Michael du Toit

2193161

BudgieCoin

Release 2 Documentation

**Contents**

[Conceptual Framework: 1](#_Toc72670341)

[Description of Improvements and Components: 1](#_Toc72670342)

[Features and Programming Concepts Applied: 2](#_Toc72670343)

[Constraints: 2](#_Toc72670344)

[Debugging: 3](#_Toc72670345)

[Espresso Testing: 5](#_Toc72670346)

[Application Screenshots: 8](#_Toc72670347)

# Conceptual Framework:

*Conceptual framework outlining the scope of this assignment.*

*If it is a new project, what functions have you included to meet the learning outcomes?*

*If you choose to build upon your Assignment 1, specify the improvements on this version.*

Assignment 2 built upon the BudgieCoin application that I started developing in Assignment 1.

For the 2nd release of the application, the scope of Assignment 2 was primarily the inclusion of an SQLite Database to store data, alongside its related CRUD functionality, and Expresso testing.

Alongside these two main requirements, the rest of the scope of Assignment 2 was to finish implementing functionality that was started in Assignment 1 that required a database to be finished, such as the View Transactions and View Accounts activities, who both use data from the database to display in custom list views, along with custom adapters for them.

The login credentials for the application are still admin (username) and 4321 (pin).

# Description of Improvements and Components:

*Descriptions of the components (for the new topic) or improvements (if you’re building upon your assignment 1) that you created.*

A key item that was implemented was a SQLite Database for the application. The database code was implemented in the *DBHandler* class, which manages the creation and interactions with the database via well-defined methods that the other code can interact with. This helped centralize all the database code and declarations.

After having implemented the SQLite Database, I finished implementing View Transactions (*ViewTransactionActivity*) and View Account Balances (*AccountBalancesActivity*). These two Activities get data from the database by using the *DBHandler* class and places them into a list view. This was accomplished by implementing two custom adaptors (*TransactionViewAdapter* and *AccountsBalancesAdaptor*). The items in these list views also have onClick listeners attached to them to open the selected item into a new activity which is populated with the data of the selected item.

Initially adding a transaction was implemented as a single activity, however after some reconsideration I changed this into fragments. After some further consideration I refactored the *TransactionExpenseFragment* and *TransactionIncomeFragment* to inherit from an abstract *TransactionBaseFragment* due to them sharing a lot of similiar code, with only the type of transaction being processed being the primary difference. Additionally, they reuse the same XML layout since the only difference between an income and expense is managed in the Java code.

Additionally, I implemented a very similar structure for the *TransactionUpdateIncomeFragment* and *TransactionUpdateExpenseFragment* classes to inherit from a base class (*TransactionUpdateBaseFragment*) and reuse their XML layout.

I also implemented confirmation dialogs that prompt the user when they attempt to delete an Account or Transaction to confirm their action before proceeding.

Finally, I implemented Espresso testing for the application, which can be found in the *app/src/androidTest* folder.

# Features and Programming Concepts Applied:

*List of features and programming concepts applied in your design.*

The Object-Oriented concepts of Inheritance and Abstraction was applied to the project, with the implementation of some “base” classes that implemented all the shared code that other classes would require. Examples of this are the *TransactionIncomeFragment* and *TransactionExpenseFragment* who inherit from the *TransactionBaseFragment*, and similarly the *TransactionUpdateIncomeFragment* and *TransactionUpdateExpenseFragment*, who inherit from the *TransactionUpdateBaseFragment*.

I also utilized an Object-Relational Mapper (ORM) type approach to interacting with the database.

With my ORM type approach, each table in the database have a corresponding Java classes who’s field model the table columns of the table. Using the *DBHandler* class implementation, data from the database was stored in the *Transaction* and *Account* class objects (from the Transactions and Accounts tables respectively). All the activities and classes then used the *DBHandler* class along with the class objects to interact with the database.

# Constraints:

**File Naming:**

As the number of files within the project increased, it started becoming challenging to give each file a short, but still descriptive name as to what that file does. Some of the files ended up with very similar names. To resolve this issue, I had to start using longer multiple word names to describe files which made it easier again to identify the purpose of files.

**Date & Time:**

Date and Time stamps can sometimes be a challenge to work within a programming environment.

Initially I implemented the date and time for transactions as strings in the database, whoever when it came to implementing the Update functionality for these transactions, it became problematic with getting values to set the date and time picker.

