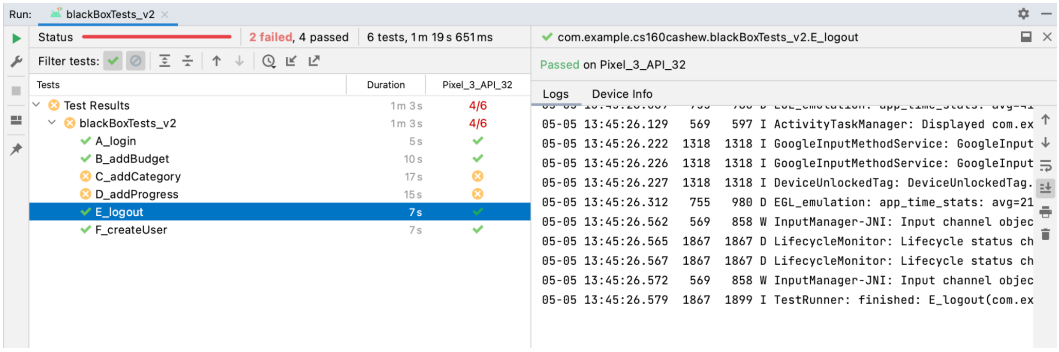
- 

  The test passes when the app successfully displays the created budget. It fails if the budget isn't displayed properly.
  - This test hasn't uncovered any potential bugs but will continue to run to ensure the functionality continues to work.
- Test 3:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests_v2.java > C_addCategory()
  - This case tests whether a user can create a new category inside of a budget but using a generic name value to test it. To execute the test, run the C_addCategory() test inside the blackBoxTests_v2.java file.
  - The rationale behind this test is to see if the functionality of creating categories is working properly. The test was generated by using Espresso and recording the interaction with the emulator so that Espresso can automatically re-run the test.
- 

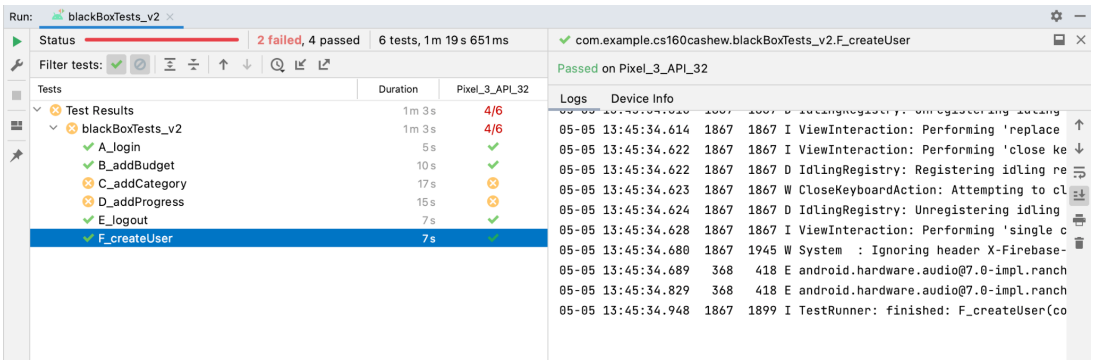
  The test passes when the app successfully creates the category. It fails if the category isn't created properly.
  - This test did help discover that the button to add a category is not always visible to the user or, more often, partially cut off. This is a potential issue if a user can't locate the ability to create a budget. We will move the button in order to resolve the issue.
- Test 4:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests_v2.java > D_addProgress()
  - This case tests whether a user can input progress that they have made to their budget. It'll attempt to add a progress of $500 and see if it deducts it from the budget's limit. To execute the test, run the D_addProgress() test inside the blackBoxTests_v2.java file.

- The rationale behind this test is to test the functionality of adding a transaction they have done. It's important to have this functionality because that's the whole point of the app, to see if their spending has stayed inside their budget limit. The test was generated by using Espresso and recording the interaction with the emulator so that Espresso can automatically re-run the test.
- 

- The test passes when the progress is successfully added and adjusts the limit left.
- This test did help discover that the button to add progress is not always visible to the user or, more often, partially cut off. This is a potential issue if a user can't locate the ability to add progress to their budget. We will move the button in order to resolve the issue.

