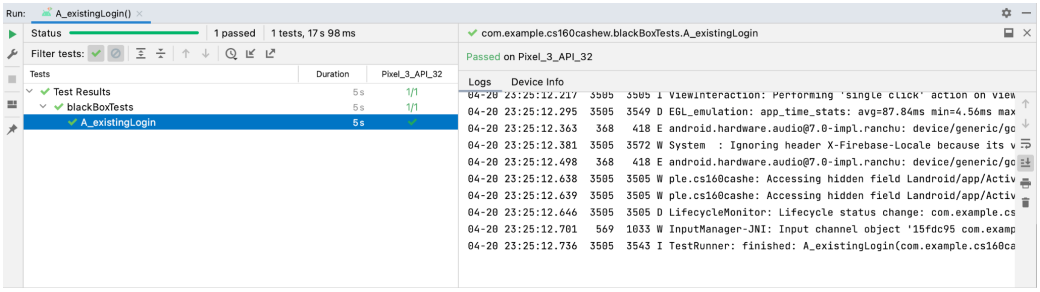
CS160 Testing Document

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

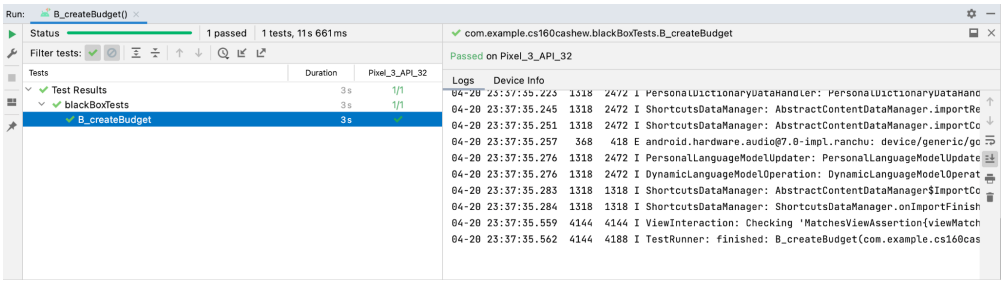
2. Preface
   The purpose of CASHew is an Android Budget Management App that helps users manage their spending habits. Everyone struggles with how to spend their money, and what to spend their money on. This app will help change that while making it interactive and fun for the user.

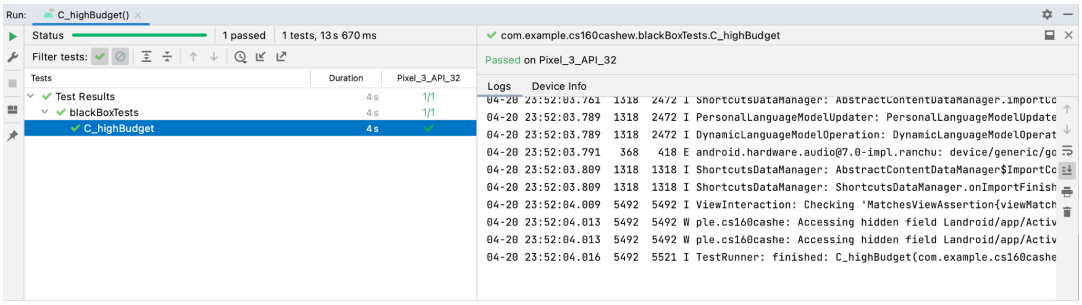
3. Instructions
   - To run the black-box tests, navigate to the blackBoxTests.java file inside the project. It's contained within the androidTest folder in the project. Or the file path is: “CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests.java”. You can either run all the tests at once with the “>>” symbol next to line 43, 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

