University of Regina | Go far, together.

ENSE 375 – Software Testing and Validation

VaultGuard Testing

Brydon Herauf (200454546)

Ansar Ahmed (200470692)

Gursharan Singh Rehal (200480626)

Table of Contents

1. Introduction

VaultGuard is a password manager tool accessible through a command line interface. To test this application, we used a combination of structure-based and specification-based testing. This included path testing, data flow testing, integration testing, boundary value analysis, equivalence class testing, decision tables, state transition testing, and use case testing. We also informally tested each core class by utilizing test-driven-development.

2. Test Requirements

To confirm that this application is functioning as intended, we must confirm that it satisfies these requirements:

- Users must be able to register for an account with valid credentials and login with them later.
- Users must be able to logout.
- Users must be able to exit the program.
- Users must be able to add new keys to their vault.
- Users must be able to edit the values of existing keys.
- Users must be able to delete existing keys.
- Users must be able to view a list of all of their keys.
- Users must be able to copy a key’s value to their clipboard.
- The system must validate inputs and reject any invalid requests.

3. Structure Based Testing

3.1. Path Testing

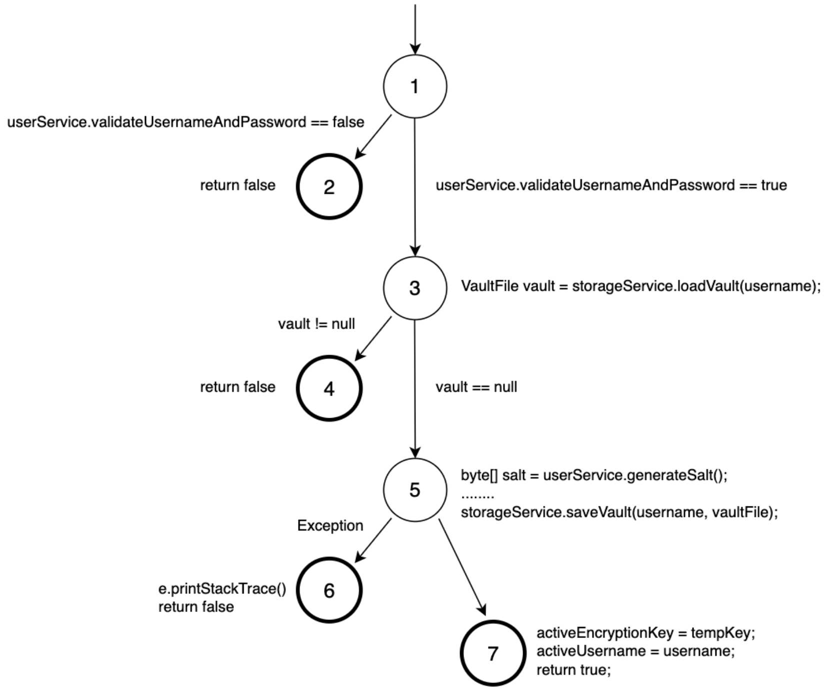
We performed path testing on the “register” method of “PasswordManager.java”.

Requirements Tested:

- Users must be able to register for an account with valid credentials and login with them later.

Specifications:

The control flow graph for “register” is:


1
userService.validateUsernameAndPassword == false
return false
2
userService.validateUsernameAndPassword == true
3
VaultFile vault = storageService.loadVault(username);
vault != null
return false
4
vault == null
5
byte[] salt = userService.generateSalt();
....
storageService.saveVault(username, vaultFile);
Exception
e.printStackTrace()
return false
6
7
activeEncryptionKey = tempKey;
activeUsername = username;
return true;

Performing prime path analysis yielded the following paths:

- [1, 3, 5, 6]
- [1, 3, 5, 7]
- [1, 3, 4]
- [1, 2]

The set of test paths that achieve prime path coverage are the same as those listed above.

Test Cases:

These four test paths are explicitly covered in the “Structure-based Testing” section of “PasswordManagerTest.java”.

Results:

All tests passed.

3.2. Data Flow Testing

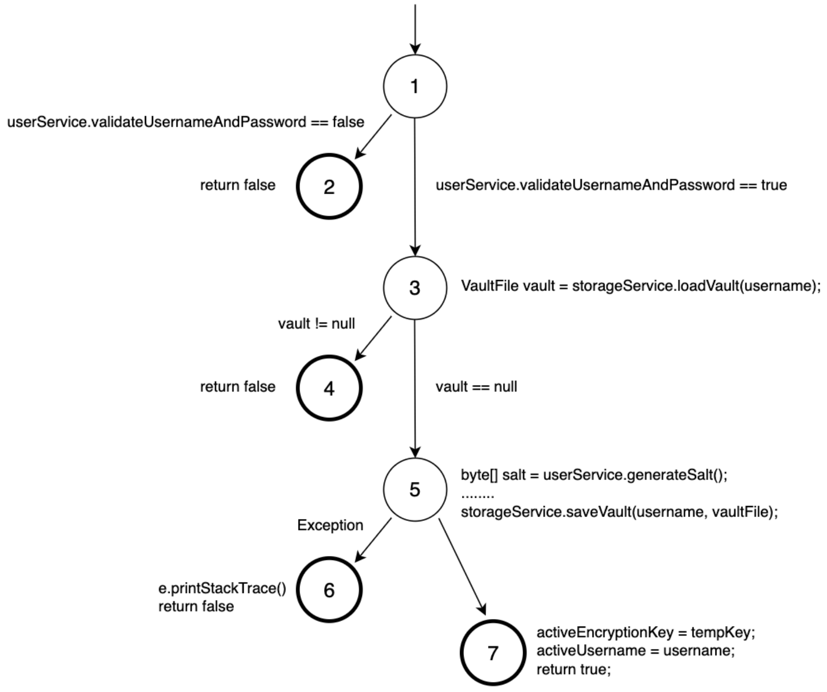
We performed data flow testing on the “register” method of “PasswordManager.java”.