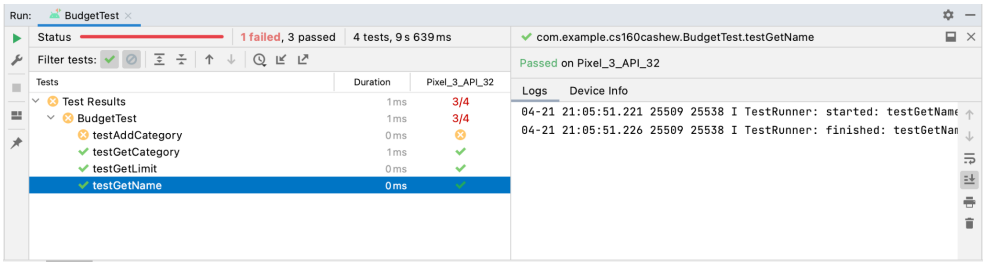
- Test 5:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests_v2.java > E_logout()
  - This case tests whether a user can logout of their account within the app. To execute the test, run the E_logout() test inside the blackBoxTests_v2.java file.
  - The rationale behind this test is to make sure that the logout feature of the app works. This feature is important to ensure security within the app so that if the user logs out, then no one else can access the account. The test was generated by using Espresso and recording the interaction with the emulator so that Espresso can automatically re-run the test.
  - 

  - The test passes when the app successfully logs out of the user's account. It fails, if the logout was not completed correctly.
  - This test hasn't uncovered any issue but will continue to run to ensure it stays that way.

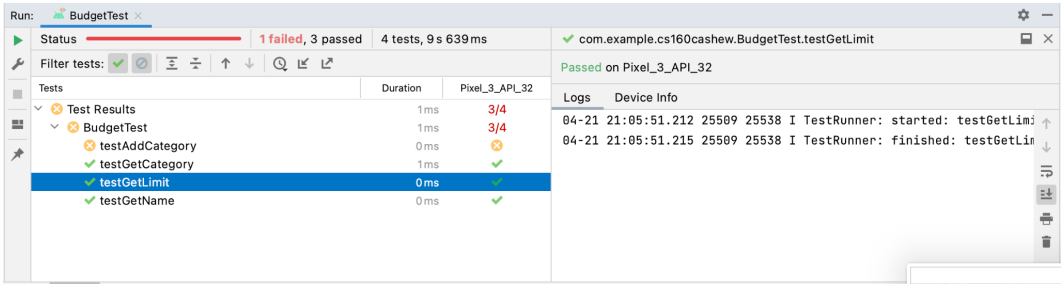
- Test 6:
    - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests_v2.java > F_createUser()
    - This case tests whether the app can create a new user with valid inputs. It's important for this functionality to always be available so that new users can use the app. To execute the test, run the F_createUser() test inside the blackBoxTests_v2.java file.
    - The rationale behind this test is to make sure that an account can be created on the creation page. The case creates a first and last name, a random new email, a password, and a phone number to then try and create a new account. The test was generated by using Espresso and recording the interaction with the emulator so that Espresso can automatically re-run the test.
    - 

    - The test passes if the app creates the new account It fails if the account isn't created.
    - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.

5. White-Box Tests
    - Test 1:
        - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > BudgetTest.java > testGetName()
        - This case tests whether the budget class can properly get the name of any given budget object. To execute the test, run the testGetName() test inside the BudgetTest.java file.
        - The rationale behind this test is to make sure that at any given moment, a budget item should be able to access its proper name. The test was generated using JUnit inside of Android studio and can be run automatically.

