CSCI 5708

ADVANCED TOPICS IN MOBILE COMPUTING

MIDTERM

Date: March 12, 2024

Full Name: Vishaka Vinod

Banner ID: B00955686

Dal Email: vs235403@dal.ca

Gitlab: https://git.cs.dal.ca/vvinod/csci5708_w24_b00955686_vishaka_vinod

Project Description

The objective of the assigned midterm project was to develop an intuitive expense/income logging application to assist users in efficiently managing their financial transactions. The aim of the midterm was to understand and integrate key concepts covered during the lectures and labs. It includes topics like RecyclerViews, CardViews, Fragments, Shared Preferences, and database integration using ROOM or SQLite. The project helped to provide a practical implementation of these concepts while enhancing understanding and proficiency in Android development.

Activity One:
This activity comprises of both the login and sign-up functionalities. It uses separate fragments within a single activity, and the interface renders each fragment based on user interactions. The login fragment provides user authentication using input fields for username and password, along with a login button for validation. It also contains a sign-up button for users to navigate to the sign-up page to create a new account. The sign-up fragment helps in user registration. It has input fields for profile name, username, and password. The signup button helps users to complete their registration.

Activity Two:

This activity allows users to manage and track their spending habits efficiently. It uses RecyclerViews for displaying an overview of the user's income and expenses. This activity also includes buttons for logging out and adding new expenses or income. It displays the user profile name at the top of the screen. The expenses data is integrated with Room or SQLite databases for efficient management and retrieval.

Activity Three:

This activity is invoked when users wish to either create a new expense/income entry or modify an existing one. Data management is done using databases to maintain consistency with the overview displayed in Activity Two.

Technical Details & Code Snippets:

This project uses the Android Studio IDE and the code is written using Kotlin. The database used for CRUD operations is the SQLite database. All activities are declared in the `AndroidManifest.xml` file.

Activity One

Activity one handles the user login and signup. When the user first opens the application, the *ActivityOne.kt* file loads the *LoginFragment.kt* using a support fragment manager. Here, the users input their credentials. If validation is successful, *ActivityTwo.kt* is started, else a toast with the appropriate error massage will be displayed. The login fragment is rendered by *activity_login_fragment.xml*. If the user does not have an account and wants to sign up, the user will click the sign-up button and the *SignupFragment.kt* is displayed. The signup fragment layout is present in *activity_signup_fragment.xml*. The entire layout and structure of *ActivityOne.kt*, including the login and signup fragments, are defined in *activity_one.xml*. The *Users.kt* in the model for how user data will be stored in the shared preferences file.

```kotlin
class ActivityOne : AppCompatActivity() {
    Vishaka Vinod *
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_one)

        // Setting initial screen fragment to the user login page
        val loginFragment : Fragment = LoginFragment()
        val loginFragmentTransaction : FragmentTransaction =
            supportFragmentManager.beginTransaction()
        loginFragmentTransaction.replace(
            R.id.activityOneFrameLayout,
            loginFragment
        ).commit()
```

Figure 1: Code in ActivityOne.kt for setting login fragment on user login [1]

```kotlin
// Method to save user details into a shared preference file using JSONs
Vishaka Vinod
private fun signUpUser(profileName: String, username: String, password: String) {

    val userInfo = Users(profileName, username, password)
    val sharedPreferences : SharedPreferences = getSharedPreferences( name: "Users", MODE_PRIVATE)
    val editor: SharedPreferences.Editor = sharedPreferences.edit()
    val userInfoJson = Gson().toJson(userInfo)

    editor.putString(username, userInfoJson)
    editor.apply()
}
```

Figure 2: Code in SignupFragment.kt for storing user data in a Shared Preference [2]

```kotlin
// If user selects the login button, an intent is sent to ActivityOne to display the LoginFragment
val logBtn: Button = view.findViewById(R.id.logIn)
logBtn.setOnClickListener { it: View!
    val username: EditText = view.findViewById(R.id.userName)
    val password: EditText = view.findViewById(R.id.userPassword)

    // Ensures that user does not leave any fields blank
    if(username.text.isNotEmpty() && password.text.isNotEmpty()){
        val intent = Intent(view.context, ActivityOne::class.java)
        intent.putExtra( name: "logInMsg",  value: "Logging in user...")
        intent.putExtra( name: "logInUsername", username.text.toString())
        intent.putExtra( name: "logInPassword", password.text.toString())
        startActivity(intent)
    }
    else {
        // Display toast message if user leaves any input fields empty
        Toast.makeText(
            view.context,
            text: "Please don't leave any fields empty",
            Toast.LENGTH_SHORT
        ).show()
    }
```