Requirements Tested:

- Users must be able to register for an account with valid credentials and login with them later.

Specifications:

The control flow graph for “register” is:


1
userService.validateUsernameAndPassword == false
return false
2
userService.validateUsernameAndPassword == true
3
VaultFile vault = storageService.loadVault(username);
vault != null
return false
4
vault == null
5
byte[] salt = userService.generateSalt();
....
storageService.saveVault(username, vaultFile);
Exception
e.printStackTrace()
return false
6
7
activeEncryptionKey = tempKey;
activeUsername = username;
return true;

Def-Use Analysis:

Node Defs and Uses:

Node	Defs	Uses
1	{ username, password }	{ }
2	{ }	{ }
3	{ vault }	{ username }
4	{ }	{ }
5	{ salt, tempKey, encryptedAuthKey, authVaultEntry, keys, saltString, vaultFile }	{ password, salt, tempKey, encryptedAuthKey, authVaultEntry, saltString, keys, username, vaultFile }
6	{ e }	{ e }
7	{ activeEncryptionKey, activeUsername }	{ tempKey, username }

Edge Uses:

Edge	Uses
(1,2)	{ username, password }
(1,3)	{ username, password }
(3,4)	{ vault }
(3,5)	{ vault }
(5,6)	{ }

(5,7)	{ }

DU Pairs and DU Paths:

Variable	DU Pairs	DU Paths
username	(1, (1,2)) (1, (1,3)) (1,3) (1,5) (1,7)	[1,2] [1,3] [1,3] [1,3,5] [1,3,5,7]
password	(1, (1,2)) (1, (1,3)) (1,5)	[1,2] [1,3] [1,3,5]
vault	(3, (3,4)) (3,(3,5))	[3,4] [3,5]
salt	(5,5)	[5]
tempKey	(5,5) (5,7)	[5] [5,7]
encryptedAuthKey	(5,5)	[5]
authVaultEntry	(5,5)	[5]
keys	(5,5)	[5]
saltString	(5,5)	[5]
vaultFile	(5,5)	[5]

activeEncryptionKey	None	
activeUsername	None	
e	(6,6)	[6]

Unique DU Paths:

Variable	DU Paths
username	[1,2] [1,3] [1,3,5] [1,3,5,7]
vault	[3,4] [3,5]
salt	[5]
tempKey	[5,7]
e	[6]

We can achieve all-du-paths coverage with the following test paths:

- [1,2]
- [1,3,4]
- [1,3,5,6]
- [1,3,5,7]

Test Cases:

This happens to be the same set of test paths we used to perform path testing in the previous section. These test paths are already covered in the “Structure-based Testing” section of “PasswordManagerTest.java” and there is no point in writing them again.

Results:

All tests passed.

4. Integration Testing

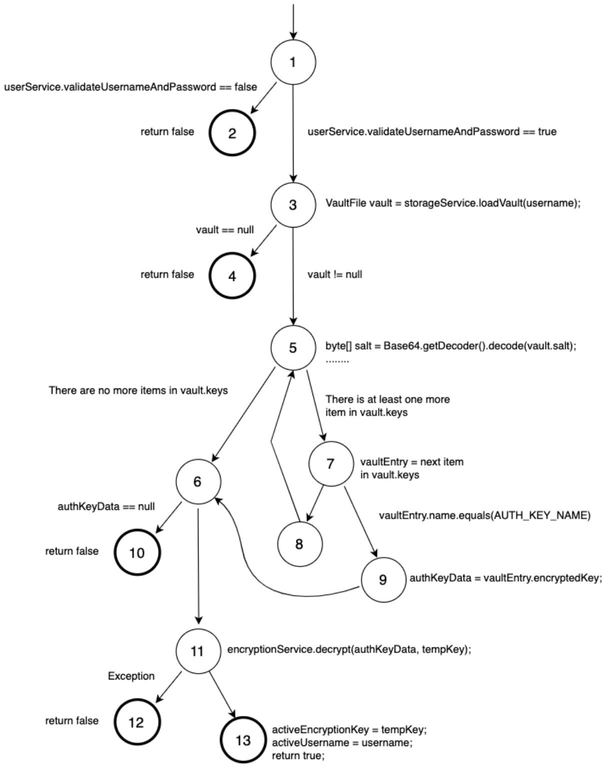
To demonstrate our understanding of integration testing, we will be exploring the connection between “PasswordManager” and “UserService”. The integration of these two modules will be tested in a big-bang style where neither class is mocked. The “UserService” is utilized in the “login” and “register” methods of “PasswordManager”.

