Quality Assurance Assignment 3

Frontend Requirements Testing - Group 42

Alex Mason 20019045

Ted Munn 20001247

Marc De Verteuil 20005645

Part 1: Testing Approach

We decided to forgo the templated approach altogether and use pytest in a much simpler, maintainable and readable way. All of the functionality for this program is as modular as possible, with functions performing only a single task and then passing the output to another function or to the user.

This separation allowed us to easily create unit and requirement tests for all of our important features. We created test files for each primary feature and for all other functions, including, for example, our validation functions. Separating the tests like this makes it easy to organize work on each feature and see how any changes made to that feature affect the outcome of the tests independently. It is also very readable, as individual tests are separated into individual functions, per the pytest framework, and test only one aspect of the function.

Tests are located within the tests folder, and named after their feature or purpose. Further details on errors in the code found and changing of testing strategy are provided in the failure reporting section.

/tests
- test_deposit.py
- test_transfer.py
- test_withdraw.py
- test_validate.py

Part 2: Failure Reporting

Errors found in the code

Test	**Purpose of test**	**Problem with output**	**Problem with code**	**How it was fixed**
test_transfer_invalid_amount_atm	Ensure that atm mode cannot transfer more than $10000	Transaction was created when it shouldn't have been	The transfer amount in the test was in dollars not cents	Transfer amount was converted to cents
test_login_success	To ensure the login function returns the correct boolean result given the input	Initially the function had no return output	Initially we changed a global logged_in variable in the function itself but this made it difficult to test	We now update the global variable in the controller function and just return True or False in the login function
test_X_negative_amount_machine	Ensure that a negative amount cannot be used	A negative amount was successfully used	Amount validation did not check for negative value	Added checking for negative values to fail validation
test_deposit_no_account_agent	Check for a case where account number not in valid accounts list	Error in code for exception handling	Code would throw an error and not except the case	Added exception handling for case where the provided account number is not in the account list

Errors found in testing strategy

Test	Purpose of test	Problem with test planning	How it was fixed
Test_withdraw_invalid_total_amount_atm Test_deposit_invalid_total_amount_atm test_transfer_invalid_total_amount_atm	Ensure that the atm cannot withdraw/deposit /transfer over a certain amount in a day	Original written case was overly complicated, running the features multiple times	It was easier to initialize the sum of daily limits to a value close to the max then try to go over than to run the functions multiple times
test_X_success_agent test_X_success_atm	Test each feature for successful cases for atm and agent	Original tests did not check the success of both agent and atm modes of operation	Created test cases to cover the success both of the agent and atm mode for all functions (withdraw, transfer, deposit)
X_invalid_account	Testing features where the account number is invalid	Initially had each function checking all of the cases for invalid account number input which required unnecessary reuse of code	We realized we can simply check invalid account numbers during account creation then check that the account is in the accounts list in the functions
Test_login Test_create_account	Testing the login and create account functionality in the main.py file	Original tests did not cover login functionality since it was hard to simulate command line input.	Once it was realized that we could simulate command line input using pytest, tests to cover the main file for login and create account were made