SE 3XA3: Test Report
PocketSaver

Team 12
Mevin Mathew, mathem1
Shalmi Patel, patels19
Diya Mathews, mathewsd

December 7, 2017

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
12/04/2017	1.0	Initial
12/06/2017	1.1	Rev1

This document describes the results of all testing done for PocketSaver based on the Test Plan we documented.

1 Functional Requirements Evaluation

1.0.1 Input

1. 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
   Expected Output/Result: The transaction should update the database
   Actual Output/Result: The transaction is updated on the database
   Result: Unit Test Passed

2. 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
   Expected Output/Result: Modified transaction should update according to the modifications in the database
   Actual Output/Result: The modified transaction is correctly updated on the database
   Result: Unit Test Passed

3. 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
   Expected Output/Result: Deleted transaction should not appear within database
   Actual Output/Result: The deleted transaction is deleted on the database
   Result: Unit Test Passed

4. Set monthly budget inputTest-id4 : Testing if user can set monthly budget
   Type: Functional, Dynamic, Manual
   Initial State: no budget set
   Input/Condition: User enters a numerical value
   Expected Output/Result: budget set, StorageSV is updated
   Actual Output/Result: The budget is set and StorageSV is updated
   Result: Manual Test Passed

5. View monthly expenses inputTest-id5 : Select month to view expenses.
   Type: Functional, Dynamic, Manual
   Initial State: 'select one of the following options' is displayed
   Input/Condition: User selects month
   Expected Output/Result: Month's total expense is displayed
   Actual Output/Result: Correct total expense is displayed
   Result: Manual Test Passed

1.0.2 Application Output

1. 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
   Initial State: Database has at least one transaction
   Input/Condition: User enters a new transaction on current date
   Expected Output/Result: Entered transaction should appear in database and total amount spent today should be incremented.
   Actual Output/Result: The entered transaction is present on the database and daily amount is incremented
   Result: Unit Test Passed

2. 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
   Expected Output/Result: Entered transaction should appear in database and transaction list view

Actual Output/Result: The entered transaction is present on the database and transaction list view
Result: Unit Test Passed

3. View daily expenses outputTest-id3 : Display current days' expenses
Type: Functional, Dynamic, Manual
Initial State: 'select one of the following options' is displayed
Input/Condition: User selects 'Daily'
Expected Output/Result: 'Daily' total expense is displayed
Actual Output/Result: The current days' expense is correctly displayed
Result: Unit Test Passed

4. Display current budget outputTest-id4 : Display current budget
Type: Functional, Dynamic, Manual
Initial State: No budget is entered
Input/Condition: User inputed numerical value into textbox
Expected Output/Result: Inputted budget value should be shown to user
Actual Output/Result: Current budget is displayed correctly
Result: Unit Test Passed

2 Nonfunctional Requirements Evaluation

The nonfunction requirements were tested using a combination of manual testing, the usabilty survey we referenced in the test plan, and the feedback from the inlab presentation. Based on the feedback from the survey and inlab presentation, we made some modifications to our project and added a couple of features to better improve out application.

2.1 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. The application runs smoothly and with minimal wait/loading time.
6. I was comfortable with entering all information that the application asked me to.
7. Please provide detailed answers for the following questions (point-form is acceptable)
8. Does the application run smoothly on your device? Please indicate the device you used.
9. Were you offended by anything in this game? Please provide details
10. Is there anything you feel is missing or could be improved in this application?

2.1.1 Look and Feel

1. lookTest-id1
   lookTest-id1 : This will be tested by survey question 4 and 3 and 8
   Expected Result: An average of 90% rating (4.5/5)
   Actual Result: Test Passed (Q3: 18/20 * 100% = 90%, 20/20 * 100% = 100%)

2.1.2 Usability and Humanity Requirements

1. usabilityTest-id1
   usabilityTest-id1 : This will be tested by survey question 2 and 3
   Expected Result: An average of 90% rating (4.5/5)
   Actual Result: Test Passed (Q1: 19/20 * 100% = 95%, Q2: 18/20 * 100% = 90%)
2. usabilityTest-id2
   usabilityTest-id2 : This will be tested by survey question 6
   Expected Result: An average of 90% rating (4.5/5)
   Actual Result: Test Passed (20/20 * 100% = 100%)

2.1.3 Performance Requirements

1. performanceTest-id1
   performanceTest-id1 : This will be tested by survey question 5
   Expected Result: An average of 90% rating (4.5/5)
   Actual Result: Test Passed (19/20 * 100% = 95%)