Requirements Tested:

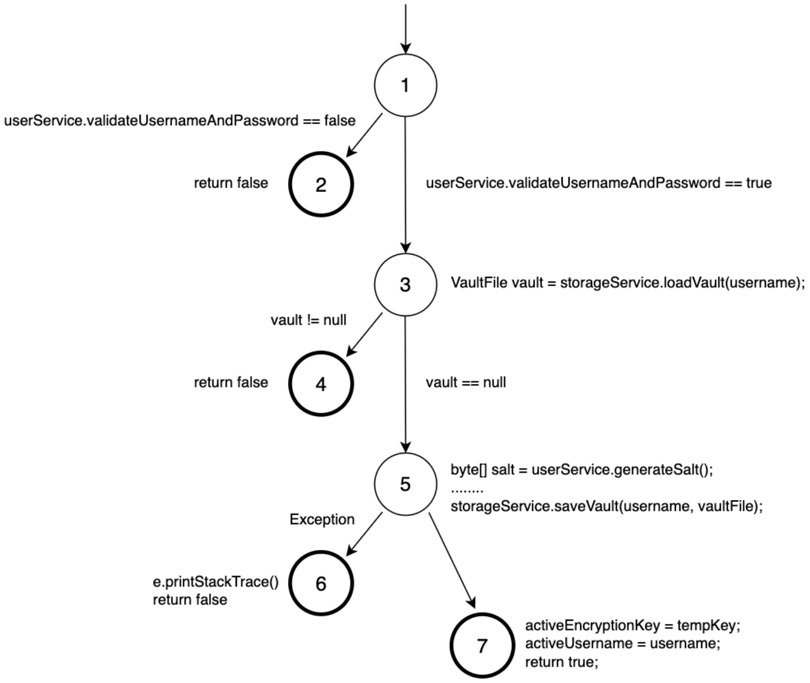
- Users must be able to register for an account with valid credentials and login with them later.

Specifications:

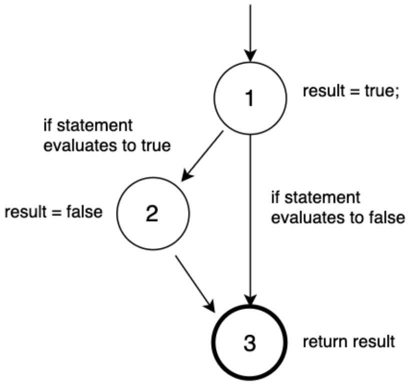
The control flow graphs of all methods are:


1
userService.validateUsernameAndPassword == false
return false
2
userService.validateUsernameAndPassword == true
3
VaultFile vault = storageService.loadVault(username);
vault == null
return false
4
vault != null
5
byte[] salt = Base64.getDecoder().decode(vault.salt);
....
There are no more items in vault.keys
There is at least one more
item in vault.keys
7
vaultEntry = next item
in vault.keys
6
authKeyData == null
return false
10
8
vaultEntry.name.equals(AUTH_KEY_NAME)
9
authKeyData = vaultEntry.encryptedKey;
11
encryptionService.decrypt(authKeyData, tempKey);
Exception
return false
12
13
activeEncryptionKey = tempKey;
activeUsername = username;
return true;

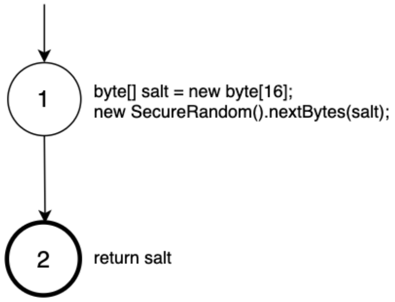
Control Flow Graph for “PasswordManager.login”


1
userService.validateUsernameAndPassword == false
2
return false
userService.validateUsernameAndPassword == true
3
VaultFile vault = storageService.loadVault(username);
vault != null
4
return false
vault == null
5
byte[] salt = userService.generateSalt();
....
storageService.saveVault(username, vaultFile);
Exception
6
e.printStackTrace()
return false
7
activeEncryptionKey = tempKey;
activeUsername = username;
return true;

Control Flow Graph for “PasswordManager.register”



Control Flow Graph for “UserService.validateUsernameAndPassword”


1
byte[] salt = new byte[16];
new SecureRandom().nextBytes(salt);
2
return salt

Control Flow Graph for “UserService.generateSalt”

Using node/edges in place of statement number, the coupling DU pairs between these modules are:

1. (register, username, 1) - (validateUsernameAndPassword, username, (1,2))
2. (register, username, 1) - (validateUsernameAndPassword, username, (1,3))
3. (register, password, 1) - (validateUsernameAndPassword, password, (1,2))
4. (register, password, 1) - (validateUsernameAndPassword, password, (1,3))
5. (login, username, 1) - (validateUsernameAndPassword, username, (1,2))
6. (login, username, 1) - (validateUsernameAndPassword, username, (1,3))
7. (login, password, 1) - (validateUsernameAndPassword, password, (1,2))
8. (login, password, 1) - (validateUsernameAndPassword, password, (1,3))
9. (validateUsernameAndPassword, result, 1) - (register, N/A, (1,3))
10. (validateUsernameAndPassword, result, 2) - (register, N/A, (1,2))
11. (validateUsernameAndPassword, result, 1) - (login, N/A, (1,3))
12. (validateUsernameAndPassword, result, 2) - (login, N/A, (1,2))
13. (generateSalt, salt, 1) - (register, salt, 5)

Tests Cases:

We have achieved all-coupling uses coverage by exercising each of the 13 listed last-def first-use pairs. The tests are in “PasswordManagerTest” under the “Integration Testing” section.