Figure 3: Code in LoginFragment.kt for validating user input and credentials

Users.xml

```xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="vish@yahoo.com">{"password":"123","profileName":"vish","username":"vish@yahoo.com"}</string>
    <string name="mary@gmail.com">{"password":"123","profileName":"mary","username":"mary@gmail.com"}</string>
    <string name="siya@gmail.com">{"password":"123","profileName":"siya","username":"siya@gmail.com"}</string>
    <string name="vvinod@gmail.com">{"password":"qwerty","profileName":"vishaka","username":"vvinod@gmail.com"}</string>
    <string name="vish">{"password":"123","profileName":"vish","username":"vish"}</string>
</map>
```

Figure 4: The Users.xml file generated by Android Studio to store the shared preferences [2]

Activity Two

Activity two defines the layout for displaying the overview of the expense and income activities logged by the user. The *ActivityTwo.kt* file renders the user landing page, and its layout is defined in *activity_two.xml*. It contains a text view of the user's profile name, a logout button, and an add button. There is also a recycler view defined in the layout that recycles the cards within *activity_income_expense_cardview.xml*. The recycler view makes use of Adapters and CardViewHolders. The *DBHelper.kt* file helps connect this activity and its components to the SQLite database. This class performs the CRUD operations to allow users to add, delete and edit their expense tracking. The *Expenses.kt* in the model for how expense tracking data is stored.

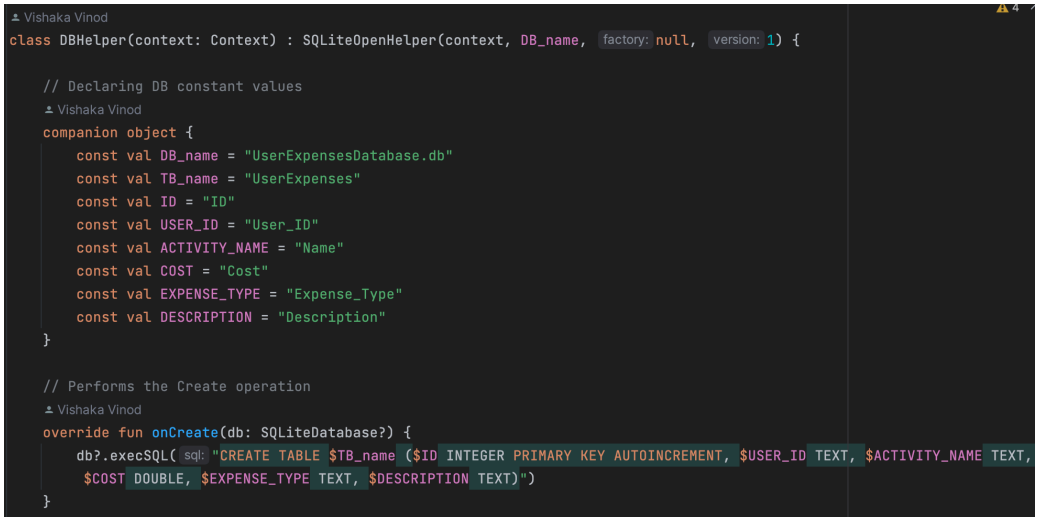

Figure 5: Code in DBHelper.kt to create a database and its corresponding tables [3]