To resolve this issue, I will likely need to update the database implementation and re-write sections of the code for the transaction-related activities, which I’ll have to do in Final Project due to not having enough time to implement this before the Assignment 2 deadline.

**Inheritance and Abstraction:**

Some of my classes shared a large amount of code (such as *TransacationUpdateIncomeFragment* and *TransactionUpdateExpenseFragment*) with only minor differences between them. In this case I abstracted the shared code into a base class (*TransactionUpdateBaseFragment*) which the other classes could then extend from and override specific methods or implement them as required.

# Debugging:

A lot of debugging was accomplished with the Log functionality. Additionally, blocks of code that are likely to fail have been wrapped in try/catch blocks that write a Log message if it fails. Some examples of my usage of Logging to debug my code are shown below.

Text

Description automatically generated

Figure 1 - AddAccountActivity's createAccount method

Figure 1 above shows the try-catch block for the createAccount method in the AddAccountActivity which Logs an exception message if it fails.

Text

Description automatically generated

Figure 2 - TransactionUpdateActivity's onCreate method

Figure 2 above shows debugging the value of the transaction that has been passed with the intent to ensure it is correct.

Text

Description automatically generated

Figure 3 - TransactionAdaptor's constructor

Figure 3 shows the debugging log error for if the TransactionAdapter’s try/catch fails.

Text

Description automatically generated

Figure 4 - TransactionUpdateIncomeFragment's UpdateTransaction method

Figure 4 shows the debugging within the try/catch for the UpdateTransaction method to check up to where in the statement the code was running.

# Espresso Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Class | Test Case / Method: | Test Description: | Status | Image |
| AccountBalancesActivityTest | checkAllViewElements Present | Checks that all the UI view elements that should be visible are displayed. | Pass | Figure 5 |
| AddAccount  ActivityTest | checkAllViewElements Present | Checks that all the UI view elements that should be visible are displayed. | Pass | Figure 6 |
| AddAccount  ActivityTest | testAccountCreation | Enters a data for creating an account, clicks the “Create” button and then checks that the account has been created. | Pass | Figure 6 |
| AddAccount  ActivityTest | testAccountCancellation | Finds the Cancel button on the Create Account view, then clicks cancel and checks that the app returns to the Main Activity. | Pass | Figure 6 |
| LoginActivityTest | checkAllViewElements Present | Checks that all the UI elements that should be visible are displayed. | Pass | Figure 7 |
| LoginActivityTest | testLoginInvalidLogin | Enters invalid login credentials into the input fields, clicks the “Login with Pin” button and checks that the login error message is displayed. | Pass | Figure 7 |
| LoginActivityTest | testLoginValidLogin | Enters valid login credentials into the input fields, clicks the “Login with Pin” button and checks that the app proceeds to the Main Activity. | Pass | Figure 7 |
| MainActivityTest | checkAllViewElements Present | Checks that all the UI view elements that should be visible are displayed. | Pass | Figure 8 |
| MainActivityTest | testAddTransaction  Button | Checks that when the “Add Transaction” button is clicked that it navigates to the Transaction Activity | Pass | Figure 8 |
| MainActivityTest | testViewTransactions Button | Checks that when the “View Transactions” button is clicked that it navigates to the View Transactions Activity | Pass | Figure 8 |
| MainActivityTest | testViewAccount Balances | Checks that when the “View Account Balances” button is clicked that it navigates to the Account Balances Activity | Pass | Figure 8 |
| ViewAccountActivityTest | testAllViewElements Present | Checks that all the UI view elements that should be visible are displayed. | Pass | Figure 9 |
| ViewTransactionsActivityTest | testAllViewElements Present | Checks that all the UI view elements that should be visible are displayed. | Pass | Figure 10 |

A screenshot of a computer

Description automatically generated

Figure 5 - AccountBalancesActivityTest Results Screenshot  
  
A screenshot of a computer

Description automatically generated  
Figure 6 - AddAccountActivityTest Results Screenshot  
  
A screenshot of a computer

Description automatically generated

Figure 7 - LoginActivityTest Results Screenshot

A screenshot of a computer

Description automatically generated

Figure 8 - MainActivityTest Results Screenshot

A screenshot of a computer

Description automatically generated

Figure 9 - ViewAccountActivityTest Results Screenshot

A screenshot of a computer

Description automatically generated

Figure 10 - ViewTransactionsActivityTest Results Screenshot

# Application Screenshots:

*Screenshots of the final app components.*