COMP 354
Test Document for the project myMoney

Team PA-PK

April 8, 2018

Table 1: Team

Name	ID Number
Anne-Laure Ehresmann	27858906
Marc-Antoine Dube	40029307
Kadeem Caines	26343600
Abdel Rahman Jawhar	27192142
Keith Dion	40036340
Hrachya Hakobyan	40041555
Andrew-Smith	40034936
Dongyu Chen	27241909
Yauheni Karaniuk	40005680
Renny Xu	40005262
Wei Wang	40041116

Table 2: Revision history

Version	Date	Changes
1.0	15 March 2018	Completed test document

Contents

List of Figures

List of Tables

1 Introduction

The aim of this document is to ensure that a coherent and accurate testing strategy is used by each member of the testing team. It seeks to test the implementation of the system described in the Design Plan, testing its validity, robustness, and reliableness as a software, as well as ensuring that the requirements in the Requirements Specification are met. It seeks to do this in a rigorous and justified manner. This document contains an overarching test plan, which seeks to outline each test subsystem, its strategy with regards to testing the associated requirements, and its execution strategy. This document then contains, for each subsystem, a detailed explanation of the set of tests included, and a test case for each individual test. Put together, the test subsystems group into a entire system test.

2 Test Plan

The system test plan has been split into five subsystems:

- **Functional Testing:** This test subsystem seeks to certify the functionality of the software against the use cases in the Requirements Specification. This category will use black-box testing as its strategy, verifying the usability given different inputs and regardless of the implementation of the sfotware. In its execution, a developer running such a test will typically first identify how the software should perform. Then, he or she verifies the functionality and reliability of the software given valid user behaviour, and then checks for robustness given invalid user behaviour.

- **Structural Testing:** This test subsystem seeks to verify the structure and code logic of the software. We ensure here that each part of the code functions as expected given both valid and invalid input, and test the behaviour of the system in unexpected states. This will let us confirm the valid flow of our code, and ensure logic faults are caught. For the execution of the test, we will use JUnit to create individual tests for each case. Each test will have an initial setup phase, a test phase, and a teardown phase, to ensure independence of state between each test. A test will also use Mockito, a mocking library, to ensure that the failure of some other, unrelated component of the code does not affect the performance of the tested component in each test.

- **Performance Testing:** This test subsystem seeks to measure the behaviour of the software in extreme states, when under particular workloads or dealing with extremely large datasets. It is useful for testing a number of our non-functional requirements, notably reliability, scalability, and, obviously, performance. In its execution, The tests measure performance statistics given a normal or 'control' environment, then compare it to the performance statistic given a particular dataset or workload.

- **Acceptance Testing:** This test subsystem seeks to meet the requirements of the use cases in the requirements set in the Requirements Specification. This is also a black-box testing category, as in functional testing, but unlike the aforementioned, we are instead performing a validation of the system: is our system actually what the user needs? In its execution, the system is given to a user, who will assert whether his needs are met and correspond to what how he or she expects the software to function.

- **Installation Testing:** This test subsystem seeks to verify that the installation process is both successful and easy in the platforms to be supported. This means ensuring that the choices taken by the user are respected (location of installation, installation just for one user or for whole computer...), verify that all dependent files and libraries are successfully linked and loaded, and valid configurations and connectivity to the database. The execution of this category is simply an activity wherein the installation process is attempted in a particular environment, testing all decisions and options available in the installation.

3 Functional Testing

Create User Account

Test Case	First name, last name, username and password are mandatory
Description	The user cannot sign up without providing a valid first name, last name, a username and a password
Input/Steps	1. Go to 'Sign Up' 2. Leave all the input fields blank 3. Click 'Sign up'
Output/Results	• Sign up fails, the account is not created • An error window displays all the errors

Test Case	The username must be unique
Description	The user cannot sign up with an already existing username
Input/Steps	1. Successfully sign 2. Log out 3. Go to 'Sign up' 4. Fill in all the input fields 5. Set the username field to be the username of the user created in the first ste 6. Click 'Sign Up'
Output/Results	• Sign up failed, the account is not created • An error window notifies that the username already exists

Test Case	The password must be valid
Description	The user cannot sign up with a password not matching the required format, as specified in the business rules

Input/Steps	1. Go to ’Sign up’ 2. Fill in all the input fields 3. Set the password to an alpha-numeric sequence of length less than 4 4. Set the repeat password field to match the password field 5. Click ’Sign Up’
Output/Results	• Sign up failed, the account is not created • An error window notifies that the password is not valid

Test Case	The user account is successfully created
Description	The user must be able to successfully create an account provided that all input information is valid
Input/Steps	1. Go to ’Sign up’ 2. Fill in all the input fields with valid data 3. Click ’Sign Up’ 4. Moved to the login page: input the username and the passowrd 5. Click ’Login’
Output/Results	• Sign up successful, the account is created • The user is logged-in to the newly created account

