*Xpendit*

Test Plan

May 6, 2019

TABLE OF CONTENTS

[1 Introduction 3](#_Toc7859866)

[1.1 Purpose of The Test Plan Document 3](#_Toc7859867)

[2 Unit Testing 3](#_Toc7859868)

[2.1 Items to be tested/not tested 3](#_Toc7859869)

[2.2 Test Approach(s) 3](#_Toc7859870)

[2.3 Tests 3](#_Toc7859871)

[2.4 Test Pass / Fail Criteria 16](#_Toc7859872)

[3 Use Case Testing 16](#_Toc7859873)

[3.1 TODO 16](#_Toc7859874)

[4 User acceptance Testing 16](#_Toc7859875)

[4.1 TODO 16](#_Toc7859876)

# Introduction

## Purpose of The Test Plan Document

The Test Plan Document outlines the tests which the Xpendit app will go through to ensure the application function as intended. These tests include unit tests, use case tests, and acceptance tests. Only tests for the features which are expected to be finished by the end of the semester (Deliverable 4) are outlined in this document, which will be update if changes occur. It’s intended audience is the project owner, scrum master, and development team. In the future, the document may be shared with others whose input or approval is needed.

# Unit Testing

## Items to be tested/not tested

The features being tested are creating account, logging in, creating a room, adding members to a room, creating a new shared expense and creating a group shared list and the functions related to them.

## Test Approach(s)

Tests will be done using the dart “Testing” package. The package allows users to test each function or unit within Android Studio with fast results.

## Tests

**Test data used for testing the Register GUI, Login GUI, and Room Creation GUI. These tests will be done each GUI. The GUI must display the string correctly when the function is called. Tests will be done to test the function works with both valid and invalid inputs.**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RG\_00 | | | |
| GUI: displayMsg():string | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | call “displayMsg()” with test data | string = “hello world” | GUI displays “hello world” |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RG\_01 | | | |
| Register GUI: displayMsg():string | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | call “displayMsg()” with test data | string = “” | GUI displays empty string |
| Test Case ID: RG\_02 | | | |
| Register GUI: displayMsg():string | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | call “displayMsg()” with test data | string = 4 (integer not a string) | GUI does not display anything. |

**Test data used for testing the Account object. Accounts must be able to create a room, create a charge and sum charges. Tests will be done with valid and invalid inputs.**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: AC\_00 | | | |
| Account: createRoom() | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | call “createRoom()” from an account | (No input empty constructor) | A room linked to the accounted is created in the room database |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: AC\_01 | | | |
| Account: createCharge() | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | call “createCharge()” from an account | (No input empty constructor) | A charge is created on the account |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: AC\_03 | | | |
| Account: sumCharge() | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | No charges added to account | (No input empty method) | Total Charge = 0 |
| 3 | call “createCharge()” from an account |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: AC\_04 | | | |
| Account: sumCharge(): double | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Charges added to account | Charges = [35, 10, 155] | Total Charge = 200 |
| 3 | call “createCharge()” from an account |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: AC\_05 | | | |
| Account: sumCharge() | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Charges added to account | Charges = [infinity, 12, 155] | Total Charges = “error” |
| 3 | call “createCharge()” from an account |  |  |

**Test data used for testing the Register Controller. Register controller must be able to create a user and test passwords for minimum requirements. The minimum requirements being greater than 10 characters and contains at least three types of characters.**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RC\_00 | | | |
| Account: createUser() | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | call “createUser()” | (No input) | A user is created and successfully added to the user database |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RC\_01 | | | |
| Account: minRequirements()”bool | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create test password string | Password = null | false |
| 2 | Call “minRequirements” on test password |  |  |
| Test Case ID: RC\_02 | | | |
| Account: minRequirements()”bool | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create test password string | Password = “p@ssw0rd” | false |
| 2 | Call “minRequirements” on test password |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RC\_03 | | | |
| Account: minRequirements()”bool | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create test password string | Password = “g00dp@ssword” | true |
| 2 | Call “minRequirements” on test password |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RC\_04 | | | |
| Account: minRequirements()”bool | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create test password string | Password = “badPassword” | false |
| 2 | Call “minRequirements” on test password |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RC\_05 | | | |
| Account: minRequirements()”bool | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create test password string | Password = “badPassw0rd” | false |
| 2 | Call “minRequirements” on test password |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RC\_06 | | | |
| Account: minRequirements()”bool | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create test password string | Password = “b@dPassword” | false |
| 2 | Call “minRequirements” on test password |  |  |

