**CSCI 3060U - Software Quality and Assurance**

Project Phase I - Front-End Requirements

Wenbo Zhang, Xuan Zheng, Neel samirkumar Shah, Dev rajivkumar Thaker

**Initial Test Cases List**

| Transaction | Test Name | Intention |
| --- | --- | --- |
| Login | login01 | Test Login, no username |
| login02 | Test Login, Valid privileged(Admin) username |
| login03 | Test Login, Invalid privileged(Admin) username |
| login04 | Test Login, Valid standard username |
| login05 | Test Login, Invalid standard username |
| login06 | Test subsequent Login, Valid privileged(Admin) username |
| login07 | Test subsequent Login, Valid standard username |
| login08 |  |
| login09 |  |
| login10 |  |
| Withdrawal | withdrawal01 | Test Withdrawal privileged(Admin) user, Invalid account holdername |
| withdrawal02 | Test Withdrawal privileged(Admin) user, valid account holdername, Invalid account number |
| withdrawal03 | Test Withdrawal privileged(Admin) user, valid account holdername, valid account number, valid account number, Account balance less than 0 |
| withdrawal04 | Test Withdrawal privileged(Admin) user, valid account number, Account balance greater than 0 |
| withdrawal05 | Test Withdrawal standard user, Invalid account number, |
| withdrawal06 | Test Withdrawal standard user, valid account number, Account balance less than 0 |
|  | withdrawal07 | Test Withdrawal standard user, valid account number, Account balance greater than 0 |
| Transfer | transfer01 | Test transferring funds between valid accounts. |
| transfer02 | Ensure transferring to the same account is not allowed. |
| transfer03 | Test transferring funds between accounts with insufficient balance in the source account. |
| transfer04 | Test transferring funds to a non-existent account. |
| transfer05 | Test transferring a negative amount (invalid input). |
| transfer06 | Test transferring funds with zero amount (invalid input). |
| transfer07 | Test transfer exceeding the $1000.00 limit in a standard session. |
| transfer08 | Ensure that the transfer from an account the user does not own is rejected. |
| transfer09 | Ensure admin can transfer between any accounts. |
| transfer10 | Test transferring funds with invalid characters in the amount. |
| transfer11 | Test with missing input fields. |
| transfer12 | Ensure transfers to/from inactive accounts are disabled. |
| Paybill | paybill01 | Test paying a bill from a valid account with sufficient funds. |
| paybill02 | Test paying a bill from an account with insufficient funds. |
| paybill03 | Test paying a bill to a non-existent biller. |
| paybill04 | Test paying a bill with admin override for any account. |
| paybill05 | Test payment exceeding $2000.00 in a standard session. |
| paybill06 | Test paying a bill with zero amount (invalid input). |
| paybill07 | Test paying a bill when the account is inactive. |
| paybill08 | Test paying a bill with missing or incomplete input. |
| paybill09 | Test paying a bill with invalid characters in the amount. |
| Deposit | deposit01 | Tests functionality of depositing money into valid(existing) accounts |
| deposit02 | Verifies that the system doesn’t allow the deposit of funds into non-existing accounts |
| deposit03 | Tests that only admin is allowed to deposit into other accounts |
| deposit04 | Verifies that deposit amount doesn’t exceed account balance limit |
| deposit05 | Verifies the functionality that the system doesn’t allow negative deposit input |
| deposit06 | Verifies the functionality that the system doesn’t allow invalid inputs for deposit amount (eg, character, alphabets) |
| deposit07 | Ensures account holder name is required when depositing money |
| deposit08 | Verifies the functionality that the deposit amount in unavailable to use during the current session |
| Create | create01 | Tests that admin is able to create new accounts |
| create02 | Verifies that only admin is able to create a new account |
| create03 | Checks the functionality that initial balance is not blank |
| create04 | Verifies the system rejects invalid input for the initial balance |
| create05 | Verifies that system throws an error if account exceeds maximum balance limit |
| create06 | Verifies  the functionality that doesn’t allow the account holder name to be more that 20 characters |
| create07 | Checks the functionality that account holder name is not blank |
| create08 | Verifies that a newly created account cannot be used for transactions within the same session. |
| Delete | delete01 | Tests that admin is able to delete accounts |
| delete02 | Verifies that only admin is able to delete an account |
| delete03 | Test the functionality that system throws an error if account number provided doesn’t exist |
| delete04 | Tests if the system verifies the account number matches the account holder's name |
| delete05 | Checks if account number is not blank |
| delete06 | Checks if account holder name is not blank |
| delete07 | Checks the functionality where the system prevents deleting an already-deleted account. |
| delete08 | Confirms no transactions are allowed on an account after it is deleted |
| ChangePlan | changeplan01 | Change from SP → NP for a valid active account. |
| changeplan02 | Attempt to change plan for a disabled account. |
| changeplan03 | Attempt to change plan without admin login. |
| changeplan04 | Attempt to change plan with an invalid account number. |
| changeplan05 | Attempt to change plan for a non-existent account holder name. |
| changeplan06 | Change from SP → NP and then back to SP in the same session |
| changeplan07 | Change from NP → SP for a valid active account |
| Disable | disable01 | Disable an active account (valid name and account number). |
| disable02 | Attempt to disable an already disabled account. |
| disable03 | Attempt to disable without admin login. |
| disable04 | Attempt to disable using an invalid account number. |
| disable05 | Attempt to disable using a non-existent account holder name. |
| disable06 | Verify no transactions are accepted on a disabled account post-disable. |
| disable07 | Attempt to disable when already logged in as admin for another task. |
| logout | logout01 | Successful logout after valid login and transactions as an admin. |
| logout02 | Attempt to logout without logging in. |
| logout03 | Attempt to logout immediately after logging in (no transactions performed). |
| logout04 | Verify session termination (no transactions accepted post-logout). |
| logout05 | Ensure Standard User Can Logout |
| logout06 | Attempt to logout multiple times in the same session. |
| logout07 | Successful logout after multiple valid transactions as admin |
| logout08 | Successful logout after multiple valid transactions as standard user |
| logout09 | Attempt to logout multiple times in the same session as a standard user |
| logout10 | Ensure session termination (no transactions accepted post-logout) as a standard user |