Delete User Account

Test Case	Password required
Description	The program asks for the user’s password before to delete the account
Input/Steps	1. Go to ’Update User Account’ 2. Click ’Delete user’
Output/Results	• A an input window appears asking for the user password

Test Case	The password must be valid
Description	The user cannot delete the account if the password is invalid
Input/Steps	1. Go to ’Update User Account’ 2. Click ’Delete user’ 3. Enter a wrong password
Output/Results	• The account is not deleted • An error window must appear notifying the user that the password was invali

Test Case	The account is successfully deleted
Description	The user account is successfully deleted if the password is correct
Input/Steps	1. Go to ’Update User Account’ 2. Click ’Delete user’ 3. Enter the correct password

Output/Results	• The account is not deleted • An error window must appear notifying the user that the password was invali

Test Case	The user account is successfully created
Description	The user must be able to successfully create an account provided that all input information is valid
Input/Steps	1. Go to 'Sign up' 2. Fill in all the input fields with valid data 3. Click 'Sign Up' 4. Moved to the login page: input the username and the passowrd 5. Click 'Login'
Output/Results	• Sign up successful, the account is created • The user is logged-in to the newly created account

Add Bank Account to a User Account

Test Case	Add a valid bank account to a user
Description	A valid bank account should be added to the user
Input/Steps	1. Go to the main screen 2. Input an account ID in the 'Enter Account ID' field 3. Click the 'Add' button
Output/Results	• A row should be added in the table of account

Test Case	Add a same account to a user
Description	An account should not be added to the same user twice
Input/Steps	1. Go to the main screen 2. Input an account ID in the 'Enter Account ID' field 3. Click the 'Add' button
Output/Results	• Failure, the account is not added • An error window displays all the errors

Test Case	Add an account used by another user
Description	An account used by another user should not be added to another user
Input/Steps	1. Go to the main screen 2. Input an account ID in the 'Enter Account ID' field 3. Click the 'Add' button
Output/Results	• Failure, the account is not added • An error window displays all the errors

Remove Bank Account from a User Account

Test Case	Remove a bank account from a user
Description	An existing bank account should be removed
Input/Steps	1. Go to the main screen 2. Select a line on the accounts table 3. Click the 'Remove Selected' button
Output/Results	• The row should be removed in the table of account

Test Case	Remove no bank account from a user
Description	If no account is selected, no accounts should be removed
Input/Steps	1. Go to the main screen 2. Click the 'Remove Selected' button
Output/Results	• Nothing happens because no account was selected

View Transactions for Specific Bank Account

Test Case	Selection of a bank account is mandatory
Description	The user select an exsiting bank account.
Input/Steps	1. Click a bank account from bank account list 2. Click 'View All Transactions'
Output/Results	• Transaction list is displayed for selected bank account • Empty list is shown if there is no transactions

View All Transactions from all Bank Accounts

Test Case	Display all transactions
Description	Empty selection in bank account list return all transactions of exsiting bank accounts
Input/Steps	1. Click the button 'View All Transactions'
Output/Results	• All transactions in bank account list are shown.

Update User Account

Test Case	First name is mendatory
Description	First name is required in user profile
Input/Steps	1. Click the textfield 'First anme' 2. Input first name 3. Click the button 'Save changes'

Output/Results	• New first name is saved if it is not empty • Error message is shown if the text field is empty.

Test Case	Last anme is mendatory
Description	Last name is required in user profile
Input/Steps	1. Click the textfield 'Last anme' 2. Input last name 3. Click the button 'Save changes'
Output/Results	• New last name is saved if it is not empty • Error message is shown if the text field is empty.

Test Case	First name is valid
Description	Validating first name
Input/Steps	1. Click the textfield 'First anme' 2. Input first name 3. Click the button 'Save changes'
Output/Results	• New first name is saved if it is not empty • Error message is shown if the input does not pass the validation.

Test Case	Last name is valid
Description	Validating last name
Input/Steps	1. Click the textfield 'Last anme' 2. Input last name 3. Click the button 'Save changes'
Output/Results	• New last name is saved if it is not empty • Error message is shown if the input does not pass the validation.

Test Case	Password input validation
Description	Validate password input
Input/Steps	1. Click the textfield 'Update Password' or 'Confirm New Password' 2. Input new password
Output/Results	• New password is accepted if the password is valid. • Error message is shown if the password is not valid.

Test Case	Two password input matches
Description	Input of two passwords should match

Input/Steps	1. Click the textfield 'Update Password' 2. Input new password 3. Click the textfield 'Confirm New Password' 4. Input new password the second time 5. Click the button 'Save changes'
Output/Results	• New password is saved if two input matches • Error message is shown if two input does not match.

Test Case	Email address is valid
Description	Validating email address
Input/Steps	1. Click the textfield 'Email' 2. Input email address 3. Click the button 'Save changes'
Output/Results	• A valid email address is saved • Error message is shown if the input email address is not valid.

