Table of Test Cases:

Login

Transaction	Test Name	Intention
login	tc_login0	Test that user can exit without logging in
	tc_login1	Test that user can be logged in as standard user
	tc_login2	Test that user can be logged in as admin user
	tc_login3	Test that user enters existing account number
	tc_login4	Test that user enters existing account name
	tc_login5	Test that user does not leave account number empty
	tc_login6	Test that user does not leave account name empty

Logout

Transaction	Test Name	Intention
logout	tc_logout0	Test that user can log out
	tc_logout1	Test that user can login after logout

Withdrawal

Transaction	Test Name	Intention
withdrawal	tc_withdrawal0	Test that admin users can withdraw
	tc_withdrawl1	Test that standard users can withdraw
	tc_withdrawal2	Test that admin user enters existing account number
	tc_withdrawal3	Test that admin user enters existing account name

	tc_withdrawal4	Test that user does not exceed $500 limit in standard mode per session
	tc_withdrawl5	Test that user cannot withdraw money with insufficient balance
	tc_withdrawl6	Test that admin user can withdrawal above limit
	tc_withdrawal7	Test that user can make another withdrawal in the current session

Transfer

Transaction	Test Name	Intention
transfer	tc_transfer0	Test that admin user can transfer money
	tc_transfer1	Test that standard user can transfer money
	tc_transfer2	Test that admin user enters existing account number to transfer from
	tc_transfer3	Test that admin user enters existing account name to transfer from
	tc_transfer4	Test that user enters existing account number to transfer to
	tc_transfer5	Tests that user does not exceed the maximum amount of $1000 in standard mode
	tc_transfer6	Test that user cannot transfer money with insufficient balance
	tc_transfer7	Test that admin users can transfer above $1000
	tc_transfer8	Test that user can make another transfer in current session

Paybill

Transaction	Test Name	Intention
paybill	tc_paybill0	Test that admin can paybill
	tc_paybill1	Test that standard user can paybill
	tc_paybill2	Test that admin enters existing account number
	tc_paybill3	Test that admin enters existing account name
	tc_paybill4	Test that user selects existing company
	tc_paybill5	Tests that the user does not exceed the pay limit of $2000 in standard mode
	tc_paybill6	Test that admin can transfer above limit
	tc_paybill7	Tests that the user cannot pay a bill with insufficient funds
	tc_paybill8	Test that user can make another paybill in current session

Deposit

Transaction	Test Name	Intention
deposit	tc_deposit0	Test that admin can deposit
	tc_depoist1	Test that standard user can deposit
	tc_deposit2	Test that admin enters existing account number to deposit to
	tc_deposit3	Test that user enters existing account name to deposit to
	tc_deposit14	Test that user cannot access deposited funds in current session
	tc_deposit5	Test that user can make another deposit in current session

Create

Transaction	Test Name	Intention
create	tc_create0	Test that standard users can’t create account
	tc_create1	Test that admin user can create account
	tc_create2	Test that user enters valid account number to create account
	tc_create3	Tests the user enters new name with maximum 20 characters
	tc_create4	Test that user cannot set a balance above $99999.99

Delete

Transaction	Test Name	Intention
delete	tc_delete0	Test that standard users can’t delete account
	tc_delete1	Test that admin can delete account

	tc_delete2	Test that user enters existing account number
	tc_delete3	Test that user enters existing account name

Disable

Transaction	Test Name	Intention
disable	tc_disable0	Test that standard user cannot disable account
	tc_disable1	Test that admin can disable account
	tc_disable2	Test that user enters existing bank account holder number
	tc_disable3	Test that user enters existing bank account holder name

Changeplan

Transaction	Test Name	Intention
changeplan	tc_changeplan0	Test that standard user cannot change plan
	tc_changeplan1	Test that admin user can change plan
	tc_changeplan2	Test that user enters existing bank account number
	tc_changeplan3	Test that user enters existing bank account name

Test files:

All files have been added on github

Test Organization:

The phase1 directory contains test cases as its primary organization unit for this project. Every test case possesses its own separate folder containing different files that represent distinct test aspects.

Test cases adopt an established naming protocol using tc_ as the prefix followed by test type specification then sequential order numbering. For example, the first test case for login functionality is named tc_login0 while the second one is tc_login1 followed by sequential numbers for subsequent tests. The system maintains clear and consistent naming patterns for test identification purposes.

The test case structure includes four different files which perform individual roles for testing purposes.

- .in : represents user input files
- .out : represents user output files
- .ca : represents current bank account files
- .ta : represents bank account transaction files

The systematic method of test case arrangement brings multiple positive effects to testing operations. One folder that unifies related test files under each test case simplifies both data search and maintenance as well as evaluation of test data. The folder organization with its predictable sequential numbering methodology makes test addition possible at any time without affecting current test procedures. Test case re-runs become straightforward because separate files keep inputs distinct from outputs and transactions. The process to debug becomes more efficient because each test case gets separated isolation making it simple to locate and evaluate failures from expected against actual outputs. The structured approach delivers convenient management and complete functionality validation for system testing through its methods.

Test Run Plan:

An automated script will execute tests by running the system with each test input file located in the directory. The system input process takes place through an automated script which enables the application execution to create output files. The script proceeds to compare the newly generated output with the expected output file to check for differences. The system will automatically detect any deviations between actual and expected outputs which will allow immediate error or consistency detection. Through this automatized procedure the system checks its functionality consistently and reliably across different test cases.

The project will succeed with acceptance testing as its main approach because we have complete documentation of user requirements. End users can test whether all defined requirements are fulfilled by running the software while using it like a normal user. This method functions as supplemental confirmation to pinpoint undefined test cases and develop appropriate resolutions.

When testing errors occur, the first action will involve testing the actual results against the expected results contained in test files. The software code will be the first priority since small formatting problems like character space variations and wording differences typically create discrepancies. The test case will be inspected when the code itself does not have any issues

The entire development process depends on regression testing together with failure testing to maintain operational quality. New changes that get implemented must pass regression tests to verify that previously working features remain operational whereas failure tests confirm that errors fixed in the system stay eliminated. It is vital to fix regressions and recurring failures at their earliest appearance because these problems create extended challenges for resolution.

Requirements Problems:

Code situations pertaining to error management need clarification for administrative operations. The system lacks specific instructions that detail what actions occur to transaction files after an admin session is terminated after an error occurs.

For example, in withdrawal, if an admin fails to login on behalf of a user, then there is no user info to store in the transaction file. In this case we decided against producing a transaction file for these situations because no valid transaction needed recording.