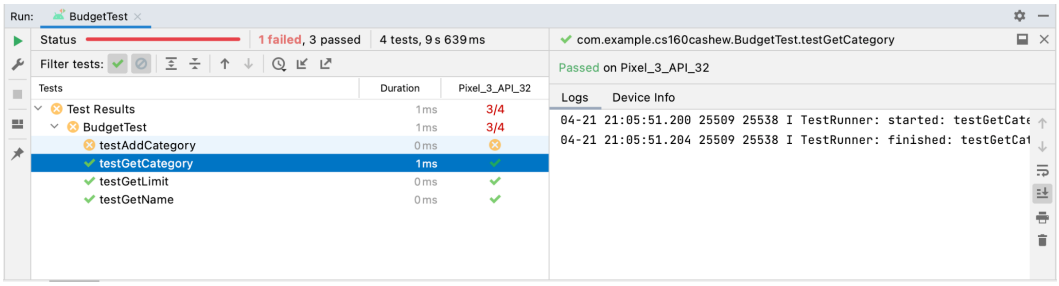
- 

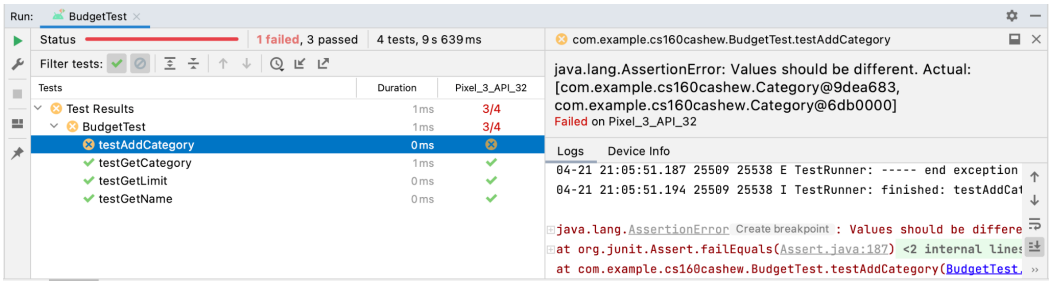
  The test passes if the budget's name is correct and fails if it is wrong.
  - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.

- Test 2:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > BudgetTest.java > testGetLimit()
  - This case tests whether the budget class can properly get the limit of any given budget object. To execute the test, run the testGetLimit() test inside the BudgetTest.java file.
  - The rationale behind this test is to make sure that at any given moment, a budget item should be able to access its proper spending limit. The test was generated using JUnit inside of Android studio and can be run automatically.
  - 

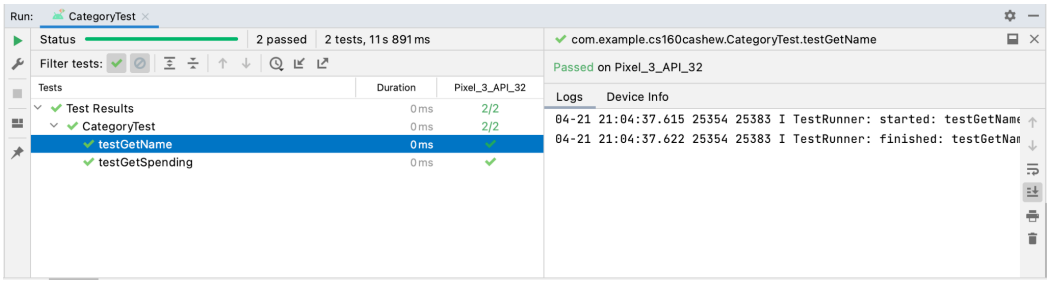
    The test passes if the budget's spending limit is correct and fails if it is wrong.
    - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.

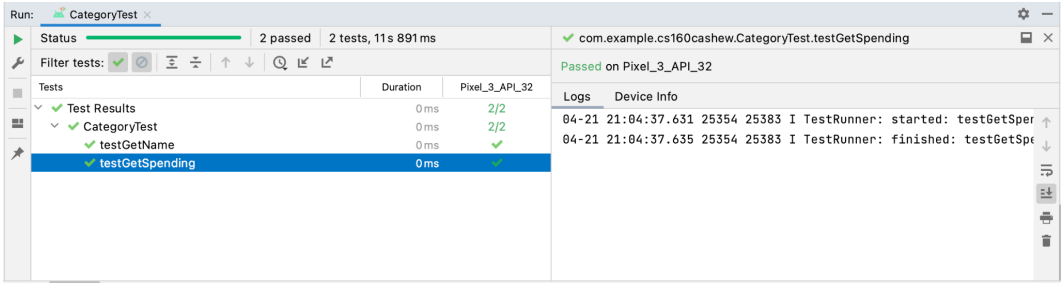
- Test 3:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > BudgetTest.java > testGetCategory()
  - This case tests whether the budget class can properly get the category list of any given budget object. To execute the test, run the testGetCategory() test inside the BudgetTest.java file.
  - The rationale behind this test is to make sure that at any given moment, a budget item should be able to access its proper category list if it exists. The test was generated using JUnit inside of Android studio and can be run automatically.