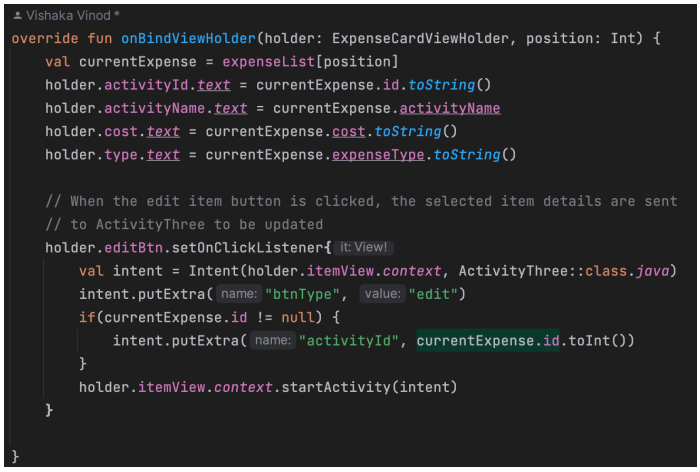

Figure 6: Code in ExpenseAdapter.kt that sets the current expense in the RecyclerView [3]

```kotlin
// Method to load data from SQLite database
Vishaka Vinod
private fun loadData(username: String){
    val db = dbHelper.readableDatabase
    val cursor = db.rawQuery( sql: "SELECT * FROM ${DBHelper.TB_name} WHERE ${DBHelper.USER_ID} = ?", arrayOf(username))
    val expenseList = ArrayList<Expenses>()

    while (cursor.moveToNext()) {
        val activityId = cursor.getInt(cursor.getColumnIndexOrThrow(DBHelper.ID))
        val activityName = cursor.getString(cursor.getColumnIndexOrThrow(DBHelper.ACTIVITY_NAME))
        val cost = cursor.getDouble(cursor.getColumnIndexOrThrow(DBHelper.COST))
        val type = cursor.getString(cursor.getColumnIndexOrThrow(DBHelper.EXPENSE_TYPE))
        val typeSymbol : String = if(type.lowercase() == "income"){
            "+"
        } else if(type.lowercase() == "expense"){
            "-"
        } else {
            ""
        }
        expenseList.add(Expenses(activityId, username, activityName, cost, typeSymbol, description: ""))
    }
    cursor.close()
    expenseAdapter = ExpenseAdapter(expenseList)
    expensesRecyclerView.adapter = expenseAdapter
    db.close()
}
```

Figure 7: Code in ActivityTwo.kt that loads the data from the SQLite database and renders it [4]

Activity Three

Activity three performs the CRUD operations using the SQLite database. The *ActivityThree.kt* renders the same *activity_three.xml* view in two different formats. The file contains input fields for activity name, amount, description, and type of activity (income or expense). The first format is rendered when the user wants to add a new entry. Here the page is loaded without any pre-set values in the input fields. The second format is when the user wants to edit an already existing item. Here, the page will be displayed with the pre-set values of the selected items in the input field. This layout also contains a save and delete button. The save button should checks if it is an existing entry and updates the database or inserts into the database. The delete button, clears the data from the database.

```kotlin
// Method to get item details by id
Vishaka Vinod *
private fun getItemById(itemId: Int): Expenses?{
    val db = dbHelper.readableDatabase
    val cursor = db.rawQuery( sql: "SELECT * FROM ${DBHelper.TB_name} " +
            "WHERE ${DBHelper.ID} = ?", arrayOf(itemId.toString())
    )

    var item: Expenses? = null

    while (cursor.moveToNext()) {
        val username = cursor.getString(cursor.getColumnIndexOrThrow(DBHelper.USER_ID))
        val activityName = cursor.getString(cursor.getColumnIndexOrThrow(DBHelper.ACTIVITY_NAME))
        val cost = cursor.getDouble(cursor.getColumnIndexOrThrow(DBHelper.COST))
        val type = cursor.getString(cursor.getColumnIndexOrThrow(DBHelper.EXPENSE_TYPE))
        val description = cursor.getString(cursor.getColumnIndexOrThrow(DBHelper.DESCRIPTION))

        item = Expenses(itemId, username, activityName, cost, type, description)
    }
    cursor.close()
    db.close()
    return item
}
```