2. performanceTest-id2
   performanceTest-id2 : This will be tested by survey question 6
   Expected Result: An average of 90% rating (4.5/5)
   Actual Result: Test Passed (20/20 * 100% = 100%)

2.1.4 Legal Requirements

1. legalTest-id1
   legalTest-id1 : This will be tested by survey question 9
   Expected Result: An average of 90% of surveys say no
   Actual Result: Test Passed (20/20 * 100% = 100%)

2.1.5 Security Requirements

1. securityTest-id1
   securityTest-id1 : This will be tested by survey question 7
   Expected Result: An average of 90% of surveys say no
   Actual Result: Test Passed (18/20 * 100% = 90%)

2.1.6 Additional Suggestions from the Survey and inLab Feedback

- Add a today button to make it easier to enter todays date when entering a transaction.
- A graph implementation to view monthly transaction
- An overview of multiple months
- Filter through transaction list page

3 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.

4 Unit Testing

Unit Testing was completed using the Visual Studios Testing Framework. The tests are located in the src directory as its own solution. There were 4 tests for the API service and 3 tests for the homepage view model. When running the unit tests, it is important to run the homepage view model tests before the API service because the API service generates new data which the homepage view model unit tests are not expecting. Therefore causing the homepage view model to fail even if its functionality is correct.

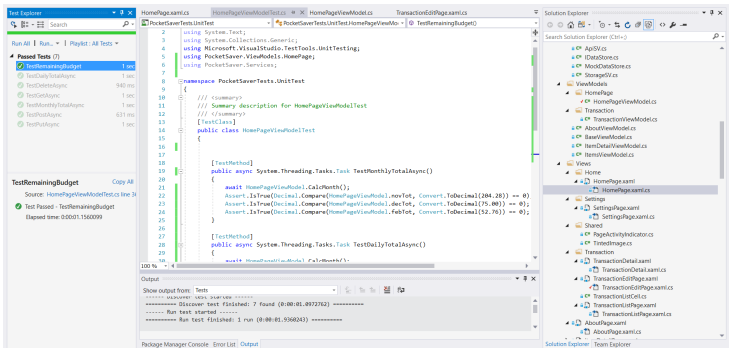

Figure 1: Sample of the Unit Testing in Visual Studios

5 Changes Due to Testing

After conducting the non functional testing, we created a today button for easier use of the entering a transaction function of the applicaiton. We also fixed a few bugs that those who took our survey caught. We ensured that the

navagation between the pages was correct and smooth. Taking into account the feedback from our tests, we were able to improve our application.

6 Automated Testing

Automated Testing was implemented using the Visual Studios Testing Framework as mentioned above. The automated testing was implementing to ensure the API calls to the database were working. This is the core of the application since without this none of the data would be saved and nothing else would work. Therefore ensuring the API calls were working was very important.

7 Trace to Requirements

The tables are found at the appendix at Table 2 and Table 3.

8 Trace to Modules

The tables are found at the appendix at Table 4.

9 Code Coverage Metrics

For our application, every testcase passed, through both unit testing and manual testing. During testing both functional and non-functional requirements were tested, with normal cases and edge cases. The application has a smooth user interface, there for it was easy to validate functionality through a usability survey (inserted in this document) where targeted users tested the application. Therefore, our goal for 90% coverage metric was reached.