Results:

All tests passed.

5. Specification Based Testing

5.1. Boundary Value Analysis

Requirements Tested:

- The system must validate inputs and reject any invalid requests.

Specifications:

Specifically, we used robust boundary value analysis on a few parts of our system. First, we tested the username and password fields on the login and registration features. According to the requirements, both username and password must be 4–32 characters. Next, we performed boundary value analysis on key name length and key value length. Key names should be between 1 and 64 characters, inclusive. Key values should be between 1 and 1024 characters, inclusive.

Test Cases and Results:

Test Case	Username Length	Password Length	Expected Result	Actual Result
1	3	16	Rejected	Rejected
2	4	16	Accepted	Accepted
3	5	16	Accepted	Accepted
4	16	16	Accepted	Accepted
5	31	16	Accepted	Accepted
6	32	16	Accepted	Accepted
7	33	16	Rejected	Rejected
8	16	3	Rejected	Rejected

9	16	4	Accepted	Accepted
10	16	5	Accepted	Accepted
11	16	16	Accepted	Accepted
12	16	31	Accepted	Accepted
13	16	32	Accepted	Accepted
14	16	33	Rejected	Rejected

Test Case	Key Name Length	Key Value Length	Expected Result	Actual Result
1	0	5	Rejected	Rejected
2	1	5	Accepted	Accepted
3	2	5	Accepted	Accepted
4	5	5	Accepted	Accepted
5	63	5	Accepted	Accepted
6	64	5	Accepted	Accepted
7	65	5	Rejected	Rejected
8	5	0	Rejected	Rejected
9	5	1	Accepted	Accepted
10	5	2	Accepted	Accepted

11	5	5	Accepted	Accepted
12	5	1023	Accepted	Accepted
13	5	1024	Accepted	Terminal Broke!
14	5	1025	Rejected	Terminal Broke!

Interestingly, the terminal we used to test this stopped accepting characters after 1024, preventing submission on the 13th test and preventing entering the final character on the 14th test. We would fix this by decreasing the character maximum in our next development cycle.

5.2. Equivalence Class Testing

Requirements Tested:

- The system must validate inputs and reject any invalid requests.

Specifications:

Username/password equivalence classes for registration/login:

- Length < 4 - Invalid
- 4 <= Length <= 32 - Valid
- 32 < Length - Invalid

Key name equivalence classes for adding/editing/deleting:

- Length < 1 - Invalid
- 1 <= Length <= 64 - Valid
- 64 < Length - Invalid

Key value equivalence classes for adding/editing:

- Length < 1 - Invalid
- 1 <= Length <= 1024 - Valid

- 1024 < Length - Invalid

By testing each of these ranges under the single fault assumption, we can achieve weak robust ECT.

Test Cases and Results:

Test Case	Username Length	Password Length	Expected Result	Actual Result
1	1	8	Rejected	Rejected
2	10	8	Accepted	Accepted
3	35	8	Rejected	Rejected
4	8	1	Rejected	Rejected
5	8	10	Accepted	Accepted
6	8	35	Rejected	Rejected

Test Case	Key Name Length	Key Value Length	Expected Result	Actual Result
1	0	8	Rejected	Rejected
2	8	8	Accepted	Accepted
3	70	8	Rejected	Rejected
4	8	0	Rejected	Rejected

5	8	12	Accepted	Accepted
6	8	1050	Rejected	Terminal Broke!

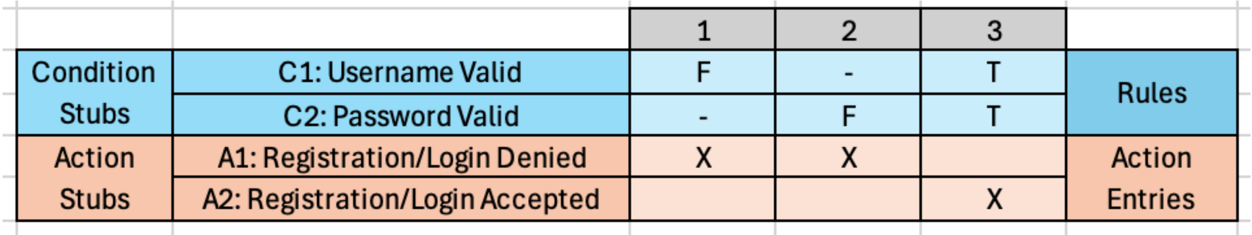
As with boundary value analysis, this highlights the need for reducing the max value length.