Figure 6: Code in AndroidThree.kt that contains a method that gets data from the DB by Id [4]

```kotlin
// method that calls DBHelper to add an item
Vishaka Vinod
private fun addNewItem(username: String, activityName: String, cost: Double, expenseType: String, description: String){
    dbHelper.addExpense(username, activityName, cost, expenseType, description)
    val intent = Intent( packageContext: this, ActivityTwo::class.java)
    intent.putExtra( name: "currentUser", username)
    startActivity(intent)
}

// method that calls DBHelper to edit an item
Vishaka Vinod
private fun editItem(id: Int, currentItem: Expenses?){
    var user = ""
    if(currentItem != null){
        dbHelper.updateExpense(id, currentItem.activityName, currentItem.cost, currentItem.description, currentItem.expen
        user = currentItem.username.toString()
    }
    val intent = Intent( packageContext: this, ActivityTwo::class.java)
    intent.putExtra( name: "currentUser", user)
    startActivity(intent)
}

// method that calls DBHelper to delete an item
Vishaka Vinod
private fun deleteItem(id: Int, currentUser: String?){
    if(currentUser != null){
        dbHelper.deleteExpense(id)
        val intent = Intent( packageContext: this, ActivityTwo::class.java)
        intent.putExtra( name: "currentUser", currentUser)
        startActivity(intent)
    } else {
        finish()
    }
```

Figure 7: Code in AndroidThree.kt that performs the CRUD operations using DBHelper.kt [4]

Results and Screenshots

The screenshots shown below are that of the working application.

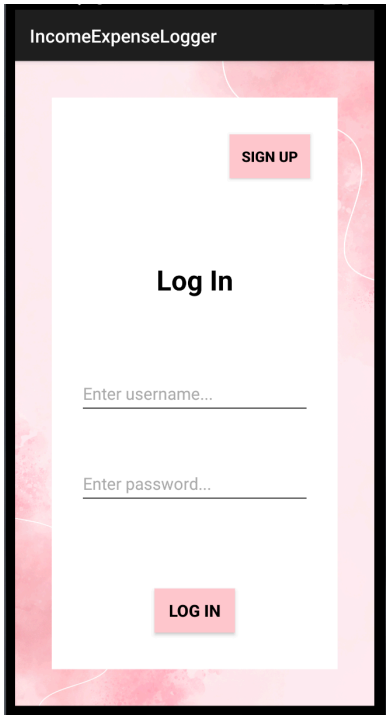

Figure 8: Login Page

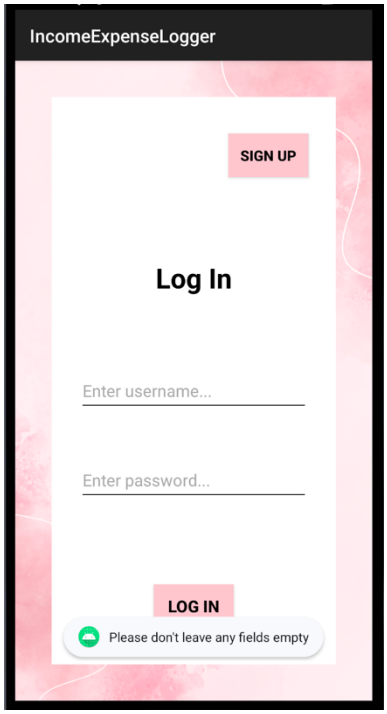

Figure 11: Login Page validation for empty inputs

Figure 12: Login Page validation for invalid username

Figures 10 shows the login page which is the first page that is loaded when the user logs into the application. Figures 11 and 12 show login page validations based on user input. If the user does not put any input the “Please don’t leave any fields empty” toast message is displayed. Similarly, if the user enters a wrong username or password a “Invalid username…” or “Invalid password…” toast message is displayed. These validations are also present in the Signup page.