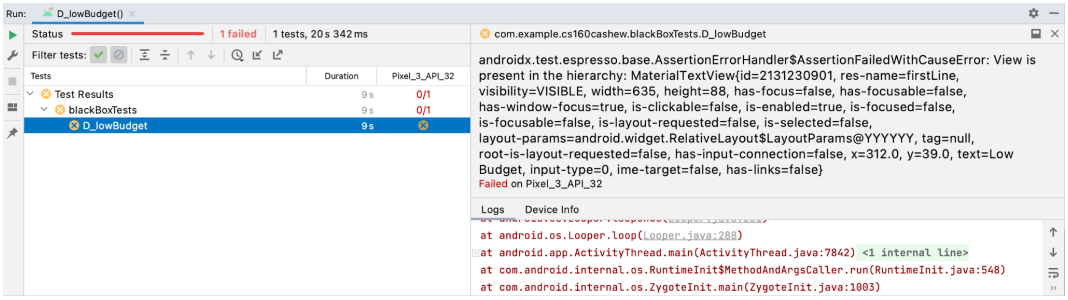
- Test 1:
    - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests.java > A_existingLogin()
    - This case tests whether the app will accept a pre-existing login and successfully accept the username and password to get into the user's list of budgets. To execute the test, run the A_existingLogin() test inside the blackBoxTests.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.
    - This test did help discover that depending on how the is run, sometimes the login screen is not the first thing that the user sees. Currently, the app sometimes has a memory that the user is already logged in, which affects trying to execute this test. To resolve this issue, we need to make sure there is a consistent starting screen, each time the app is run.
- Test 2:
    - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests.java > B_createBudget()
    - 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_createBudget() test inside the blackBoxTests.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.

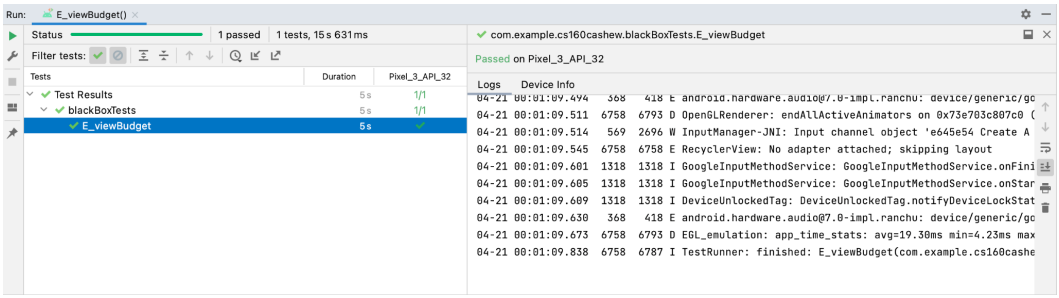
- 

  The test passes when the app successfully displays the created budget. It fails if the budget isn't displayed properly.
  - This test did help discover that sometimes it fails because a user is not logged in or needs to first. To resolve this issue, we need to make sure there is a consistent starting screen, each time the app is run.
- Test 3:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests.java > C_highBudget()
  - This case tests whether a user can create a new budget with a really high limit. It uses an average name and $1,000,000,000 for the budget's limit to test if it still creates the budget. To execute the test, run the C_highBudget() test inside the blackBoxTests.java file.
  - The rationale behind this test is to test the limit functionality when creating a budget. Certain users may have really high limits so they should be able to create these budgets without a problem. 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 did help discover that when the budget layout displays a large limit, the number could be cut off. To fix this issue, we need to adjust the parameters for displaying the budget to take into account large numbers.
- Test 4:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests.java > D_lowBudget()
  - This case tests whether a user can create a new budget with no limit. It uses an average name and $0 for the budget's limit to test if the budget is created. To execute the test, run the D_lowBudget() test inside the blackBoxTests.java file.

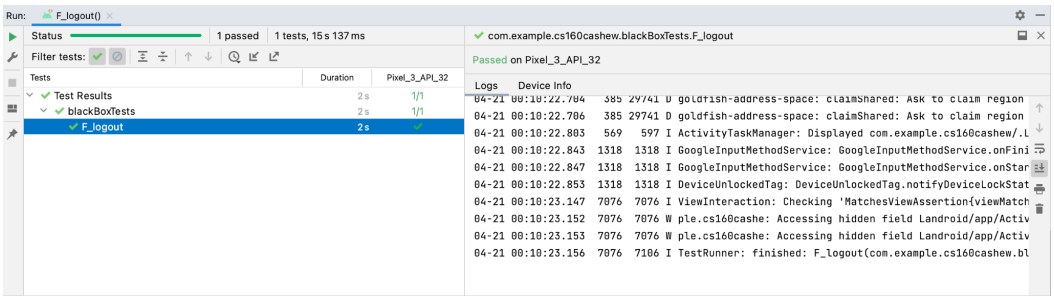
- The rationale behind this test is to test the limit functionality when creating a budget. Certain users may accidentally create a budget with no limit and the app should respond in the correct manner. The test was generated by using Espresso and recording the interaction with the emulator so that Espresso can automatically re-run the test.
- 

- The test passes when the app doesn't display the budget because there is no limit. It fails if the budget is displayed with a $0 limit.
- This test did help discover that even with no limit, the app still creates the budget. To fix this, we need to change the creation process, so that if the user puts $0 as the limit, the app displays an error and it's not accepted.