5.3. Decision Table Testing

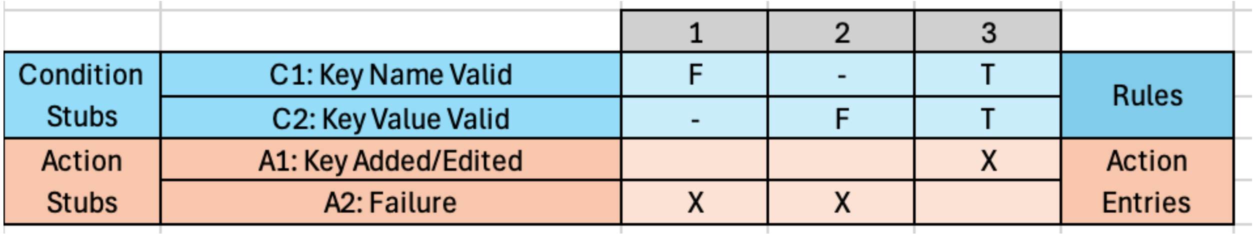
Requirements Tested:

- The system must validate inputs and reject any invalid requests.

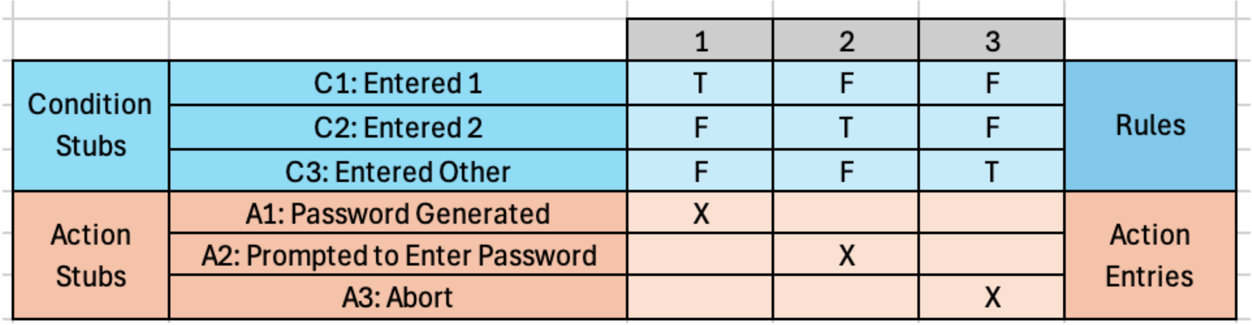
Specifications:



		1	2	3	
Condition Stubs	C1: Username Valid	F	-	T	Rules
	C2: Password Valid	-	F	T	
Action Stubs	A1: Registration/Login Denied	X	X		Action Entries
	A2: Registration/Login Accepted			X	



		1	2	3	
Condition Stubs	C1: Key Name Valid	F	-	T	Rules
	C2: Key Value Valid	-	F	T	
Action Stubs	A1: Key Added/Edited			X	Action Entries
	A2: Failure	X	X		



		1	2	3	
Condition Stubs	C1: Entered 1	T	F	F	Rules
	C2: Entered 2	F	T	F	
	C3: Entered Other	F	F	T	
Action Stubs	A1: Password Generated	X			Action Entries
	A2: Prompted to Enter Password		X		
	A3: Abort			X	

Test Cases and Results:

Test Case	Username Valid	Password Valid	Expected Result	Actual Result

1	T	F	Denied	Denied
2	F	T	Denied	Denied
3	T	T	Accepted	Accepted

Test Case	Key Name Valid	Key Value Valid	Expected Result	Actual Result
1	T	F	Failure	Failure
2	F	T	Failure	Failure
3	T	T	Success	Success

Test Case	Entered	Expected Result	Actual Result
1	1	Password Generated	Password Generated
2	2	Prompted	Prompted
3	abc	Abort	Abort

5.4. State Transition Testing

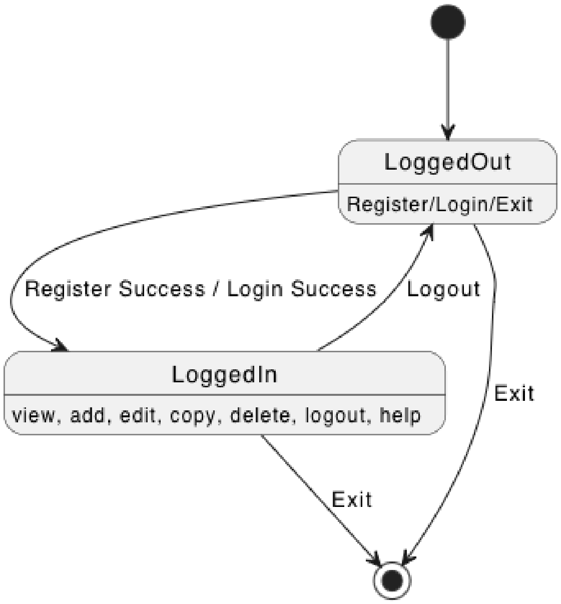
Requirements Tested:

- Users must be able to register for an account with valid credentials and login with them later.
- Users must be able to logout.

- Users must be able to exit the program.

