Course Project Assignment #1 - Front End Requirements

Stuart Calverley - 100522058
Jason Runzer - 100520993
Muhammad Ahmad - 100518917

Due Date: Feb 14, 2016

1)

Test Case	**Intentions**
login-functions-rejected-nologin	Makes sure the system does not allow the user to enter any other valid commands until the user is logged in.
login-standard	Ensures that the system asks for standard or admin mode and asks for the account holder's name when Standard mode is selected.
login-standard-accepted	Tests an accepted login during standard mode.
login-standard-rejected	Ensures that a invalid standard bank account login is rejected
login-gibberish	Ensures random input in the login phase is rejected.
login-admin	Checks if it asks for the account holder's name
login-standard-login-rejected	Ensures that a user is not allowed to try to login while already logged in.
login-END_OF_FILE-rejected	Ensures the end of file account is not accessible for a standard session.
logout-admin	Ensures the system correctly logs an admin user that is in a session out.
logout-standard	Ensures a standard user is logged out correctly.
withdrawal-admin	Ensures the system asks for the account holder's name and account number
withdrawal-standard	Checks to see if it asks for the account holder's account number.
withdrawal-amount	Checks to see if it asks for the amount to withdraw for admin and standard sessions.
withdrawal-double	Ensures the system rejects withdrawals of a non-double or integer amount.

withdrawal-edge	Checks to see if boundary values are rejected and accepted for standard and admin accounts. Standard accounts cannot withdraw over $500. This includes boundary tests for the amount of money in an account.
withdrawal-valid-admin	Tests whether withdrawals are rejected for accounts that do not exist for an admin session.
withdrawal-valid-standard	Tests whether withdrawals are rejected for accounts that do not exist for a standard session.
withdrawal-negative-rejected	Ensures rejection of a withdrawal that is negative for both standard and admin sessions.
withdrawal-validAmount	Tests for rejections of amounts that are not multiples of 5's. for both standard and admin sessions.
withdrawal-END_OF_FILE-rejected	Checks if the end of file account can be accessed for an admin withdrawal.
withdrawal-deposit-unavailable	Checks to see that deposited funds cannot be used for withdrawal.
withdrawal-transactionFee	Checks if the system rejects amounts on standard accounts that do not have enough money with the transaction fee.
transfer-standard	Checks the outputs of the prompt for the user to enter the account number they want to transfer to.
transfer-admin	Ensures it asks for the account holder's name and account number from and to.
transfer-validAccount	Checks whether the source account number entered to transfer money from belongs to user currently logged in. Also checks the destination account number entered exists.
transfer-validAmount	Checks to see if the amount of money you are trying to transfer is allowed. Checks to see if you have enough money in your account to transfer.

transfer-edge	Checks to see if boundary values are rejected and accepted for standard and admin accounts. Standard and Admin accounts cannot do transactions that transfer more than $1000. This includes boundary tests for the amount of money in an account
transfer-negative-rejected	Checks if the system rejects a transfer that is negative for both standard and admin sessions.
transfer-transactionFee	Makes sure the user has enough money in the account after the transaction fee has been deducted.
transfer-deposit-unavailable	Ensures deposited money cannot be used in the same session for the transfer transaction.
transfer-END_OF_FILE	Ensures the END_OF_FILE account cannot be transferred money to or from.
paybill-admin	Checks if the system asks for the account holder's name and number.
paybill-transactionFee	Makes sure the user has enough money in the account after the transaction fee has been deducted.
paybill-standard	Checks if it asks for the account number.
paybill-company	Tests if it rejects an invalid company.
paybill-amount	Checks to see if the account has enough money to pay the bill
paybill-edge	Checks the edge conditions of the paybill requirement. It checks just above $2000 and just below $2000 and boundary amount for account money.
paybill-validAccount	Checks if the account number entered belongs to user currently logged in.
paybill-negative-rejected	Ensures rejection of a paybill that is negative for both standard and admin sessions.
paybill-END_OF_FILE	Ensures rejection for the END_OF_FILE input.

paybill-deposit-unavailable	Ensures deposited money cannot be used in the same session for the paybill transaction.
deposit-standard	Checks if it asks for the account number in a standard session
deposit-admin	Checks if it asks for the account holder's name and number in an admin session.
deposit-amount	Checks if it asks for an amount to deposit for both standard and admin sessions.
deposit-double	Checks if it rejects amount that are not double/int for both a standard or admin session.
deposit-valid-standard	Ensures the system requires a valid bank account that the selected user owns.
deposit-valid-admin	Requires a valid bank account name and number for the selected user.
deposit-standard-accepted	Ensures correct output for a complete valid standard deposit transaction.
deposit-admin-accepted	Ensures correct output for a complete valid admin deposit transaction.
deposit-admin-rejected	Checks if the system rejects a admin deposit that ends in the account going over $99999.99.
deposit-standard-rejected	Checks if the system rejects a standard deposit that ends in the account going over $99999.99
deposit-admin-END_OF_FILE	Checks if the system rejects a deposit into the END_OF_FILE account.
deposit-negativeValue	Checks if the system rejects a deposit of a negative amount.
create-standard	Checks if the system rejects a create transaction in a standard session
create-admin	Ensures that it ask for the account holder's name and initial balance.