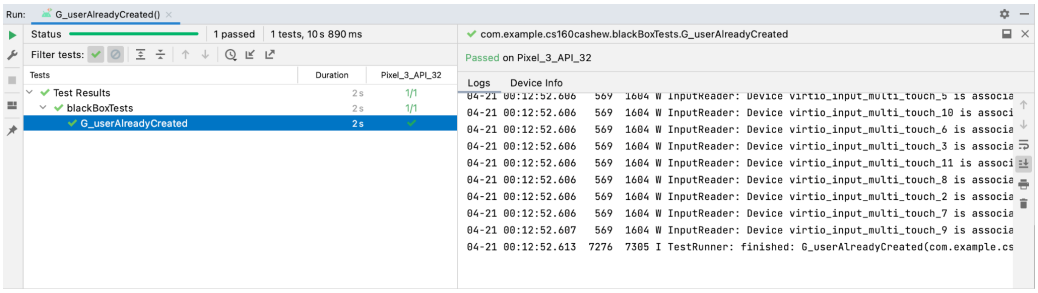
- Test 5:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests.java > E_viewBudget()
  - This case tests whether a user can view a created budget on the budget page. It'll select a budget and test whether the app successfully displays the details. To execute the test, run the E_viewBudget() test inside the blackBoxTests.java file.
  - The rationale behind this test is to test if the app successfully displays the budget's details page. This should be a consistent feature in the app so it should be tested often. 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 properly displays the budget details screen and the options on that screen are accessible. It fails, if the screen is not properly displayed.
  - This test did help discover that the options on the budget details screen need to be implemented so more. There isn't much functionality on the details page so in order to fix it, we need to implement those features.

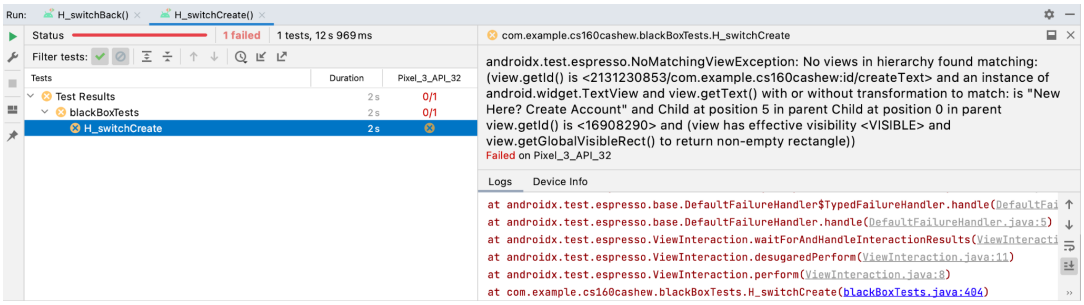
- Test 6:

- CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests.java > F_logout()
- This case tests whether a user can logout of their account within the app. To execute the test, run the F_logout() test inside the blackBoxTests.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.
- 

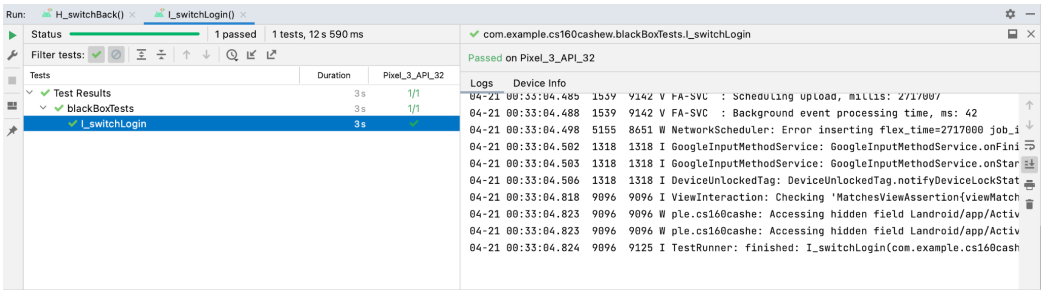
- The test passes when the app successfully logs out of the user's account. It fails, if the logout was not completed correctly.
- This test did help discover that sometimes the logout option isn't available. This ties into having a consistent splash screen so that the option is available.