Specifications:

At a high-level, the state of the application is either “logged in” or “logged out”, and everything stems from there. That flow is represented by the following diagram:


LoggedOut
Register/Login/Exit
Register Success / Login Success
Logout
Exit
LoggedIn
view, add, edit, copy, delete, logout, help
Exit

Test Cases and Results:

Test Case	Current State	Action	Expected Next State	Actual Next State
1	Logged out	Login	Logged in	Logged in
2	Logged out	Register	Logged in	Logged in
3	Logged out	Exit	Terminated	Terminated
4	Logged in	Logout	Logged out	Logged out

5	Logged in	View keys	Logged in	Logged in
6	Logged in	Exit	Terminated	Terminated

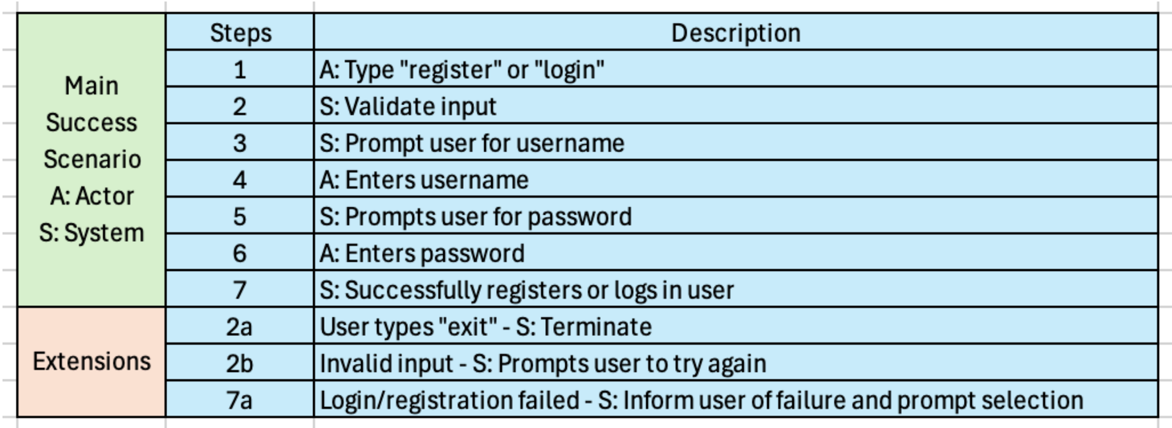
5.5 Use Case Testing

Requirements Tested:

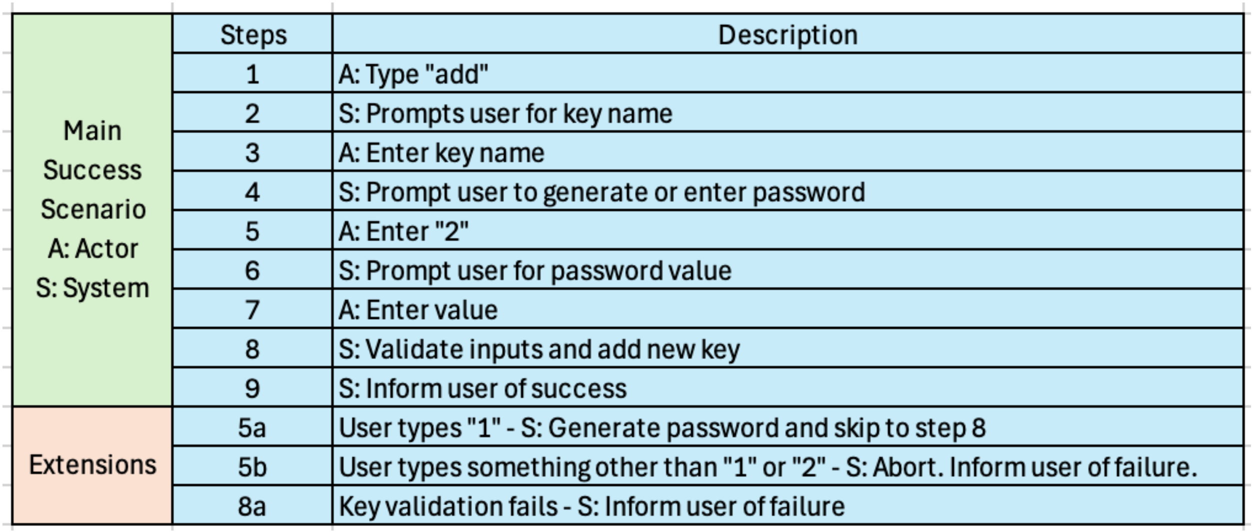
- Users must be able to register for an account with valid credentials and login with them later.
- Users must be able to logout.
- Users must be able to add new keys to their vault.
- Users must be able to edit the values of existing keys.
- Users must be able to delete existing keys.
- Users must be able to view a list of all of their keys.
- Users must be able to copy a key’s value to their clipboard.
- The system must validate inputs and reject any invalid requests.