- 

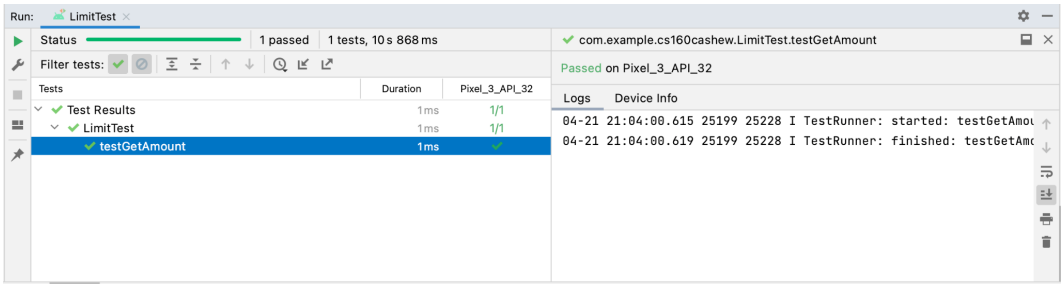
  The test passes if the budget's category list exists and fails if it's empty.
  - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.
- Test 4:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > BudgetTest.java > testAddCategory()
  - This case tests whether the budget class can properly add a category to the category list of any given budget object. To execute the test, run the testAddCategory() test inside the BudgetTest.java file.
  - The rationale behind this test is to make sure that when a new category is added by the user, then the budget's category list should be updated. The test was generated using JUnit inside of Android studio and can be run automatically.
- 

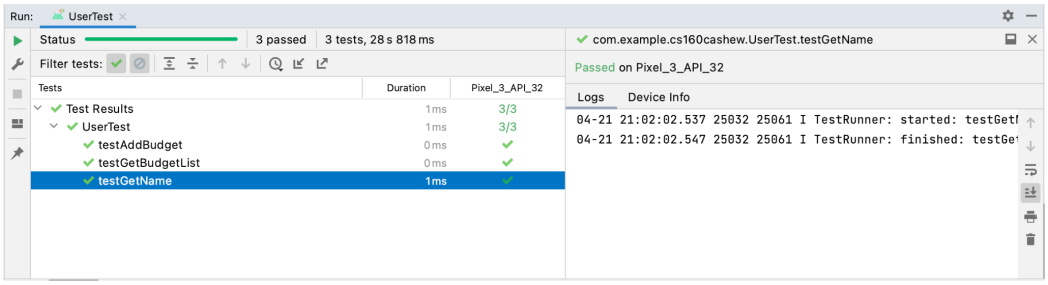
  The test passes if the budget's category list is updated after adding a new category and fails if it doesn't update.
  - This test has uncovered that when updating the list, there is a potential issue saving the new data. The fix this, we need to ensure that updated budgets are kept and saved for the user.
- Test 5:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > CategoryTest.java > testGetName()
  - This case tests whether the category class can properly get the name of any given category object. To execute the test, run the testGetName() test inside the CategoryTest.java file.
  - The rationale behind this test is to make sure that at any given moment, a category item should be able to access its proper name. The test was generated using JUnit inside of Android studio and can be run automatically.