- Test 7:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests.java > G_userAlreadyCreated()
  - This case tests whether the app will accept a new user who already has a pre-existing account. For this test, it tries to create a new user using the test@email.com account which already exists. To execute the test, run the G_userAlreadyCreated() test inside the blackBoxTests.java file.
  - The rationale behind this test is to make sure that the app prevents two users from having the same login credentials. If an email is already in use, the app should not allow a new user to be created under that email. 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 prevents the new user from being created. It fails, if the case was successful in creating a new account.
  - This test so far hasn't uncovered any new issues. But it will continue to be run to ensure future changes don't affect this preventive measure.

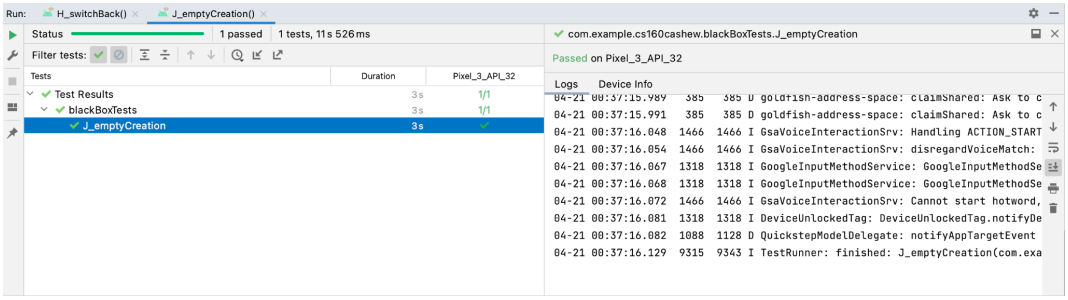
- Test 8:
    - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests.java > H_switchCreate()
    - This case tests whether the app allows the user to navigate to the create a new account page if they need to make one. To execute the test, run the H_switchCreate() test inside the blackBoxTests.java file.
    - The rationale behind this test is to make sure that the option to create a new account is accessible. For new users, if they can't create an account, it would prevent them from accessing 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 if the app properly displays the create a new user option. It fails, if that option isn't displayed
    - This test has uncovered an issue where the option is not available on the splash screen. In order to fix this issue, we need to ensure it's always available at the start page.
- Test 9:
    - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests.java > I_switchLogin()
    - This case tests whether the app allows the user to navigate to the returning user login page if they need to sign into their pre-existing account. To execute the test, run the I_switchLogin() test inside the blackBoxTests.java file.
    - The rationale behind this test is to make sure that the option to back into an already existing account is accessible. For returning users, if they can't login to their account, it would prevent them from accessing the budgets that they created. The test was generated by using Espresso and recording the interaction with the emulator so that Espresso can automatically re-run the test.



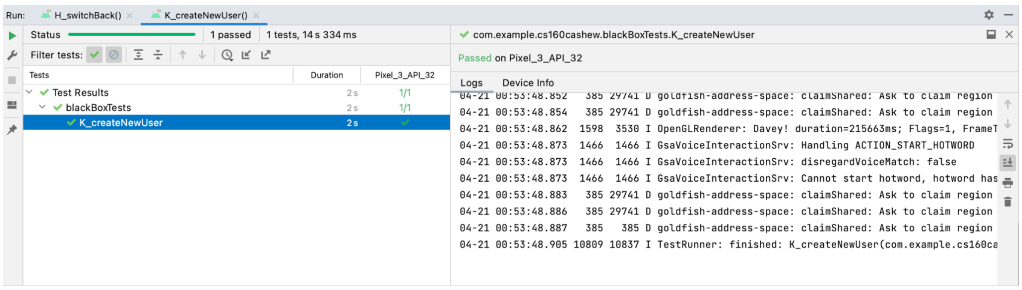
- The test passes if the app properly displays the returning user login option. It fails, if that option isn't displayed
- This test has uncovered an issue where the option is sometimes not available on the splash screen. In order to fix this issue, we need to ensure it's always available at the start page.