Test Case	Phone number can be saved
Description	A phone number can be saved to profile
Input/Steps	1. Click the textfield 'Phone Number' 2. Input a phone number 3. Click the button 'Save changes'
Output/Results	• The phone number can be saved to user profile • Error message is shown if saving failed.

Test Case	Current address can be saved
Description	An address can be saved to profile
Input/Steps	1. Click the textfield 'Current address' 2. Input an address 3. Click the button 'Save changes'
Output/Results	• The address can be saved to user profile • Error message is shown if saving failed.

Test Case	Delete user account
Description	User profile is deleted
Input/Steps	1. Click the button 'Delete User'
Output/Results	• User profile is removed from the database • Error message is shown if deletion fails

Sort transactions by any attribute

Test Case	Sort by Date
Description	The user wants to see the transactions sorted by date.
Input/Steps	1. click on 'View All Transactions' button or double click on a specific bank acc 2. click on the attribute 'Date' one time to sort in ascending order or two times
Output/Results	• The transactions list is sorted in ascending or descending date order

Test Case	Sort by Amount
Description	The user wants to see the transactions sorted by amount.
Input/Steps	1. click on 'View All Transactions' button or double click on a specific bank acc 2. click on the attribute 'Amount' one time to sort in ascending order or two ti
Output/Results	• The transactions list is sorted in ascending or descending amount order

Test Case	Sort by Type
Description	The user wants to see the transactions sorted by type.
Input/Steps	1. click on 'View All Transactions' button or double click on a specific bank acc 2. click on the attribute 'Type' one time to sort in ascending order or two times
Output/Results	• The transactions list is sorted in types of transactions

Test Case	Sort by Category
Description	The user wants to see the transactions sorted by categories.
Input/Steps	1. click on 'View All Transactions' button or double click on a specific bank acc 2. click on the attribute 'Category' one time to sort in ascending order or two t
Output/Results	• The transactions list is sorted in categories.

Categorize transaction

Test Case	Categorize from predefined list
Description	The user wants to set the category of the transaction from the predefined categories.
Input/Steps	1. click on 'View All Transactions' button or double click on a specific bank acc 2. select the desired transaction to be categorized. 3. press on the category option and choose the appropriate category from the d
Output/Results	• The transaction's category is set to the one chosen by the user.

Test Case	Create a new category
Description	The user wants to create a category for the transaction.
Input/Steps	1. click on 'View All Transactions' button or double click on a specific bank acc 2. select the desired transaction to be categorized. 3. press on the category option and type in the new category.

Output/Results	• The transaction's category is set to the one created by the user.

Test Case	Category created is too long.
Description	The user wants to create a category for the transaction.
Input/Steps	1. click on 'View All Transactions' button or double click on a specific bank acc 2. select the desired transaction to be categorized. 3. press on the category option and type in the new category which is longer th
Output/Results	• The transaction's category is not set because the category entered is too long.

Filter transactions by date range

Test Case	Filter transactions with valid date range
Description	The user wants to see the transactions within a valid date range.
Input/Steps	1. click on 'View All Transactions' button or double click on a specific bank acc 2. select a start date which is before at least one transaction. 3. select an end date or keep it blank if the desired date is the current date.
Output/Results	• The displayed transactions are withing the date range selected.

Test Case	Select an invalid date range
Description	The user sets an invalid date range.
Input/Steps	1. click on 'View All Transactions' button or double click on a specific bank acc 2. select a start date which is after any transaction.
Output/Results	• The list of transactions is empty.

Search transaction by existing category

Test Case	Filter transactions by category in all transactions view
Description	The All Transactions view should let filter by categories
Input/Steps	1. Click the button 'View All Transactions' 2. Input a category in the 'category' field
Output/Results	• Only the categories starting with what was inputted should be displayed

Test Case	Filter transactions by category in detailed account view
Description	The accounts details view should let filter by categories
Input/Steps	1. Select an account by double clicking on a row 2. Input a category in the 'category' field
Output/Results	• Only the categories starting with what was inputted should be displayed

Test Case	An empty search should return all transactions

Description	When the category field is empty, all transactions should be shown
Input/Steps	1. Click the button 'View All Transactions' 2. Input a category in the 'category' field 3. Clear the category field
Output/Results	• All transactions from before the filtering should be shown

Test Case	A search with a non-existing category should yield no result
Description	If no categories match the category filter, no transactions should be shown
Input/Steps	1. Click the button 'View All Transactions' 2. Input a category that does not exist in the 'category' field
Output/Results	• No transactions should be shown

Generate transaction statement by exporting to CSV

Test Case	Generate statement from All Transactions view
Description	The All Transactions view should let generate a statement
Input/Steps	1. Click the button 'View All Transactions' 2. Click the button 'Generate Excel' 3. Select the location of the generated file
Output/Results	• A file named 'all-transactions$TIMESTAMP.csv'shouldbegeneratedinthesele$

Test Case	Generate statement from Account Details view
Description	The Account Details view should let generate a statement
Input/Steps	1. Select an account by double clicking on a row 2. Click the button 'Generate Excel' 3. Select the location of the generated file
Output/Results	• A file named 'transactions$TIMESTAMP.csv'shouldbegeneratedintheselecte$