Specifications

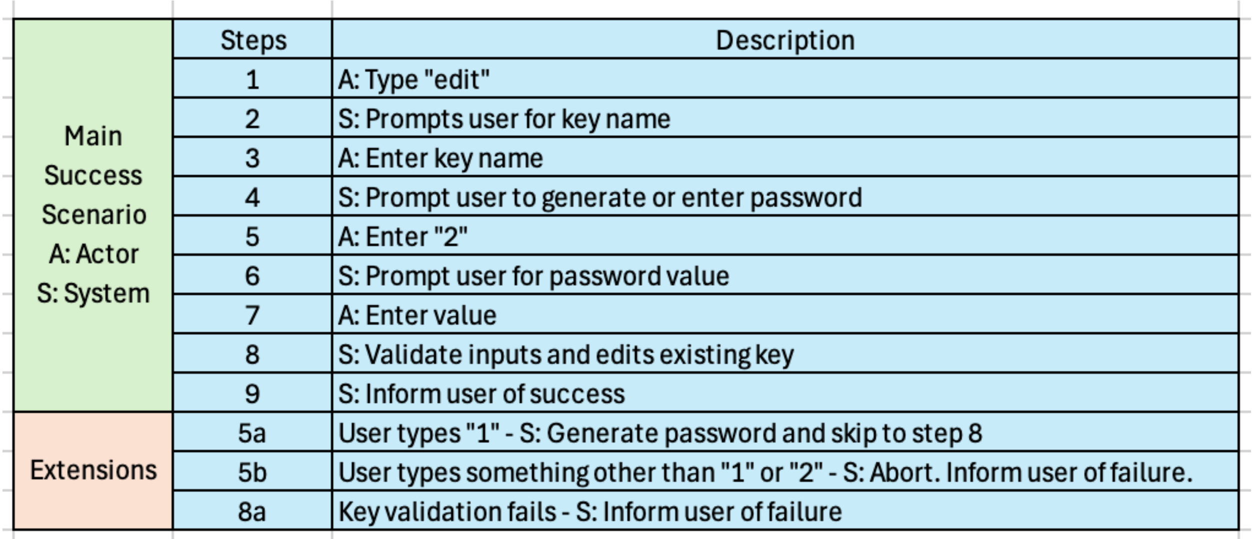
Tests cases were derived off the following flows:



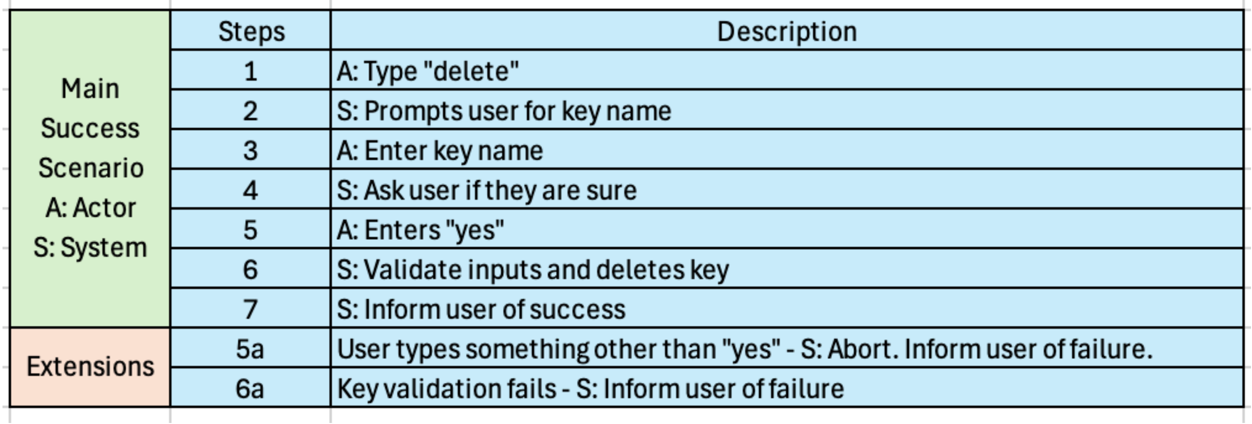
	Steps	Description
Main Success Scenario A: Actor S: System	1	A: Type "register" or "login"
	2	S: Validate input
	3	S: Prompt user for username
	4	A: Enters username
	5	S: Prompts user for password
	6	A: Enters password
	7	S: Successfully registers or logs in user
Extensions	2a	User types "exit" - S: Terminate
	2b	Invalid input - S: Prompts user to try again
	7a	Login/registration failed - S: Inform user of failure and prompt selection



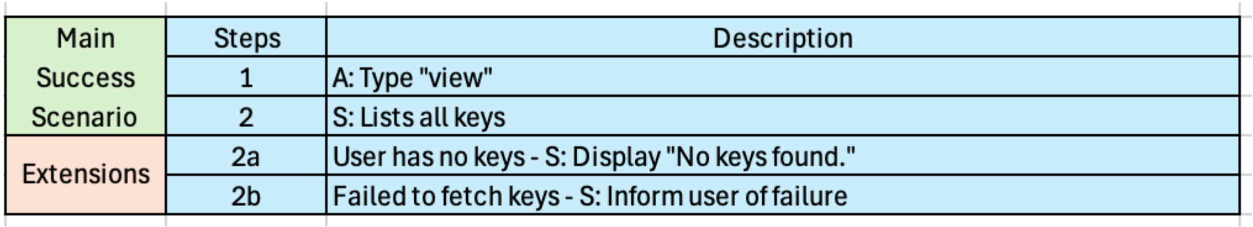
	Steps	Description
Main Success Scenario A: Actor S: System	1	A: Type "add"
	2	S: Prompts user for key name
	3	A: Enter key name
	4	S: Prompt user to generate or enter password
	5	A: Enter "2"
	6	S: Prompt user for password value
	7	A: Enter value
	8	S: Validate inputs and add new key
	9	S: Inform user of success
Extensions	5a	User types "1" - S: Generate password and skip to step 8
	5b	User types something other than "1" or "2" - S: Abort. Inform user of failure.
	8a	Key validation fails - S: Inform user of failure