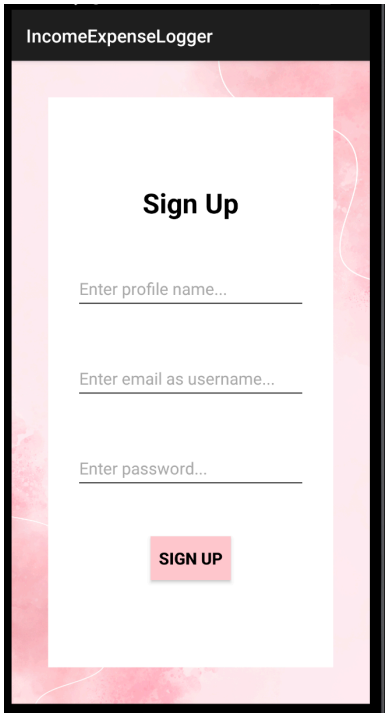

Figure 13: Signup Page

Figure 14: Creating a new user

Figure 15: Logging in as new user

Figure 16: Empty Recycler View for user

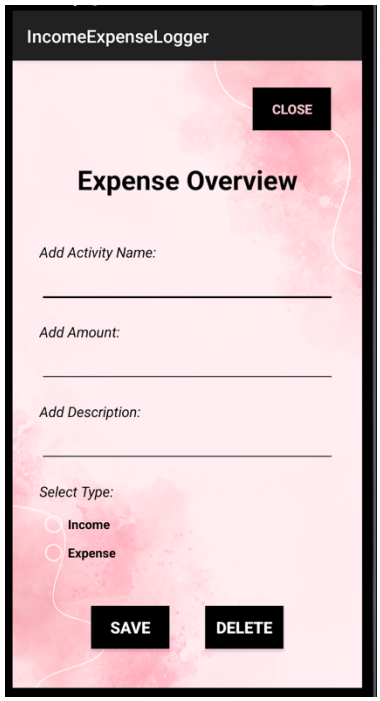

Figure 17: View to add new entries

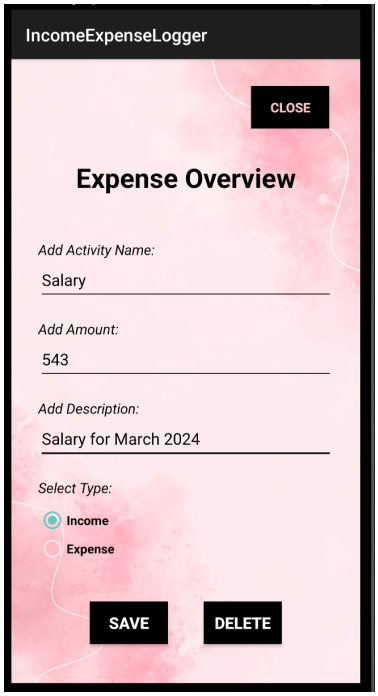

Figure 18: Adding expense and income entries

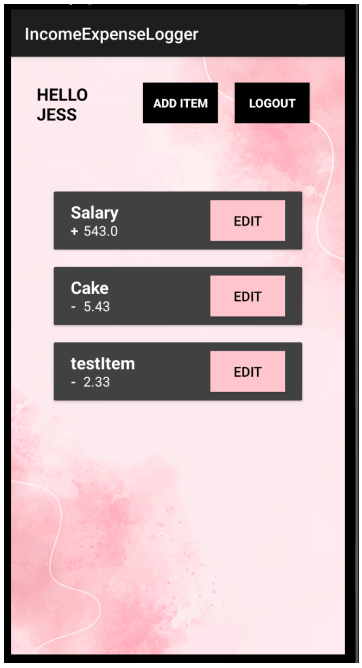

Figure 19: RecyclerView of user activity items

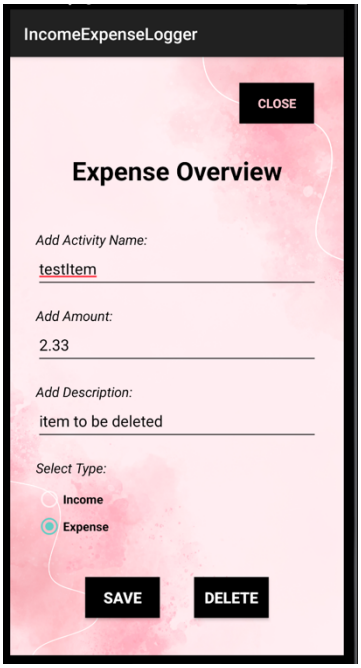

Figure 20: Editing an item

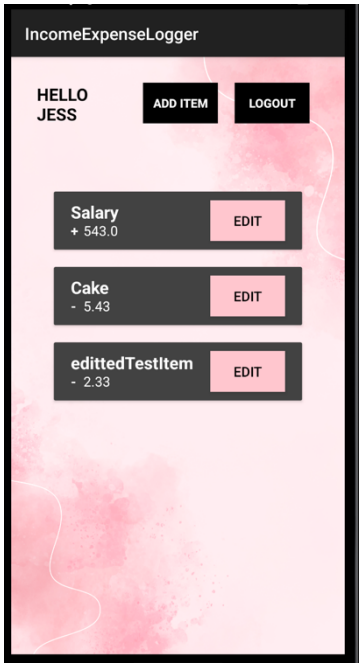