Test Case	Generate empty statement
Description	An account with no transactions should still generate a statement
Input/Steps	1. Click the button 'View All Transactions' 2. Click the button 'Generate Excel' 3. Select the location of the generated file
Output/Results	• A file named 'all-transactions$TIMESTAMP.csv'shouldbegeneratedinthesele$

Send statement by email

Test Case	Send statement by email from All Transactions view

Description	A statement should be sent by email from the All Transactions view
Input/Steps	1. Click the button 'View All Transactions' 2. Click the button 'Email CSV'
Output/Results	• An email containing the transactions in your inbox

Test Case	Send statement by email from Account Details view
Description	A statement should be sent by email from the Account Details view
Input/Steps	1. Select an account by double clicking on a row 2. Click the button 'Email CSV'
Output/Results	• An email containing the transactions in your inbox

Test Case	Send statement by email when no email is configured
Description	A statement can't be sent when no email is configured
Input/Steps	1. Click the button 'Update User Account' 2. Remove the email 3. Click the button 'Save Changes' 4. Click the button 'View All Transactions' 5. Click the button 'Email CSV'
Output/Results	• An error window notifies the user that his email is not configured

4 Structural Testing

4.1 Unit Test cases

AccountService.addAccount(request, user)

Table 47: addAccount(request, user)

Tester Name	Hrachya	
Test Date	2/7/18	
Class Name	com.github.comp354project.model.account.AccountService	
Method Name	addAccount(request, user)	
Purpose	This test suite tests the functionality of adding a new bank account	
Use Cases	03	
Test Scenarios		
testAddAccount_withInvalidParameters_shouldThrow		
Input Specification	request	accountOwner
	null	null
Expected Output	ValidationException is thrown The number of ValidationErrors is equal to 2	

Actual Output	ValidationException is thrown The number of ValidationErrors is equal to 2	
Bug Found	false	
Purpose	Adding an account with invalid request or user should fail	
testAddAccount_withNonexistentRemoteAccount_shouldThrow		
Input Specification	request	accountOwner
	ID: 1	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111
Expected Output	AccountDoestNotExistException is thrown	
Actual Output	AccountDoestNotExistException is thrown	
Bug Found	false	
Purpose	A request for adding a nonexistent account should fail	
testAddAccount_withInvalidUser_shouldThrow		
Input Specification	request	accountOwner
	ID: 1	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111
Expected Output	ValidationException is thrown	
Actual Output	ValidationException is thrown	
Bug Found	false	
Purpose	Adding an account with an invalid owner should throw	
testAddAccount_withExistingAccount_shouldThrow		
Input Specification	request	accountOwner
	ID: 1	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111

Expected Output	AccountExistsException is thrown		
Actual Output	AccountExistsException is thrown		
Bug Found	false		
Purpose	Adding an already existing account should throw		
testAddAccount_withValidAccount_shouldReturnValidAccount			
Input Specification	request	accountOwner	expectedAccount
	ID: 1	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111	ID: 1 user: accountOwner bankName: TD type: Checking balance: 15823.12
Expected Output	The account is fetched and persisted in the database The persisted account is equal to the 'expectedAccount' object The returned account is equal to the 'expectedAccount' object		
Actual Output	The account is fetched and persisted in the database The persisted account is equal to the 'expectedAccount' object The returned account is equal to the 'expectedAccount' object		
Bug Found	false		
Purpose	Adding a valid account with a valid owner must succeed		

AccountService.deleteAccount(account)

Table 48: deleteAccount(account)

Tester Name	Anne-Laure
Test Date	3/5/18
Class Name	com.github.comp354project.model.account.AccountService
Method Name	deleteAccount(account)
Purpose	This test suite tests the functionality of removing a user's bank account
Use Cases	04
Test Scenarios	
testDeleteAccount_withNullAccount_shouldThrow	
Input Specification	account
	null
Expected Output	ValidationException is thrown
Actual Output	ValidationException is thrown
Bug Found	false
Purpose	Delete a null account should fail
testDeleteAccount_withAccountWithNullID_shouldThrow	

Input Specification	account
	ID: null
	user: null
	bankName:
	type:
	balance: 0
Expected Output	ValidationException is thrown
Actual Output	ValidationException is thrown
Bug Found	false
Purpose	Deleting an account with null ID should fail
testDeleteAccount_withNonExistentAccount_shouldThrow	
Input Specification	account
	ID: 1
	user: accountOwner
	bankName: TD
	type: Checking
	balance: 15823.12
Expected Output	ValidationException is thrown
Actual Output	ValidationException is thrown
Bug Found	false
Purpose	Deleting an nonexistent account should fail
testDeleteAccount_withValidAccount_shouldSucceed	
Input Specification	account
	ID: 1
	user: accountOwner
	bankName: TD
	type: Checking
	balance: 15823.12
Expected Output	The account is deleted from the database
Actual Output	The account is deleted from the database
Bug Found	false
Purpose	Deleting an existing account should succeed
testDeleteAccount_withValidAccount_shouldDeleteAllAssociatedTransactionsAndAccount	
Input Specification	account
	ID: 1
	user: accountOwner
	bankName: TD
	type: Checking
	balance: 15823.12
	transactions: [object Object]
Expected Output	The account is deleted from the database
	All the associated transactions are deleted from the database