- Test 10:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests.java > J_emptyCreation()
  - This case tests whether the app allows the user to create a new account that is missing parts of the creation process. The test attempts to create a user with only a first name, not a full email, but missing a phone number and password. To execute the test, run the J_emptyCreation() test inside the blackBoxTests.java file.
  - The rationale behind this test is to make sure that an account can't be created if it's missing necessary parts of the creation page. Without a proper password, it would make it difficult for a user to login to their account in the future so it's important for the app to prevent this upfront. The test was generated by using Espresso and recording the interaction with the emulator so that Espresso can automatically re-run the test.

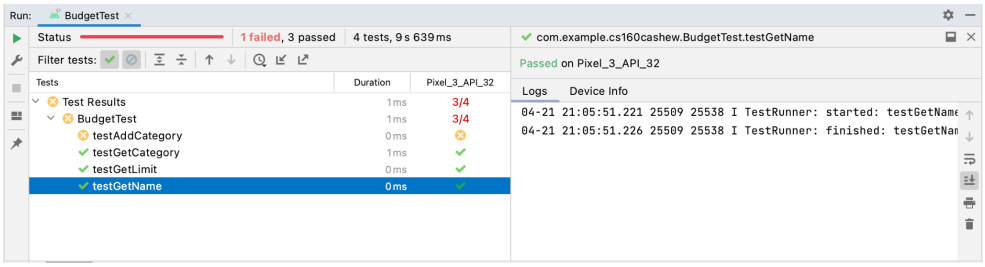


- The test passes if the app doesn't create the new account and remains on the creation page. It fails if the account is created and the app progresses forward.
- This test hasn't uncovered any issues yet because the app successfully prevents the creation of bad accounts. The case will continue to be run to ensure it remains the same in the future.

- Test 11:
  - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > blackBoxTests.java > K_createNewUser()

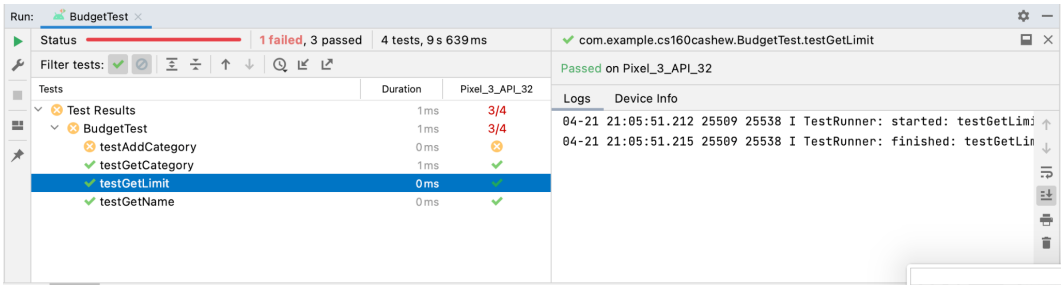
- This case tests whether the app can create a new user with valid inputs. Its important for this functionality to always be available so that new users can use the app. To execute the test, run the K_createNewUser() test inside the blackBoxTests.java file.
- The rationale behind this test is to make 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.

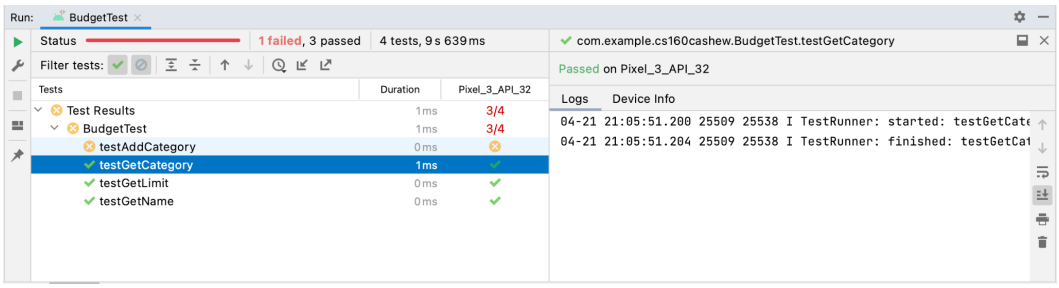
5. White-Box Tests
   - Test 1:
     - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > BudgetTest.java > testGetName()
     - This case tests whether the budget class can properly get the name of any given budget object. To execute the test, run the testGetName() 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.