- 

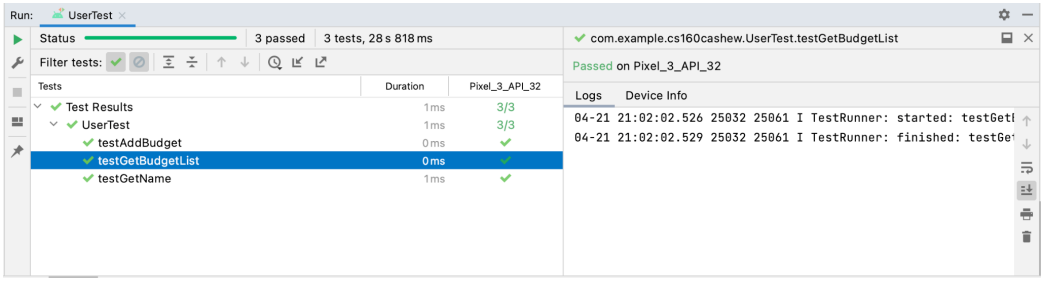
  - The test passes if the category's name is correct and fails if it is wrong.
  - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.
- Test 6:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > CategoryTest.java > testGetSpending()
  - This case tests whether the category class can properly get the spending limit of any given category object. To execute the test, run the testGetSpending() test inside the CategoryTest.java file.
  - The rationale behind this test is to make sure that at any given moment, a category item should be able to access its proper spending limit. The test was generated using JUnit inside of Android studio and can be run automatically.
  - 

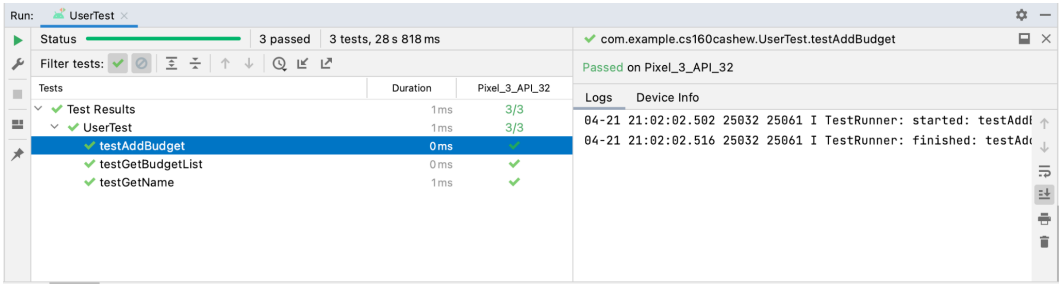
  - The test passes if the category's spending limit is correct and fails if it is wrong.
  - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.
- Test 7:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > LimitTest.java > testGetAmount()
  - This case tests whether the limit class can properly get the limit amount of any given limit object. To execute the test, run the testGetAmount() test inside the LimitTest.java file.
  - The rationale behind this test is to make sure that at any given moment, a limit item should be able to access its proper limit amount. The test was generated using JUnit inside of Android studio and can be run automatically.

- 

  - The test passes if the limit's amount is correct and fails if it is wrong.
  - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.
- Test 8:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > UserTest.java > testGetName()
  - This case tests whether the user class can properly get the name of any given user object. To execute the test, run the testGetName() test inside the UserTest.java file.
  - The rationale behind this test is to make sure that at any given moment, a user item should be able to access its proper name. The test was generated using JUnit inside of Android studio and can be run automatically.
  - 

    - The test passes if the user's name is correct and fails if it is wrong.
    - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.
- Test 9:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > UserTest.java > testGetBudgetList()
  - This case tests whether the user class can properly get the list of budgets for any given user object. To execute the test, run the testGetBudgetList() test inside the UserTest.java file.
  - The rationale behind this test is to make sure that at any given moment, a user item should be able to access the list of budgets that it contains. The test was generated using JUnit inside of Android studio and can be run automatically.

- 

  The test passes if the user's budget list is filled with the proper data and fails if it is wrong.
  - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.
- Test 10:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > UserTest.java > testAddBudget()
  - This case tests whether the user class properly updates the list of budgets if a new one is added. To execute the test, run the testAddBudget() test inside the UserTest.java file.
  - The rationale behind this test is to make sure that if a new budget is added to a user, that this new budget is updated to the budget list. The test was generated using JUnit inside of Android studio and can be run automatically.
  - 

    The test passes if the user's budget list is updated with the new data and fails if it is wrong.
    - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.