**Test data used for testing the User Database. The user data base must be capable of saving a user to the data base and finding a user though email and user name.**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: UD\_00 | | | |
| Account: saveUser() | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create an “Account” | Account with:  Username= “TestUser”  Password= “g0odP@ssword”  Email = “fakeemail@mail.com” | The account is added to the user database. |
| 2 | Call “saveUser()” on the account |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: UD\_01 | | | |
| Account: saveUser() | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create an “Account” | Account = null; | User is not saved to the database |
| 2 | Call “saveUser()” on the account |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: UD\_02 | | | |
| Account: saveUser() | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create an integer | Integer = 45; | Database does not save number (non-Account object). |
| 2 | Call “saveUser()” on the account |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: UD\_03 | | | |
| Account: getUser():Account | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Add Account to data base | Account with:  Username= “TestUser”  Password= “g0odP@ssword”  Email = “fakeemail@mail.com” | Database returns the created account object |
| 2 | Call “getUser(TestUser, g0odP@ssword)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: UD\_04 | | | |
| Account: getUser():Account | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Add Account to data base | Account with:  Username= “TestUser”  Password= “g0odP@ssword”  Email = “fakeemail@mail.com” | Database returns null |
| 2 | Call “getUser(TestUser, wrongpassword)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: UD\_05 | | | |
| Account: getUser():Account | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Add Account to data base | Account with:  Username= “TestUser”  Password= “g0odP@ssword”  Email = “fakeemail@mail.com” | Database returns null |
| 2 | Call “getUser(WrongUser, g0odP@ssword)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: UD\_06 | | | |
| Account: getUser():Account | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | No account added to database | (none) | Database returns null |
| 2 | Call “getUser(TestUser, g0odP@ssword)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: UD\_07 | | | |
| Account: findUser():Account | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Add Account to data base | Account with:  Username= “TestUser”  Password= “g0odP@ssword”  Email = “fakeemail@mail.com” | Database returns the added Account. |
| 2 | Call “findUser(TestUser)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: UD\_08 | | | |
| Account: findUser():Account | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Add Account to data base | Account with:  Username= “TestUser”  Password= “g0odP@ssword”  Email = “fakeemail@mail.com” | Database returns null |
| 2 | Call “findUser(uncreatedUser)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: UD\_09 | | | |
| Account: findUser():Account | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Add Account to data base | Account with:  Username= “TestUser”  Password= “g0odP@ssword”  Email = “fakeemail@mail.com” | Database returns added account |
| 2 | Call “findUser(fakeemail@mail.com)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: UD\_10 | | | |
| Account: findUser():Account | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Add Account to data base | Account with:  Username= “TestUser”  Password= “g0odP@ssword”  Email = “fakeemail@mail.com” | Database returns null |
| 2 | Call “findUser(random@mail.com)” |  |  |

**Test data used for testing the Login GUI. The Login GUI must display the string correctly when the function is called. Tests will be done to test the function works with both valid and invalid inputs.**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: LC\_00 | | | |
| Login Controller: verify(username): boolean | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create account | Username on account is “TestUsername” | true |
| 2 | Add account to user database |  |  |
| 3 | Call “verify(TestUsername)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: LC\_01 | | | |
| Login Controller: verify(username): boolean | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create account | Username on account is “TestUsername” | false |
| 2 | Add account to user database |  |  |
| 3 | Call “verify(invalidUsername)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: LC\_02 | | | |
| Login Controller: verify(username): boolean | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create account | Username on account is “TestUsername” | false |
| 2 | Add account to user database |  |  |
| 3 | Call “verify(“”)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: LC\_03 | | | |
| Login Controller: verify(username): boolean | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create account | Username on account is “” | true |
| 2 | Add account to user database |  |  |
| 3 | Call “verify(“”)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: LC\_04 | | | |
| Login Controller: verify(username): boolean | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create account | Username on account is “” | false |
| 2 | Add account to user database |  |  |
| 3 | Call “verify(username)” |  |  |

**Test data used for testing the Room/Room Manager. Rooms must be able to add users and add a group charge.**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RO\_00 | | | |
| Room: addUser(): void | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create account | Username on account is “TestUsername” | The user is added to the room |
| 2 | Add account to user database |  |  |
| 3 | Call “addUser(TestUsername)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RO\_01 | | | |
| Room: addUser(): void | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create account and room | Username on account is “TestUsername” | Nothing added to room |
| 2 | Add account to user database |  |  |
| 3 | Call “addUser()” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RO\_02 | | | |
| Room: addUser(): void | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create account | Username on account is “TestUsername” | Nothing added to room |
| 2 | Add account to user database |  |  |
| 3 | Call “addUser(waffle)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RO\_03 | | | |
| Room: addUser(): void | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create account | Email on account is “fakeemail@mail.com” | User added to room |
| 2 | Add account to user database |  |  |
| 3 | Call “addUser(fakeemail@mail.com)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RO\_04 | | | |
| Room: addUser(): void | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create account | Email on account is “fakeemail@mail.com” | No user added to room |
| 2 | Add account to user database |  |  |
| 3 | Call “addUser(wrongemail@mail.com)” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RO\_05 | | | |
| Room: addUser(): void | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create account | Email on account is “fakeemail@mail.com” | No user added to room |
| 2 | Add account to user database |  |  |
| 3 | Call “addUser()” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RO\_06 | | | |
| Room: addGroupCharge(): void | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create two accounts | Two accounts with different usernames | Charge added to each account |
| 2 | Add accounts to user database |  |  |
| 3 | Add accounts to the room |  |  |
| 4 | Call “addGroupCharge()” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RO\_07 | | | |
| Room: addGroupCharge(): void | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Create one account | Sample account values from other tests | Charge added to the account |
| 2 | Add accounts to user database |  |  |
| 3 | Add accounts to the room |  |  |
| 4 | Call “addGroupCharge()” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RO\_08 | | | |
| Room: addGroupCharge(): void | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Add no accounts to the room | Sample account values from other tests | No error occurs |
| 2 | Call “addGroupCharge()” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RO\_09 | | | |
| Room: addChargeToUsers(): void | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Add no accounts to the room | Sample account values from other tests | No error occurs |
| 2 | Call “addChargeToUsersCharge()” |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID: RO\_10 | | | |
| Room: addChargeToUsers(): void | | | |
| Step | Test Steps | Test Data(s) | Expected Result |
| 1 | Add three accounts to the room |  | Charge applied to the correct users |
| 2 | Call “addChargeToUsersCharge()” on only two of the accounts |  |  |