create-double	Checks if the system rejects any amount if not double/int.
create-validName	Ensures that create is rejected if digits are used in account holder name, or if the empty string is entered.
create-validAmount	Ensures the maximum amount can be $99999.99, and that an account cannot be created with negative funds.
create-available	Ensures the account is not available for other transactions in the session
create-truncate	Checks if it truncates the account name if too long.
create-unique	Ensures it creates a unique bank account number in the transaction file. This also tests that it saves the account.
delete-standard	Ensures delete is rejected for a standard session.
delete-admin	Ensures that it asks for the account holder's name and number.
delete-END_OF_FILE	Ensures it rejects a deletion of the END_OF_FILE account.
delete-existing	Ensures it requires a valid account to delete.
delete-accountNumberExist	Ensures the account number must exist.
delete-unavailable	Ensures no transactions can be accepted for the deleted account
disable-standard	Ensures disable is rejected in a standard session.
disable-validAccount	Checks if it asks for the bank account holder's name and account number.
disable-deposit	Checks if it rejects the user from depositing into their account

disable-withdrawal	Checks if it rejects the user from withdrawing from their account
disable-transfer	Checks if it rejects the user from transferring money to another account.
disable-nameMatch	Checks if the name of the admin matches the name of the account he is disabling
disable-accountNumberExist	Ensures the account number must exist to disable.
disable-accountFail	Ensures that the account has been disabled.
disable-unavailable	Checks that no transactions are accepted for the disabled account
disable-END_OF_FILE	Checks that it rejects a disable on the END_OF_FILE account.
enable-standard	Ensures enable is rejected in a standard session.
enable-admin	Checks if it ask for the bank account holder's name and account number.
enable-checkDeposit	Checks if it allows user to deposit money into their account.
enable-checkTransfer	Checks if it allows user to transfer money between accounts.
enable-checkWithdrawal	Ensures that it allows user to withdrawal money from their account.
enable-nameMatch	Makes sure the name of the admin matches the name of the account he is enabling.
enable-accountNumberExist	Ensures the account number must exist to enable.
enable-accountFail	Ensures that the transaction fails if the account is already enabled.

enable-END_OF_FILE	Ensures that it rejects an enable for the END_OF_FILE account.
changeplan-standard	Ensures that changeplan is rejected for a standard session.
changeplan-NP	Makes sure that it changes the student plan (SP) to a non-student plan (NP).
changeplan-SP	Ensures that it changes the non-student plan (NP) to a student plan (SP) .
changeplan-existing	Ensures that the account holder name must exist.
changeplan-validAccount	Ensures that the account number must exist.
changeplan-admin-END_OF_FILE	Checks if it rejects a change plan for the END_OF_FILE account.

Test Plan

Overview

Testing the Bank System will include testing each individual transaction which includes login, withdrawal, transfer, paybill, deposit, create, delete, disable, changeplan, logout and enable. Each test is atomic, or very close to being atomic, due to the restriction of having to logout after every test case.

After further examination of testing in the windows platform, we have collectively decided to switch to testing in the linux environment. This will make the scripting and input/output redirection easier.

Structure

The tests are located in a folder called Tests. This folder contains all of the directories of the unit tests of the system. The unit tests are each a separate transaction that can be completed in the system. Inside each unit test, there is an input, output and TF folder. The input folder contains all the input files for each test that is concerned with that specific transaction. The output folder will contain all of the expected output files regarding each test from the input folder. The TF folder contains all the expected transaction files regarding each test. Each test file name corresponds to the table above. If a test fails, the person debugging can easily lookup the test description according to the test name. To keep the files organized, the input files all end with a -I.txt, the output files end with a -O.txt and the expect transaction files end with a -TF.txt.

Execution

Each test will have two parts that will be evaluated as a pass or fail and will be indicated with the words PASS in green or FAIL in red in the terminal output. The first part of a test includes checking the expected output of the terminal. The second part tests whether an appropriate transaction file is created by the system. During the start of a unit test, the script will go through each file in the input directory for the test, and start redirecting the input into the program from that file. It will redirect the output of the program to a current output file with the name of the test followed by a -AO.txt in the folder name actual-output.That file will be compared with the expected output file in the output folder. These files will be compared with the diff command, and be sent to the difference-output folder to make it easier for the tests to be further analysed if it fails.This file will be checked to see if it is empty. If it is empty a PASS will be printed next to the name of the test, if not, a FAIL will be printed to the terminal next to the name.

During the start of each test, we will ensure that all transaction files in the banking system's directory are deleted. This will make it easier so we know which transaction file created by the system to check against. Once the actual transaction file is created by the system, the script will copy over the contents of the file into a different file named after the test ending in a -ATF in the folder actual-TF for debugging purposes. Just like the output, the actual transaction file will be compared with the expected transaction file using the diff command. The output of the diff command will be sent to the difference-TF folder named after the specific test. It will then be evaluated as a pass or fail.

At the end of each unit test, the number of passing and failing tests will be displayed, and after the end of all tests in the system, the number of failing and passing tests will be displayed as the final result.

Test Order

The tests will be executed in the order; login, logout, withdrawal, transfer, paybill, deposit, create, changeplan, disable, enable, delete. We will first run the login and logout tests and fix our system until the both those unit tests pass. By doing this we can ensure that the next transaction tests will not fail based on an error from using the login or logout transaction. After those tests pass, one at a time, we will add the next set of tests alongside the previous tests, and fix the program until each test passes. This will ensure that we focus on one aspect of the program at a time, and progressively fix the problems. It will also ensure that if we modify a part of the program, that the other requirements of the program that had already passed the test still continue to execute appropriately.

Conclusion

After ensuring that all tests pass for the expected output and transaction files we can be confident that the front-end of the banking system is complete and bug free.