

SE 3XA3: Development Plan PocketSaver

Team 12

Mevin Mathew, mathem1

Shalmi Patel, patels19

Diya Mathews, mathewsd

December 7, 2017

Contents

1	General Information	1
1.1	Purpose	1
1.2	Scope	1
1.3	Acronyms, Abbreviations, and Symbols	1
1.4	Overview of Document	1
2	Plan	2
2.1	Software Description	2
2.2	Test Team	2
2.3	Automated Testing Approach	2
2.4	Testing Tools	2
2.5	Testing Schedule	2
3	System Test Description	3
3.1	Tests for Functional Requirements	3
3.1.1	Input	3
3.1.2	Application Output	4
3.2	Tests for Nonfunctional Requirements	5
3.2.1	Look and Feel	5
3.2.2	Usability and Humanity Requirements	5
3.2.3	Performance Requirements	5
3.2.4	Security Requirements	5
3.2.5	Legal Requirements	6
3.3	Traceability Between Test Cases and Requirements	6
4	Tests for Proof of Concept	6
4.1	Page Flow	6
4.2	DataBase Testing	6
5	Comparison to Existing Implementation	7
6	Unit Testing Plan	8
6.1	Unit testing of internal functions	8
6.2	Unit testing of output files	8

7	Appendix	9
7.1	Symbolic Parameters	9
7.2	Usability Survey Questions?	9

List of Tables

1	Revision History	ii
2	Software Description	2

List of Figures

Table 1: **Revision History**

Date	Version	Notes
10/27/2017	1.0	Initial
12/06/2017	1.1	Rev1

This document is to describe to Testing Plan for the PocketSaver mobile application for iOS and Android

1 General Information

1.1 Purpose

The purpose of testing this project is to confirm all requirements that were outlined in the Requirements Specifications as well as check whether or not the requirements are met or not. This also helps to determine if the application is implemented correctly.

1.2 Scope

In order to test the re-implementation of CoCoin into the PocketSaver application, the test plan provides a basis and structure for the testing that is to transpire. Its objective is to prove that PocketSaver has met its requirements as specified in the Requirement Document as well as create metrics to the requirements so that the requirements specified are quantifiable. The testing plan allows for a means and structure for the testing on PocketSaver. This document will also provide what is to be tested of the application and outline testing methods as well as tools that are to be utilized for testing.

1.3 Acronyms, Abbreviations, and Symbols

There are no acronyms, abbreviations or symbols that will be used throughout this document that need to be highlighted.

1.4 Overview of Document

The PocketSaver application will re-implement the project CoCoin from GitHub. The application will allow the user to enter daily personal finances in order to keep a detailed record of their transactions in everyday life. All of the applications software requirements can be referred to in the Requirements Document. This document demonstrates how the personal finance application PocketSaver will be tested, the testing schedule, and the tools that are utilized for testing.

2 Plan

2.1 Software Description

PocketSaver is a financing application created to keep track of personal finances. The backend of the application is coded in C# and the front end is coded in XAML. The data collected by the user is stored in a free online database. The table below describes various components of the software that are affected when a new transaction is inputed:

Table 2: Software Description

Inputs	Outputs	Functions of Software being tested
Enter new entry for expense	Transaction page is updated	Transaction page is updated, Accurate
Enter new entry for expense	Database is updated	Database is updated
Enter new entry for expense	Total expense updated	Total expense updates
		Smooth transition between pages

2.2 Test Team

The team as a whole will be responsible for testing. Please refer to the gantt chart for more details and responsibilities of the testing.

2.3 Automated Testing Approach

We will be using Microsoft Unit Test Framework for unit testing. Microsoft Unit testing works well with MSTest, which makes it easy to test with the application. The unit testing will run the program and compare the output values with the expected values.

2.4 Testing Tools

Testing tools that will used is MStesting unit. It is a automated testing unit.