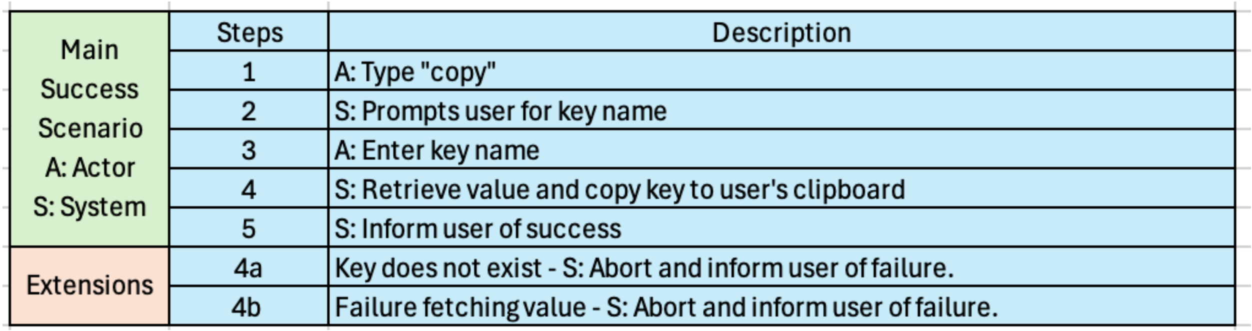
	Steps	Description
Main Success Scenario A: Actor S: System	1	A: Type "edit"
	2	S: Prompts user for key name
	3	A: Enter key name
	4	S: Prompt user to generate or enter password
	5	A: Enter "2"
	6	S: Prompt user for password value
	7	A: Enter value
	8	S: Validate inputs and edits existing key
	9	S: Inform user of success
Extensions	5a	User types "1" - S: Generate password and skip to step 8
	5b	User types something other than "1" or "2" - S: Abort. Inform user of failure.
	8a	Key validation fails - S: Inform user of failure



	Steps	Description
Main Success Scenario A: Actor S: System	1	A: Type "delete"
	2	S: Prompts user for key name
	3	A: Enter key name
	4	S: Ask user if they are sure
	5	A: Enters "yes"
	6	S: Validate inputs and deletes key
	7	S: Inform user of success
Extensions	5a	User types something other than "yes" - S: Abort. Inform user of failure.
	6a	Key validation fails - S: Inform user of failure



Main Success Scenario	Steps	Description
	1	A: Type "view"
	2	S: Lists all keys
Extensions	2a	User has no keys - S: Display "No keys found."
	2b	Failed to fetch keys - S: Inform user of failure



Main Success Scenario A: Actor S: System	Steps	Description
	1	A: Type "copy"
	2	S: Prompts user for key name
	3	A: Enter key name
	4	S: Retrieve value and copy key to user's clipboard
	5	S: Inform user of success
Extensions	4a	Key does not exist - S: Abort and inform user of failure.
	4b	Failure fetching value - S: Abort and inform user of failure.

Test Cases and Results:

Test Case 1: Register New User and Login

Step	Action	Expected Outcome
1	Start VaultGuard CLI	Welcome Message
2	Select [1] Register	Prompt for username and password
4	Enter unique username and password	“Registration Successful" message
5	Type “logout”	User is logged out, main menu shown
6	Select [2] and enter same creds	“Login Successful!" message

Result: Success.

Test Case 2: Add, View, Edit, Delete Key

Step	Action	Expected Outcome
1	Login with valid user	Logged in message
2	Type “add” enter key name, password	“Key added successfully”
3	Type “view”	“Key name” appears in list
4	Type “edit”, key name, new password	“Key updated successfully”
5	Type “delete”, key name, confirm “yes”	“Key deleted successfully”
6	Type “view”	Key list is empty or does not include the “key name”

Result: Success.

Test Case 3: Generate and Copy Password

Step	Action	Expected Outcome
1	Type “add”, choose to generate a strong password	Password is displayed and added
2	Type “Copy”, enter key name	“Password for key copied to clipboard” message

Result: Success.

Test Case 4: Logout and Security

Step	Action	Expected Outcome

1	Type “logout”	“You have been logged out”
2	Try to “add”, “edit”, or “view”	Application prompts to login

Result: Success.

Test Case 5: Invalid Operations (Negative Scenarios)

Step	Action	Expected Outcome
1	Register with an existing username	“Registration Failed."
2	Login with wrong password	“Login Failed."
3	Add duplicate key name	“Failed to add key. It may already exist..."
4	Edit or delete non-existent key	“Failed to update/delete key..."
5	Register/login with invalid credentials	Registration/login fails, error message shown
6	Enter invalid menu command	“Invalid command. Type 'help'..."

Result: Success.

Test Case 6: Password Generation

Step	Action	Expected Outcome
1	Choose to generate a password during add/edit	16-char strong password shown, accepted

Result: Success.