Actual Output	The account is deleted from the database All the associated transactions are deleted from the database
Bug Found	false
Purpose	Deleting an existing account should delete all associated transactions

AccountService.deleteAccountsForUser(user)

Table 49: deleteAccountsForUser(user)

Tester Name	Hrachya
Test Date	4/2/18
Class Name	com.github.comp354project.model.account.AccountService
Method Name	deleteAccountsForUser(user)
Purpose	This test suite tests the functionality of removing a user's bank accounts and associated transactions
Use Cases	04
Test Scenarios	
testDeleteAccountsForUser_withNullUserID_shouldThrow	
Input Specification	userID null
Expected Output	ValidationException is thrown
Actual Output	ValidationException is thrown
Bug Found	false
Purpose	Deleting accounts with null user ID should fail
testDeleteAccountsForUser_withNonexistentUser_shouldSucceed	
Input Specification	userID 1
Expected Output	No accounts are deleted. The system state is not changed.
Actual Output	No accounts are deleted. The system state is not changed.
Bug Found	false
Purpose	Deleting a nonexistent user's accounts should succeed and should not inflict any changes to the system.
testDeleteAccountsForUser_withValidUserAndEmptyAccounts_shouldSucceed	
Input Specification	user ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111

Expected Output	No accounts are deleted. The system state is not changed.	
Actual Output	No accounts are deleted. The system state is not changed.	
Bug Found	false	
Purpose	Deleting the accounts of a user who does not have any accounts should succeed and inflict no changes to the system	
testDeleteAccountsForUser_withAssociatedTransactions_shouldDeleteAccountAndTransactions		
Input Specification	user	account
	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111	ID: 1 user: user bankName: TD type: Checking balance: 15823.12 transactions: [object Object]
Expected Output	The accounts are deleted from the database All the associated transactions are deleted from the database	
Actual Output	The accounts are deleted from the database All the associated transactions are deleted from the database	
Bug Found	false	
Purpose	Deleting the accounts of the user should also delete all the associated transactions.	

AuthenticationService.authenticate(username, password)

Table 50: authenticate(username, password)

Tester Name	Hrachya	
Test Date	2/3/18	
Class Name	com.github.comp354project.model.auth.AuthenticationService	
Method Name	authenticate(username, password)	
Purpose	This test suite tests the authentication of the user	
Use Cases	01	
Test Scenarios		
testAuthenticate_withInvalidUsernameOrPassword_shouldThrow		
Input Specification	username	password
	null	null
Expected Output	ValidationException is thrown The number of ValidationErrors is equal to 2	
Actual Output	ValidationException is thrown The number of ValidationErrors is equal to 2	
Bug Found	false	

Purpose	A user with invalid credentials should not be able to authenticate		
testAuthenticate_withNonexistentUsername_shouldThrow			
Input Specification	username	password	
	username	password	
Expected Output	ValidationException is thrown		
Actual Output	ValidationException is thrown		
Bug Found	false		
Purpose	A user with a nonexistent username should not be able to authenticate		
testAuthenticate_withIncorrectPassword_shouldThrow			
Input Specification	testUser	username	password
	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111	admin	INCORRECT_PASSWORD
Expected Output	ValidationException is thrown		
Actual Output	UserLoggedInException is thrown		
Bug Found	false		
Purpose	Authentication with a valid username but an incorrect password should fail		
testAuthenticate_withCorrectCredentials_shouldReturnUser			
Input Specification	testUser	username	password
	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111	admin	admin
Expected Output	The authentication is successful and the authenticated user is returned The authenticated user is equal to the 'testUser' object		
Actual Output	The authentication is successful and the authenticated user is returned The authenticated user is equal to the 'testUser' object		
Bug Found	false		
Purpose	Authentication with a valid username but an incorrect password should fail		

RemoteAccountService.getAccount(GetRemoteAccountRequest)

Table 51: getAccount(GetRemoteAccountRequest)