10 Appendix

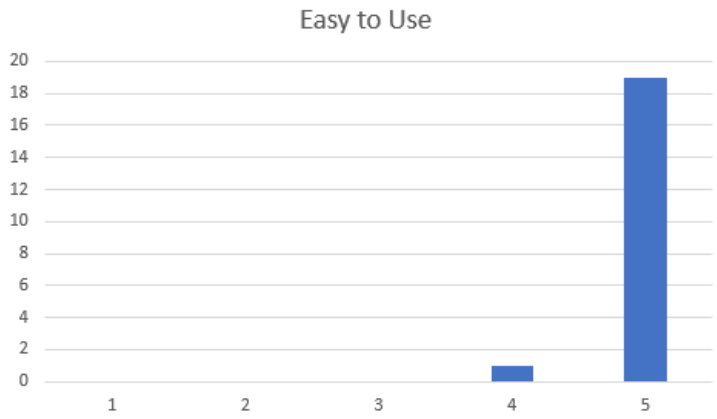

Figure 2: Results of Question 1 of the Survey

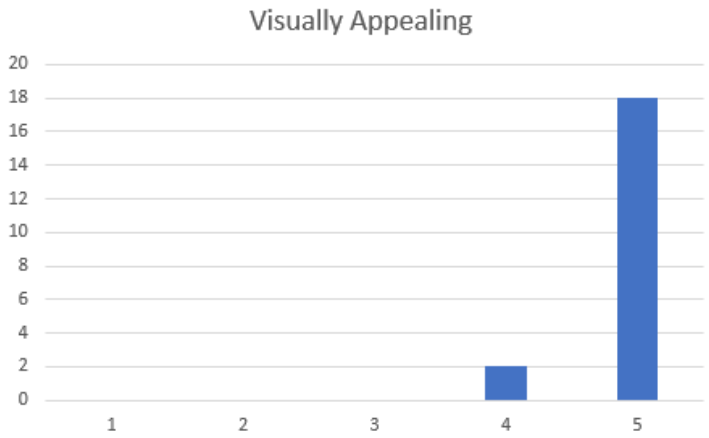

Figure 3: Results of Question 2 of the Survey

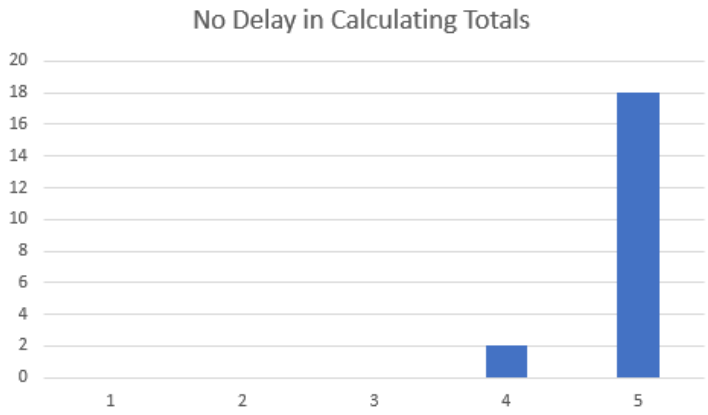

Figure 4: Results of Question 3 of the Survey

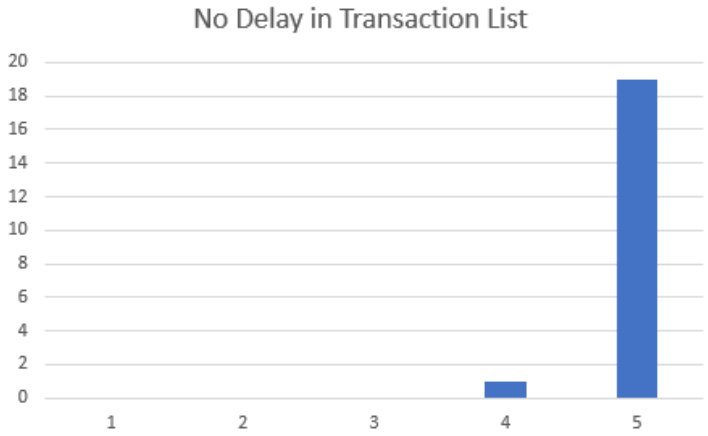

Figure 5: Results of Question 4 of the Survey

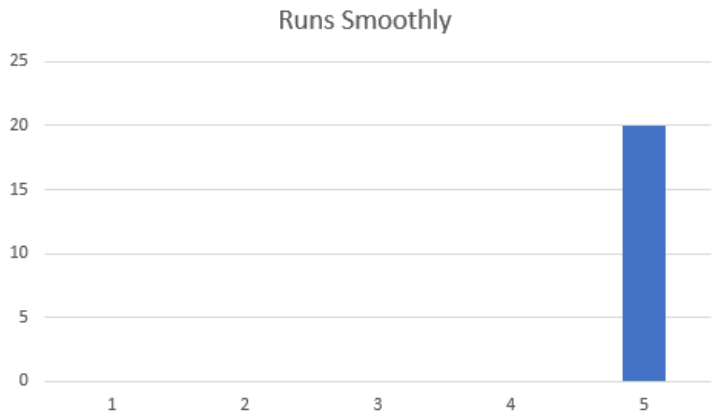

Figure 6: Results of Question 5 of the Survey

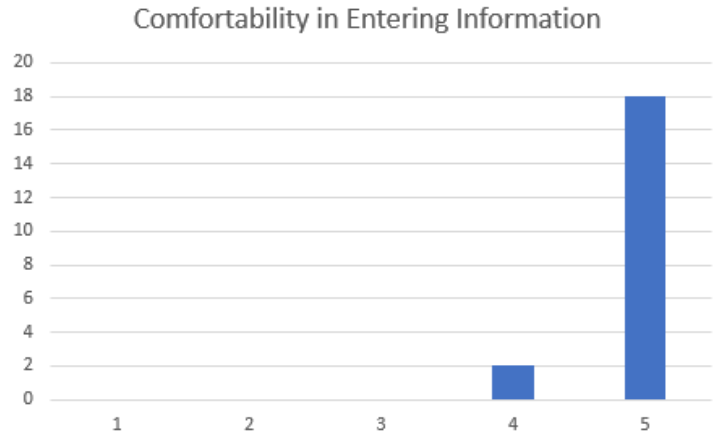

Figure 7: Results of Question 6 of the Survey

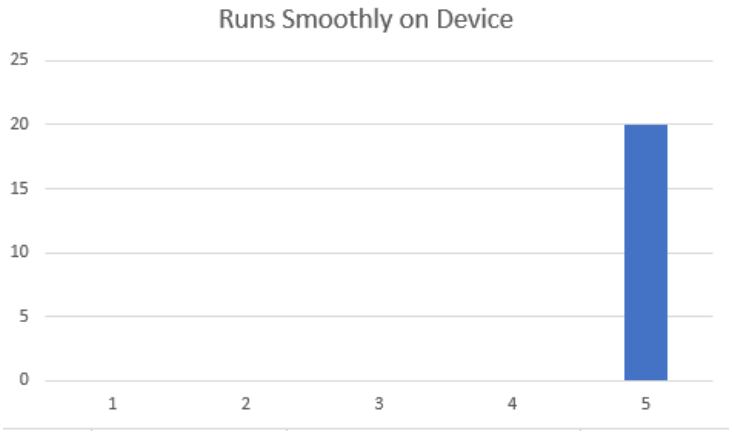

Figure 8: Results of Question 8 of the Survey

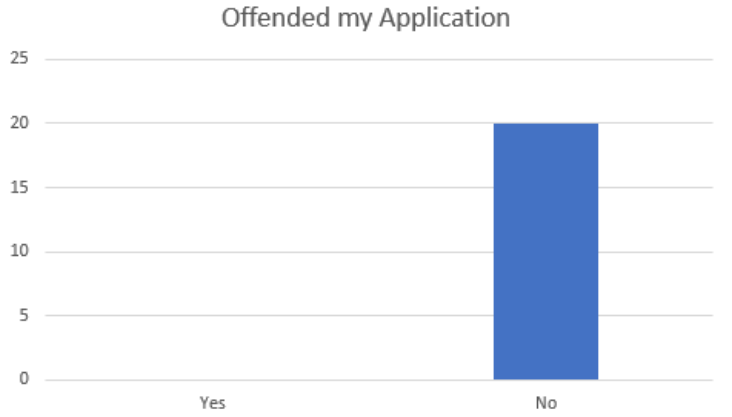

Figure 9: Results of Question 9 of the Survey

Table 2: **Trace to Funcitonal Requirements**

TestID	**Functional Requirements ID**
inputTest-id1	Functional Requirement 1 and 4
inputTest-id2	Functional Requirement 3 and 4
inputTest-id3	Functional Requirement 1 and 4
inputTest-id4	Functional Requirement 7
inputTest-id5	Functional Requirement 10
outputTest-id1.0.2	Functional Requirement 2 and 4
outputTest-id2	Functional Requirement 2 and 4
outputTest-id3	Functional Requirement 9
outputTest-id4	Functional Requirement 8

Table 3: **Trace to Non Funcitonal Requirements**

TestID	**Non Functional Requirements ID**
Survey Question 1	usabilityTest-id1
Survey Question 2	lookTest-id1, usabilityTest-id1
Survey Question 3	lookTest-id1
Survey Question 4	performance-Test-id1
Survey Question 5	usabilityTest-id2, performance-Test-id2
Survey Question 7	usabilityTest-id2, performance-Test-id2
Survey Question 6	securityTest-id1
Survey Question 8	usabilityTest-id1
Survey Question 9	legalTest-id1

Table 4: **Modular Trace to Functional Requirements**

TestID	**Functional Requirements ID**
inputTest-id1	Module 3
inputTest-id2	Module 3
inputTest-id3	Module 3
inputTest-id4	Module 17
inputTest-id5	Module 12 and 4
outputTest-id1.0.2	Module 9 and 10
outputTest-id2	Module 9 and 10
outputTest-id3	Module 12 and 4
outputTest-id4	Module 17, 12 and 4