Test Cases for lists will be added in Deliverable 4

## Test Pass / Fail Criteria

Each unit must provide the correct functionality when provided the expected input. For larger units, if applicable, proper error handling must occur as well for invalid inputs.

# Use Case Testing

## OVerview

Use case testing is split into the two major categories of interface use cases and financial use cases.

## Interface Use Cases

|  |  |
| --- | --- |
| **Title** | Log In |
| **Actor(s)** | User |
| **Main Scenario** | 1: User enters username and password  2: User submits information  3: The system verifies user  4: The system logs user in |
| **Alternatives** | 3a: Username does not exist  3a1: Display error message asking to try again or register  3b: Password is incorrect  3b: Display error message asking to try again  3c: The user database cannot be reached  3c1: Connection error displayed asking user to try again  4a: System does not log user in  4a1: User remains at login page |
| **Test Situations** | 1: User logs in successfully  2: Username does not exist in database  3: Password is incorrect  4: Cannot connect to user database  5: User database malfunctioning (does not log user in) |

|  |  |
| --- | --- |
| **Title** | Create Account |
| **Actor(s)** | User |
| **Main Scenario** | 1: User enters desired username and password  2: User submits information  3: System checks if the minimum requirements are met  4. User is added to the database |
| **Alternatives** | 3a: Username already exists  3a1: Display message asking user to choose a new username or login  3b: Password does not meet minimum requirements  3b1: Display message asking to create stronger password  3c: User database cannot be reached  3c1: Connection error displayed asking user to try again  4a: System does not save user to database  4a1: User proceeds to login but is unable to login |
| **Test Situations** | 1: User creates account in successfully  2: Username already exists in database  3: Password requirements not met  4: Cannot connect to user database  5: User database malfunctioning (does not add user to database) |

|  |  |
| --- | --- |
| **Title** | View total outstanding debt |
| **Actor(s)** | User |
| **Main Scenario** | 1: User selects “total owed” button  2: The users total outstanding debt is displayed |
| **Alternatives** | 1a: Total owed button does not work  1a1: User cannot view total owed  2a: The incorrect debt is displayed  2a: User assumes they owe the incorrect amount and incorrectly plans  2b: No debt is displayed  2b1: User cannot view total owed |
| **Test Situations** | 1: User views total outstanding debt successfully  2: Button is malfunctioning (does nothing on press)  3: The incorrect amount is shown  4: Prompt is shown but debt amount is missing |

|  |  |
| --- | --- |
| **Title** | View individual charges assessed |
| **Actor(s)** | User |
| **Main Scenario** | 1: User selects “History”  2: System displays list of individual charges  3: User exists dialog  4: User returned to previous menu |
| **Alternatives** | 1a: History button does not work  1a1: User cannot view charges  2a: The incorrect charges are displayed  2a: User assumes they have different charges than they actually do  2b: Charges are shown (when there are)  2b1: User cannot view charges |
| **Test Situations** | 1: User views individual charges successfully  2: Button is malfunctioning (does nothing on press)  3: The incorrect charges are shown (incorrect amounts or missing some)  4: Prompt is shown but charges are missing |
| **Title** | Share list |
| **Actor(s)** | User |
| **Main Scenario** | 1: User enters list name  2: System adds list to database  3: All group members can view/edit list |
| **Alternatives** | 2a: The system does not add list to database  2a1: |
| **Test Situations** | 1: User views individual charges successfully  2: Button is malfunctioning (does nothing on press)  3: The incorrect charges are shown (incorrect amounts or missing some)  4: Prompt is shown but charges are missing |

## Financial Use Cases

To be completed in Deliverable 4

# User acceptance Testing

## TODO

To be completed in Deliverable 4.