Tester Name	Abed Jawhar		
Test Date	3/13/18		
Class Name	com.github.comp354project.model.account.remote.RemoteAccountService		
Method Name	getAccount(GetRemoteAccountRequest)		
Purpose	This test suite tests fetching an account in the 'API' that connects to other systems		
Use Cases	03		
Test Scenarios			
testGetAccount_withNullRequest_shouldThrow			
Input Specification	request		
	null		
Expected Output	ValidationException is thrown		
Actual Output	ValidationException is thrown		
Bug Found	false		
Purpose	A null account can't be fetched		
testGetAccount_withInvalidRequest_shouldThrow			
Input Specification	request		
Expected Output	ValidationException is thrown		
Actual Output	ValidationException is thrown		
Bug Found	false		
Purpose	An empty account can't be fetched		
testGetAccount_withExistingAccount_shouldReturnValidAccount			
Input Specification	expectedAccount	expectedAccountTransactions	request
	ID: 1 bankName: TD type: Checking balance: 15823.12	ID: 1 account: testRem date: 1517091082 amount: 52.2 type: Transfer sourceID: null destinationID: 2	accountID: 1
Expected Output	The fetched account should be the same as the 'expectedAccount' The number of transactions fetched should be 1		
Actual Output	The fetched account should be the same as the 'expectedAccount' The number of transactions fetched should be 1		
Bug Found	false		
Purpose	A valid account should be fetched		

SessionManager.login(username, password)

Table 52: login(username, password)

Tester Name	Hrachya				
Test Date	2/7/18				
Class Name	com.github.comp354project.model.auth.SessionManager				
Method Name	login(username, password)				
Purpose	This test suite tests the login of a user				
Use Cases	02				
Test Scenarios					
testLogin_withInvalidCredentials_shouldThrow					
Input Specification	username	password			
Expected Output	ValidationException is thrown				
Actual Output	ValidationException is thrown				
Bug Found	false				
Purpose	A user with invalid credentials should not be able to login				
testLogin_withValidCredentials_shouldReturnUser					
Input Specification	testUser	username	password	loggedIn	authenticateInvoc
	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111			true	1
Expected Output	The method authenticate is invoked 1 time The user is logged in The logged in user is equal to 'testUser' object				
Actual Output	The method authenticate is invoked 1 time The user is logged in The logged in user is equal to 'testUser' object				
Bug Found	false				
Purpose	A user with valid credentials should be able to login				
testLogin_withLoggedInUser_shouldThrow					
Input Specification	username	password			
Expected Output	UserLoggedInException is thrown				
Actual Output	UserLoggedInException is thrown				
Bug Found	false				
Purpose	A user that is already logged in should not be able to login again				

SessionManager.logout()

Table 53: logout()

Tester Name	Hrachya			
Test Date	2/7/18			
Class Name	com.github.comp354project.model.auth.SessionManager			
Method Name	logout()			
Purpose	This test suite tests the function to logout			
Use Cases	02			
Test Scenarios				
testLogin_withInvalidCredentials_shouldThrow				
Input Specification	username	password	isLoggedIn	currentUser
			false	null
Expected Output	After logout, the login status should be false After logout, the current user should be null			
Actual Output	After logout, the login status should be false After logout, the current user should be null			
Bug Found	false			
Purpose	A user should be completely logged out of the application			

TransactionService.updateTransactionCategory(transactionID, category)

Table 54: updateTransactionCategory(transactionID, category)

Tester Name	Hrachya	
Test Date	3/4/18	
Class Name	com.github.comp354project.model.account.TransactionService	
Method Name	updateTransactionCategory(transactionID, category)	
Purpose	This test suite tests the functionality of updating the category of a transaction	
Use Cases	08	
Test Scenarios		
testUpdateCategory_withNullTransactionID_shouldThrow		
Input Specification	transactionID	category
	null	Leisure
Expected Output	ValidationException is thrown	
Actual Output	ValidationException is thrown	
Bug Found	false	
Purpose	Updating a null transaction ID should fail	
testUpdateCategory_withNonexistentTransaction_shouldThrow		

Input Specification	transactionID	category
	111111	Leisure
Expected Output	ValidationException is thrown	
Actual Output	ValidationException is thrown	
Bug Found	false	
Purpose	Updating a nonexistent transaction should fail	
testUpdateCategory_withNullCategory_shouldSucceed		
Input Specification	transactionID	category
	10	null
Expected Output	The ’category’ of the transaction with the specified ID is set to null	
Actual Output	The ’category’ of the transaction with the specified ID is set to null	
Bug Found	false	
Purpose	Updating the category of a valid transaction to null must succeed	
testUpdateCategory_withEmptyCategory_shouldSucceed		
Input Specification	transactionID	category
	10	
Expected Output	The ’category’ of the transaction with the specified ID is set to ”	
Actual Output	The ’category’ of the transaction with the specified ID is set to ”	
Bug Found	false	
Purpose	Updating the category of a valid transaction to an empty string must succeed	
testUpdateCategory_withValidCategory_shouldSucceed		
Input Specification	transactionID	category
	10	Leisure
Expected Output	The ’category’ of the transaction with the specified ID is set to ’Leisure’	
Actual Output	The ’category’ of the transaction with the specified ID is set to ’Leisure’	
Bug Found	false	
Purpose	Updating the category of a valid transaction must succeed	
testUpdateCategory_withInvalidCategory_shouldThrow		
Input Specification	transactionID	category
	10	AAAAAAAAAAAAAAAAA
Expected Output	ValidationException is thrown	
Actual Output	ValidationException is thrown	
Bug Found	false	
Purpose	Updating the category of a valid transaction to a an invalid value as determined by the business rules must fail	