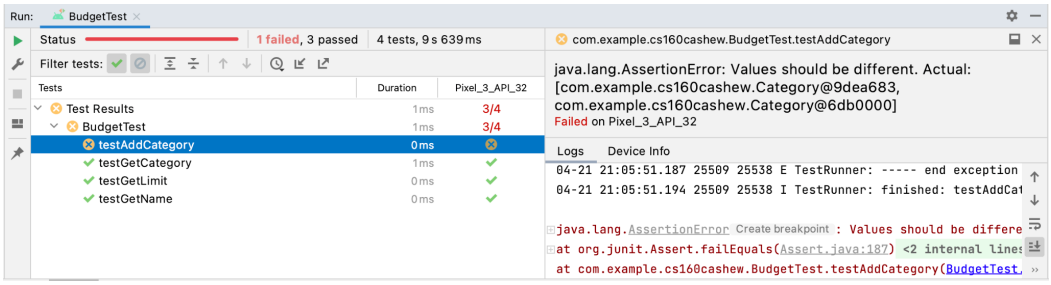
- Test 2:

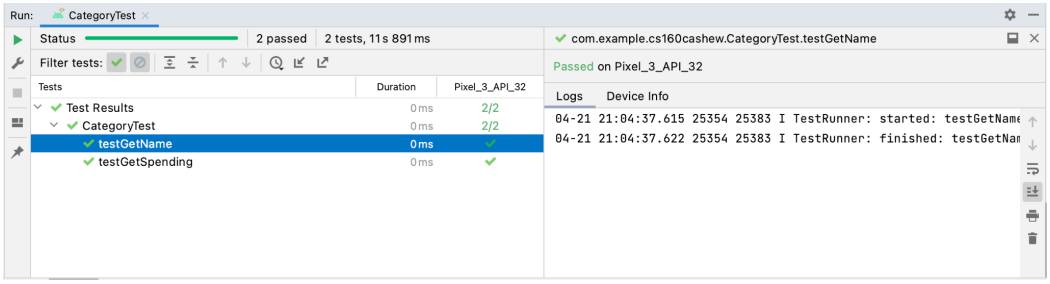
    - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > BudgetTest.java > testGetLimit()
    - This case tests whether the budget class can properly get the limit of any given budget object. To execute the test, run the testGetLimit() test inside the BudgetTest.java file.
    - 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.

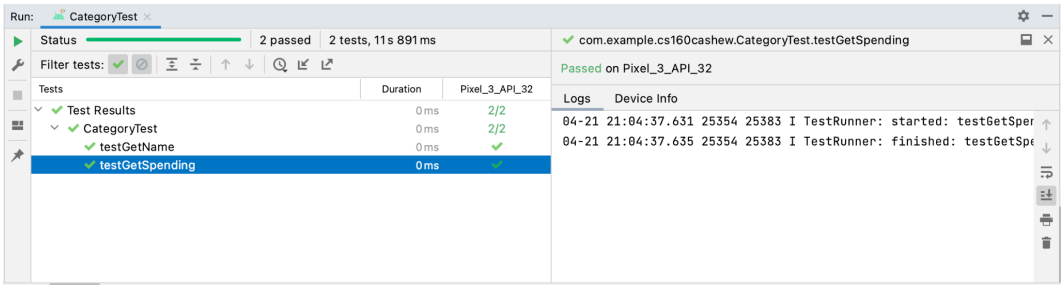
- Test 3:
    - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > BudgetTest.java > testGetCategory()
    - This case tests whether the budget class can properly get the category list of any given budget object. To execute the test, run the testGetCategory() test inside 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.
    - 

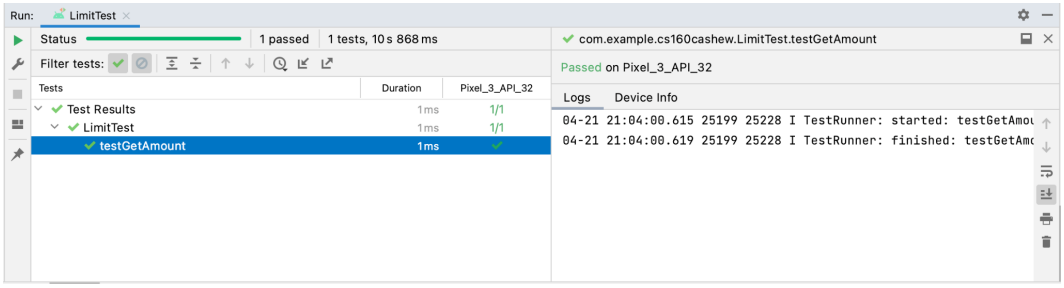
    - The test passes if the budget's category list exists and fails if it's empty.
    - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.
- Test 4:
    - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > 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.
    - 