2.5 Testing Schedule

Refer to Gantt chart for the testing Schedule, Gantt Chart

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Input

1. ~~inputTest-id1~~ Test User Input

inputTest-id1 : Testing if users input is being received

Type: Functional, Dynamic, Automatic

Initial State: Application is on the enter transaction page

Input/Condition: User inputs their transaction details and clicks Enter

Output/Result: The transaction should update the database

How test will be performed: When user enters a new transaction the database should update and should have accurate data. We will use unit testing to ensure the data in the database matches the information provided by the user.

2. ~~inputTest-id2~~ Test Entry Modification

inputTest-id2 : Testing if user is able to modify entries

Type: Functional, Dynamic, Automatic

Initial State: Database has at least one transaction

Input/Condition: User selects an existing transaction and modifies a parameter

Output/Result: Modified transaction should update according to the modifications in the database

How test will be performed: When user modifies an existing transaction the database should update and should have accurate data. We will use unit testing to ensure the data in the database matches the modification the user inputted.

3. ~~inputTest-id3~~ Test Entry Deletion

inputTest-id3 : Testing if user is able to delete entries

Type: Functional, Dynamic

Initial State: Database has at least one transaction

Input/Condition: User selects an existing transaction and deletes it

Output/Result: Deleted transaction should not appear within database

How test will be performed: When user deletes an existing transaction the database should update and correspondingly delete that entry. We will use unit testing to ensure the data in the database matching the

deleted transaction is removed.

3.1.2 Application Output

1. ~~App-test-id1~~
outputTest-id1 : Testing if the graph is up-to-date and has no delay
Type: ~~Functional, Dynamic, Manual~~
Initial State: ~~Database has at least one transaction~~
Input/Condition: ~~User enters a new transaction~~
Output/Result: ~~Entered transaction should appear on graph~~
How test will be performed: ~~When user enters a new transaction the graph should update and should have accurate data. We will view the graph and ensure that the information is accurate and without delay manually.~~
2. ~~App-test-id2~~ Test delay of Cumulative Amount Spent
outputTest-id1 : Testing if the total amount spent is up-to-date and has no delay
Type: Functional, Dynamic **Automatic**
Initial State: Database has at least one transaction
Input/Condition: User enters a new transaction
Output/Result: Entered transaction should appear in database
How test will be performed: When user enters a new transaction the total expenses should update and should have accurate data. We will use unit testing to view the total expenses amount and ensure that it is accurately updated and without delay.
3. ~~App-test-id3~~ Testing Transaction List Page
outputTest-id2 : Testing if the transaction page is up-to-date and has no delay
Type: Functional, Dynamic, **Automatic**
Initial State: Database has at least one transaction
Input/Condition: User enters a new transaction
Output/Result: Entered transaction should appear in database
How test will be performed: When user enters a new transaction the transaction page should update and should have accurate data. We will use unit testing to view the transaction page manually and ensure that the inputted information is accurately displayed and without delay.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

1. lookTest-id1

lookTest-id1 : This will be tested by survey question 3.

Pass: An average of 90% rating (4.5/5)

3.2.2 Usability and Humanity Requirements

1. usabilityTest-id1

usabilityTest-id1 : This will be tested by survey question 1 and 2

Pass: An average of 90% rating (4.5/5)

2. usabilityTest-id2

usabilityTest-id2 : This will be tested by survey question 7

Pass: An average of 90% rating (4.5/5)

3.2.3 Performance Requirements

1. performanceTest-id1

performanceTest-id1 : This will be tested by survey question 4

Pass: An average of 90% rating (4.5/5)

2. ~~performanceTest-id2~~

~~performanceTest-id2 : This will be tested by survey question 5~~

~~Pass: An average of 90% rating (4.5/5)~~

3. performanceTest-id2

performanceTest-id2 : This will be tested by survey question 6

Pass: An average of 90% rating (4.5/5)

3.2.4 Security Requirements

1. legalTest-id1

legalTest-id1 : This will be tested by survey question 10

Pass: An average of 90% of surveys say no

3.2.5 Legal Requirements

1. securityTest-id1
securityTest-id1 : This will be tested by survey question 8
Pass: An average of 90% rating (4.5/5)

3.3 Traceability Between Test Cases and Requirements

All of our test cases are designed so that by satisfying them, the requirements will also be satisfied. All the functional requirements will be tested as mentioned in section 3.1 of this document. All non-functional requirements will be tested using a survey 7.2 to assess whether they are satisfied.