UserService.createUser(User)

Table 55: createUser(User)

Tester Name	Hrachya

Test Date	1/31/18	
Class Name	com.github.comp354project.model.user.UserService	
Method Name	createUser(User)	
Purpose	This test suite tests the creation of a user	
Use Cases	01	
Test Scenarios		
createUser_withNullUser_shouldThrow		
Input Specification	user	
	null	
Expected Output	ValidationException is thrown	
Actual Output	ValidationException is thrown	
Bug Found	false	
Purpose	No null user can be saved in the database	
testCreateUser_withInvalidUser_shouldThrow		
Input Specification	user	errors
		4
Expected Output	ValidationException is thrown 4 exceptions are thrown because missing fields: username, password, firstname,	
Actual Output	ValidationException is thrown 4 exceptions are thrown because missing fields: username, password, firstname,	
Bug Found	false	
Purpose	No empty value user can be saved in the database	
testCreateUser_withValidUser_shouldReturnUser		
Input Specification	user	
	username: USERNAME password: PASSWORD firstName: FIRSTNAME lastName: LASTNAME	
Expected Output	User ID was autogenerated upon save The saved user is the same as the inputted user	
Actual Output	User ID was autogenerated upon save The saved user is the same as the inputted user	
Bug Found	false	
Purpose	A valid user should be inserted in the database	
testCreateUser_withExistingUsername_shouldThrow		
Input Specification	user	
	username: USERNAME password: PASSWORD firstName: FIRSTNAME lastName: LASTNAME	
Expected Output	ValidationException is thrown	
Actual Output	ValidationException is thrown	

Bug Found	false
Purpose	A user cannot be created if the username is already taken

UserService.deleteBankAccount(Account)

Table 56: deleteBankAccount(Account)

Tester Name	Anne-Laure		
Test Date	3/7/18		
Class Name	com.github.comp354project.model.user.UserService		
Method Name	deleteBankAccount(Account)		
Purpose	This test suite tests the deletion of a bank account		
Use Cases	04		
Test Scenarios			
testDeleteBankAccount_withNullAccount_ShouldThrow			
Input Specification	account		
	null		
Expected Output	ValidationException is thrown		
Actual Output	ValidationException is thrown		
Bug Found	false		
Purpose	No null account can be passed to the function		
testDeleteBankAccount_withoutBeingLoggedIn_ShouldThrow			
Input Specification	account	testUser	
	ID: 1 user: testUser bankName: TD type: Checking balance: 15823.12	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111	
Expected Output	AuthenticationException is thrown		
Actual Output	AuthenticationException is thrown		
Bug Found	false		
Purpose	A user that is not authenticated cannot delete his accounts		
testDeleteBankAccount_withoutProperAuthorisation_ShouldThrow			
	testUser	user2	testAccount

Input Specification	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111	username: username password: password firstName: firstname lastName: lastname ID: 999	ID: 1 user: testUser bankName: TD type: Checking balance: 15823.12
Expected Output	AuthorisationException is thrown		
Actual Output	AuthorisationException is thrown		
Bug Found	false		
Purpose	A user cannot modify the accounts of another user		
testDeleteBankAccount_WithProperAuthorisation_ShouldSucceed			
Input Specification	testUser	testAccount	invocationCount
	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111	ID: 1 user: testUser bankName: TD type: Checking balance: 15823.12	1
Expected Output	Execution of the deletion of the account once		
Actual Output	Execution of the deletion of the account once		
Bug Found	false		
Purpose	An authenticated user should succeed in deleting his own bank accounts		

UserService.deleteUser(User)

Table 57: deleteUser(User)

Tester Name	Abed Jawhar
Test Date	3/13/18
Class Name	com.github.comp354project.model.user.UserService
Method Name	deleteUser(User)
Purpose	This test suite tests the deletion of a user
Use Cases	02
Test Scenarios	
testDeleteUser_withNullUser_shouldThrow	
Input Specification	user
	null
Expected Output	ValidationException is thrown

Actual Output	ValidationException is thrown			
Bug Found	false			
Purpose	A null user can't be deleted			
testDeleteUser_withNonexistantUser_shouldThrow				
Input Specification	testUser			
	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111			
Expected Output	ValidationException is thrown			
Actual Output	ValidationException is thrown			
Bug Found	false			
Purpose	A user that does not exist can't be deleted			
testDeleteUser_withExistingtUser_shouldSucceed				
Input Specification	testUser	returnSize		
	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111	0		
Expected Output	The number of users with ID 1 is 0			
Actual Output	The number of users with ID 1 is 0			
Bug Found	false			
Purpose	A valid user should be deleted			
testDeleteUser_withExistingtUser_shouldDeleteAssociatedAccounts				
Input Specification	testUser	testAccount	returnSize	deleteAccountInvoc
	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111	ID: 1 user: testUser bankName: TD type: Checking balance: 15823.12	0	1