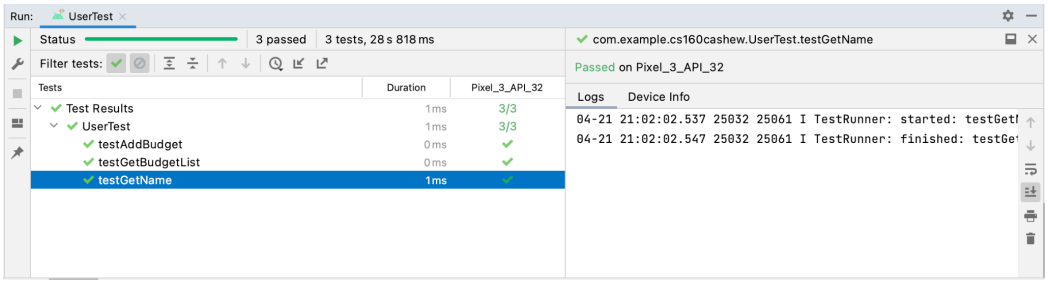
    - The test passes if the budget's category list is updated after adding a new category and fails if it doesn't update.
    - This test has uncovered that when updating the list, there is a potential issue saving the new data. The fix this, we need to ensure that updated budgets are kept and saved for 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.
    - 

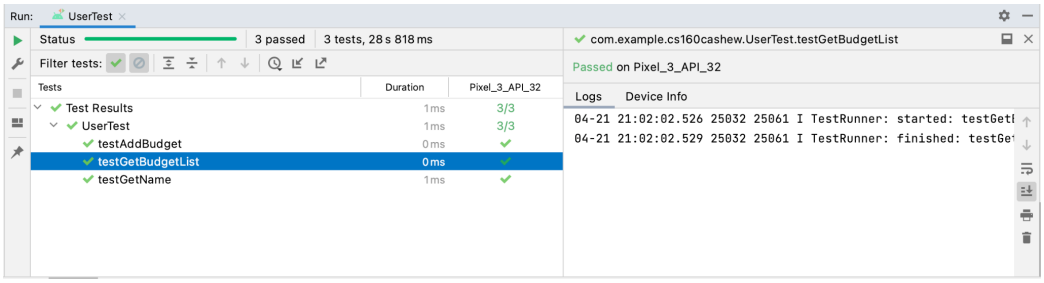
    - The test passes if the category's name is correct and fails if it is wrong.
    - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.
- Test 6:
    - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > 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.
    - 

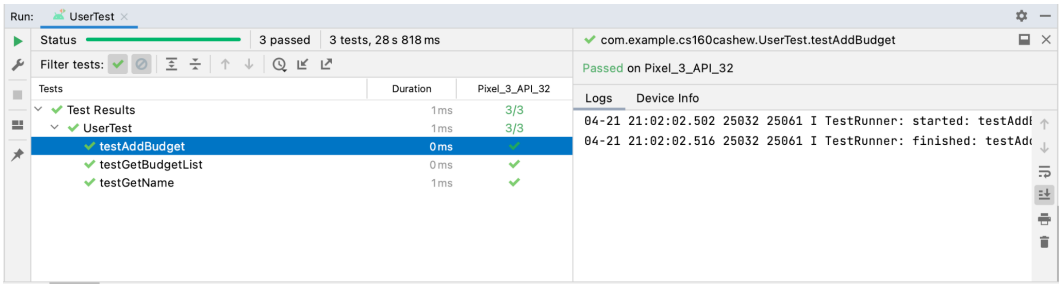
    - The test passes if the category's spending limit is correct and fails if it is wrong.
    - This test hasn't uncovered any issues yet but will continue to be run to ensure it remains the same in the future.
- Test 7:
    - CS-160-Group-Project > app > src > androidTest > java > com > example > cs160cashew > 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.