## 

## 

## 

## 

## 

## 1. Test Input and Output Organization

The test cases are well-organized in a structured folder system to ensure clarity and ease of execution. The organization follows a hierarchical structure, separating inputs, expected outputs, and transaction logs.

### Folder Structure (Generated using tree from Homebrew)

(current\_accounts\_file.txt) (Shared input file for all test cases)

inputs

│ ├── 00\_login\_inputs

│ ├── 01\_withdrawal\_inputs

│ ├── 02\_transfer\_inputs

│ ├── 03\_paybill\_inputs

│ ├── 04\_deposit\_inputs

│ ├── 05\_create\_inputs

│ ├── 06\_delete\_inputs

│ ├── 07\_changeplan\_inputs

│ ├── 08\_disable\_inputs

│ ├── 09\_logout\_inputs

outputs

│ ├── 00\_login\_outputs

│ ├── 01\_withdrawal\_outputs

│ ├── 02\_transfer\_outputs

│ ├── 03\_paybill\_outputs

│ ├── 04\_deposit\_outputs

│ ├── 05\_create\_outputs

│ ├── 06\_delete\_outputs

│ ├── 07\_changeplan\_outputs

│ ├── 08\_disable\_outputs

│ ├── 09\_logout\_outputs

transaction\_outputs

│ ├── 01\_withdrawal\_transaction\_outputs

│ ├── 02\_transfer\_transaction\_outputs

│ ├── 03\_paybill\_transaction\_outputs

│ ├── 04\_deposit\_transaction\_outputs

│ ├── 05\_create\_transaction\_outputs

│ ├── 06\_delete\_transaction\_outputs

│ ├── 07\_changeplan\_transaction\_outputs

│ ├── 08\_disable\_transaction\_outputs

│ ├── 09\_logout\_transaction\_outputs

### 

### 

### Organization Approach

1. Shared current\_accounts\_file.txt  
   This file remains constant for all test cases unless explicitly modified for specific tests.
2. Test Inputs (inputs/)
   * Each category of transactions is stored in its own subfolder.
   * Test files are named consistently, e.g., changeplan01.inp, withdrawal05.inp, etc.
3. Expected Outputs (outputs/)
   * Each test case has a corresponding expected output .out file stored in its respective transaction folder.
4. Transaction Outputs (transaction\_outputs/)
   * Each test case has an expected transaction log file .etf stored in the transaction folder.

## 2. Test Run Plan

The following automated test execution strategy ensures smooth running of test cases.

### Execution Steps

1. Prepare the Testing Environment
   * Ensure the current\_accounts\_file.txt is available and correct.
   * Ensure the program can read inputs from inputs/.
   * Clear previous output files (if necessary).
2. Batch Execution of Tests
   * Run all test cases sequentially using a test runner script:
     + ./run\_tests.sh
   * This script:
     + Loops through all .inp files in inputs/
     + Runs them through the banking system program
     + Stores outputs in the outputs/ and transaction\_outputs/ directories.
3. Validation of Test Results
   * Compare actual output files against expected output files.
   * Use a diff tool to highlight mismatches:
     + diff outputs/expected\_output.out outputs/actual\_output.out
   * If differences are found, flag the test as failed.
4. Logging Results
   * Each test case will be marked as:
     + Test Passed: withdrawal\_01
     + Test Failed: deposit\_03 (Output mismatch)
   * Store logs in test\_results.log.
5. Re-run Failed Tests (if necessary)
   * Identify failed tests and run them individually.