Figure 21: Updated data displayed

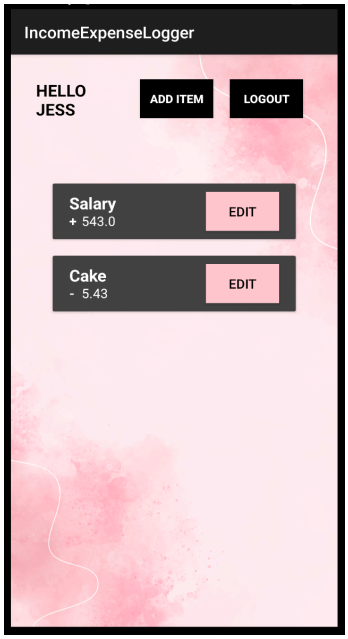

Figure 22: Deleted item from database and updated view

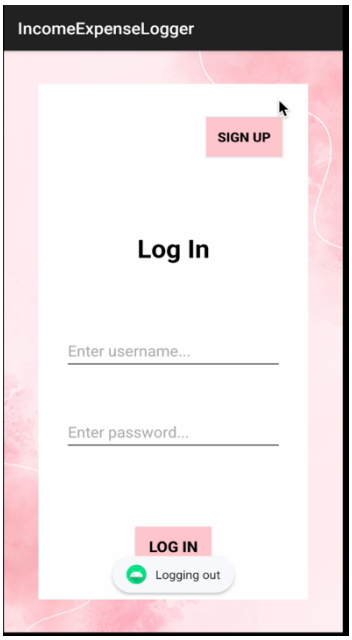

Figure 23: Logout user

Data Flow Documentation

The first flowchart diagram represents the functionality of activity one.

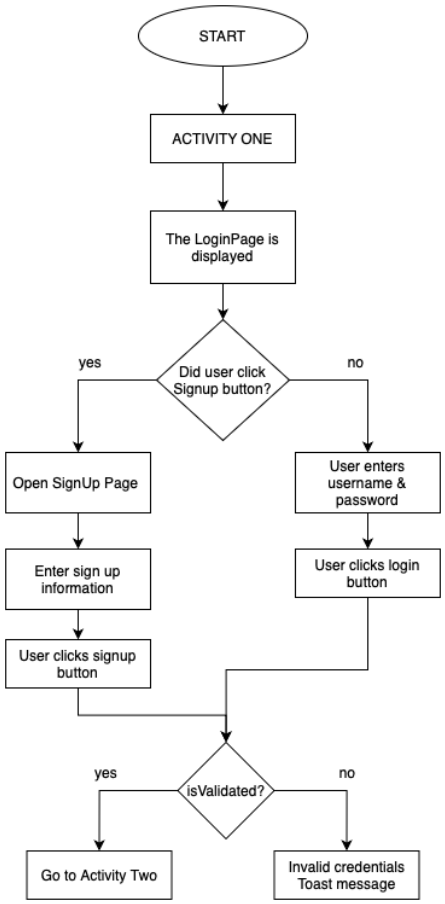

Figure 24: Flowchart diagram for Activity one [5]

The second flowchart diagram represents the functionality of activity two.

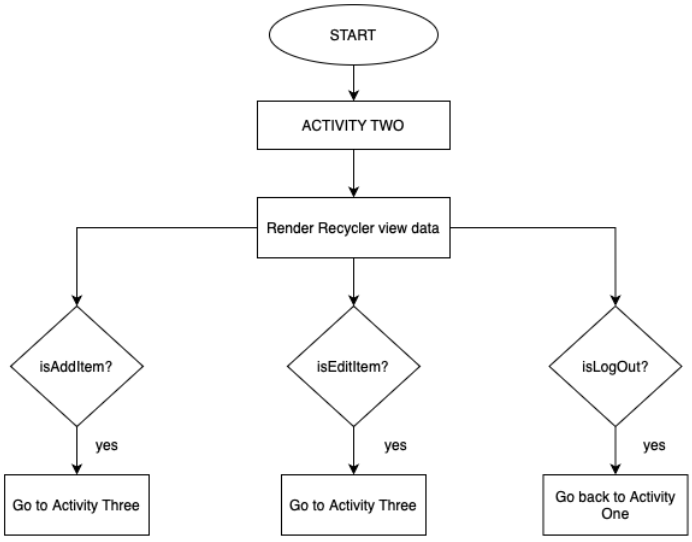

Figure 25: Flowchart diagram for Activity two [5]