Expected Output	The number of users with ID 1 is 0 Delete account should be invocated 1 time
Actual Output	The number of users with ID 1 is 0 Delete account should be invocated 1 time
Bug Found	false
Purpose	A valid user should be deleted and his accounts also

UserService.updateUser(User)

Table 58: updateUser(User)

Tester Name	Abed Jawhar			
Test Date	3/13/18			
Class Name	com.github.comp354project.model.user.UserService			
Method Name	updateUser(User)			
Purpose	This test suite tests the update of a user			
Use Cases	02			
Test Scenarios				
testUpdateUser_withNullUser_shouldThrow				
Input Specification	user			
	null			
Expected Output	ValidationException is thrown			
Actual Output	ValidationException is thrown			
Bug Found	false			
Purpose	A null user can't be updated			
testUpdateUser_withNonexistenttUser_shouldThrow				
Input Specification	testUser			
	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111			
Expected Output	ValidationException is thrown			
Actual Output	ValidationException is thrown			
Bug Found	false			
Purpose	A user that does not exist can't be updated			
testUpdateUser_withValidUser_shouldSucceed				
	testUser	firstName	lastName	password

Input Specification	ID: 1 firstName: Hrachya lastName: Hakobyan username: admin password: admin email: sample@email.com address: address phone: 111111	Abed	jawhar	admin2
Expected Output	The firstName is updated to 'Abed' The lastName is updated to 'jawhar' The password is updated to 'admin2'			
Actual Output	The firstName is updated to 'Abed' The lastName is updated to 'jawhar' The password is updated to 'admin2'			
Bug Found	false			
Purpose	A valid user should be updated			

5 Performance Testing

Tester Name	Anne-Laure
Test Date	5/4/18
Purpose	This test suite contains the series of tests performed with yourKit Java Profiler.
System specification	
OS	GNU/Linux Fedora 27 x64, version 4.13.9-300
RAM	4GB
Graphics Card	Intel Celeron 3205U @ 1.50GHz x 2
OpenJDK version	1.8.0_144
Profiler	YourKit Java Profiler 2017.02-b75

control stress test: local database with 5 accounts and 5 transactions	
CPU usage chart	CPU usage chart 1
Thread Count chart	Thread Count chart 1

Events	44m 38s: application launched 44m 40s: login menu loaded 44m 45s: logged in 44m 47s: sorted accounts by type 44m 53s: removed bank account 2 44m 58s: added back account 2 45m 02s: viewed all transactions 45m 15s: shut down application

Memory					
Heap-Memory			**Non-Heap Memory**		
Used	**Allocated**	**Limit**	**Used**	**Allocated**	**Limit**
73MB	140MB	910MB	65 MB	65 MB	65 MB

CPU				
Classes	**Threads**			
	Currently live	**Currently live daemons**	**Peak**	**Total created**
8,415	11	5	31	53

stress test: local database with 11,000 accounts and 5 transactions	
CPU usage chart	CPU usage chart 2
Thread Count chart	Thread Count chart 2
Events	2m 43s: application launched 2m 46s: login menu loaded 2m 51s: logged in 2m 52s: sorted accounts by type 2m 53s: reversed sort of accounts by type 2m 54s: sorted accounts by ID 2m 55s: clicked on an account to view account details 2m 57s: returned to account list 3m 02s: viewed all transactions 3m 15s: shut down application

Memory					
Heap-Memory			**Non-Heap Memory**		
Used	**Allocated**	**Limit**	**Used**	**Allocated**	**Limit**
123MB	203MB	910MB	67 MB	67 MB	67 MB

CPU				
Classes	Threads			
	Currently live	Currently live daemons	Peak	Total created
8,454	14	6	31	48

stress test: local database with 11,000 accounts and 10,000 transactions	
CPU usage chart	CPU usage chart 2
Thread Count chart	Thread Count chart 2
Events	34m 11s: application launched 34m 14s: login menu loaded 34m 18s: logged in 34m 20s: sorted accounts by type 34m 25s: reversed sort of accounts by type 34m 27s: sorted accounts by ID 34m 29s: clicked on an account to view account details 34m 30s: returned to account list 34m 34s: viewed all transactions 34m 37s: returned to account list 35m 1s: shut down application

Memory					
Heap-Memory			Non-Heap Memory		
Used	Allocated	Limit	Used	Allocated	Limit
158MB	207MB	910MB	70 MB	70 MB	70 MB

CPU				
Classes	Threads			
	Currently live	Currently live daemons	Peak	Total created
8,456	14	6	31	88

6 Acceptance Testing

7 Installation Testing

7.1 Glossary

Table 63: Glossary

CPU	Central Processing Unit. Handles the instructions of a computer program
Memory	Volatile memory used by the Operating System and software
Heap-memory	Pool of memory from which a program can request memory
Thread	Small sequence of programmed instructions

8 References

A Description of Input Files

Describe/include test data from input files.

B Description of Output Files

Describe/include test expected output that are output files.