4 Tests for Proof of Concept

4.1 Page Flow

1. flow-test-id1
flow-test-id1 : Open application
Type: manual
Initial state: On Load
Input: none
Output : Main page is loaded
How test will be performed: User clicks on application button, which will launch the application, and then display the main page.
2. flow-test-id2
flow-test-id1 : Flow to transaction page
Type: Dynamic
Initial State: home page
Input: None
Output: Transaction page is loaded after the home page
How test will be performed: user will click on transaction page, while will lead the user to the transaction page.

4.2 DataBase Testing

1. DB-test-id1

DB-test-id1 : Adding transaction
Type: Manual
Initial State: database is up-to-date
Input: transaction
Output: database is updated, and transaction page is updated
How test will be performed: This will be tested through Microsoft Unit Testing.

2. DB-test-id2
DB-test-id2 : Deleting transaction
Type: Manual
Initial State: transaction exists in database
Input: none
Output: Transaction is deleted from the database, and transaction page
How test will be performed: user will select a transaction, and click the delete button. Which will delete the transactions on screen.
3. DB-test-id3
DB-test-id3 : Editing transaction
Type: Manual
Initial State: Transaction page
Input: None
Output: transaction is updated
How test will be performed: Through Microsoft Unit testing , a transaction will edited, and updated in the database.

5 Comparison to Existing Implementation

As we strived to accomplish, we created a smoother user interface than the existing implementation, CoCoin. PocketSaver uses an online database rather than a local one like CoCoin to ensure data integrity and save space on the user's local device. Another obstacle we had to over come was the language barrier, CoCoin was written in Mandrian, making it very difficult for us to follow along the project.

6 Unit Testing Plan

The Microsoft Unit Test framework will be used to implement the unit testing for this project. This does not require any further downloads or installations because the Microsoft Unit Test framework comes installed with Microsoft Visual Studio 2017 Community Edition.

6.1 Unit testing of internal functions

To unit test the internal functions of the PocketSaver application, the values of the API will need to be checked in order to verify that those are the correct values that are saved in the database. For example, if a user inputs a transaction, the database needs to be checked and verified that the data was inputted correctly. For any functions that have a return (example: a total expense calculation), any test cases that have the proper input provided as well as expected output will suffice in terms of testing the function. We know that all functions cannot be tested and therefore our aim is to have about 40% of the functions to be unit tested while the others are tested manually.

6.2 Unit testing of output files

Since the PocketSaver application does not produce an output file, there is not a need to test in this area. There will be no unit testing of the output since display will be a direct output of the internal functions that are being run by the application in the background.

7 Appendix

7.1 Symbolic Parameters

N/A

7.2 Usability Survey Questions?

Please provide a rating from 1 to 5 for the following statements (1 - Strongly Disagree, 2 - Disagree, 3 - Neutral, 4 - Agree, 5 - Strongly Agree) :

1. The application is easy to navigate and easy to use.
2. The colour scheme is visually appealing.
3. Entering/modifying/deleting transactions are reflected on the total amount spent with little to no delay
4. Entering/modifying/deleting transactions are reflected on the transaction page with little to no delay
5. ~~Entering/modifying/deleting transactions are reflected on the graph with little to no delay~~
6. The application runs smoothly and with minimal wait/loading time.
7. I was comfortable with entering all information that the application asked me to.
8. Please provide detailed answers for the following questions (point-form is acceptable)
9. Does the application run smoothly on your device? Please indicate the device you used.
10. Were you offended by anything in this game? Please provide details
11. Is there anything you feel is missing or could be improved in this application?