The third flowchart diagram represents the functionality of activity three

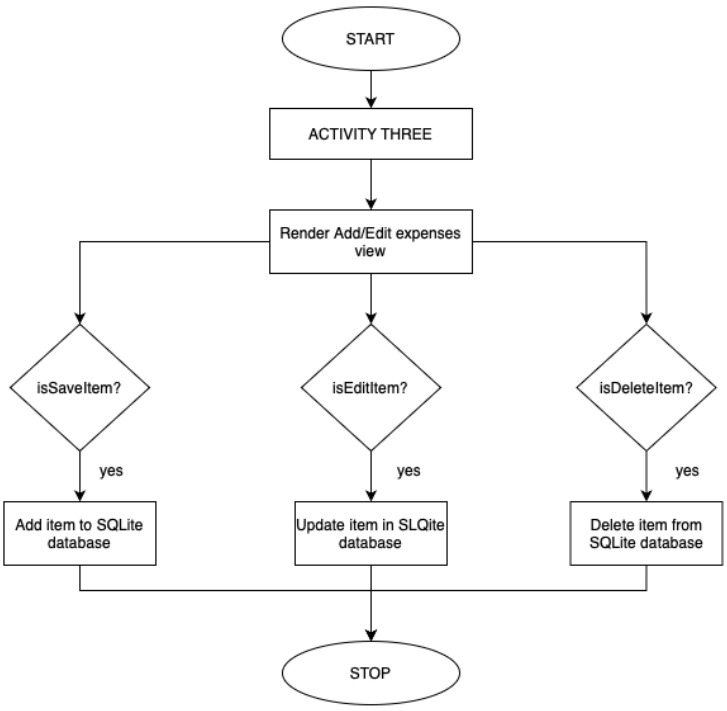

Figure 26: Flowchart diagram for Activity three [5]

Interpretations and Assumptions

Unique Username Assumption: the username entered by the user during login and registration will be unique, like an email address. This is essential for the validation logic during the sign-up and login process to ensure that users are authenticated.

Error Handling Assumption: Due to time constraints, comprehensive error handling for edge cases was not prioritized. However, basic error handling has been implemented for scenarios like invalid user inputs. Error messages are informative and user-friendly.

Background Image: The background image used throughout the application is designed using the Canva designing website for a seamless and enjoyable user experience [6].

Challenges & Learning Points

It was a challenge to completely implement the entire application and write a detailed report on the same within twenty-four hours. However, I was able to rise to the occasion and meet all expectations.

Conclusion

As a takeaway from this assignment, I plan to work on the application at my own pace and develop it further to improve my understanding and skills in android development.

References

[1] "Fragment transactions," *Android Developers*. [Online]. Available: https://developer.android.com/guide/fragments/transactions. [Accessed: March 12, 2023].

[2] M.Y. Suraki, "Android Data and File Storage," *Brightspace.com.* [Online]. Available: https://dal.brightspace.com/d2l/le/content/311805/viewContent/4138757/View. [Accessed: March 12, 2023].

[3] M.Y. Suraki, "Adapters," *Brightspace.com.* [Online]. Available: https://dal.brightspace.com/d2l/le/content/311805/viewContent/4138754/View. [Accessed: March 12, 2023].

[4] M.Y. Suraki, "Week 7 Lab Recording," *dal.sharepoint.com,* March 04, 2024. [Online]. Available: https://dalu.sharepoint.com/teams/MobileComputing-Winter2024-LabSessionRecordings/_layouts/15/stream.aspx?id=%2Fteams%2FMobileComputing%2DWinter2024%2DLabSessionRecordings%2FShared%20Documents%2FMonday%5F2024%2D03%2D04%2Emp4&ga=1&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview. [Accessed: March 12, 2023].

[5] "Security-first diagramming for teams," *draw.io*. [Online]. Available: https://www.drawio.com. [Accessed: March 12, 2023].

[6] *Canva.com*. [Online]. Available: https://www.canva.com. [Accessed: March 12, 2023].