IT2010 – Mobile Application Development
BSc (Hons) in Information Technology
2nd Year
Faculty of Computing
SLIIT

2023 – Assessment 02
Phase 2 Report

Description	Student ID	Name
Group Leader	IT21374524	Waseek M.L.
Member 2	IT21355196	Kalpajith K.L.S.
Member 3	IT21377358	Hanshani S.G.H.S
Member 4	IT21189876	Karunarathne H.B.T.N.

Application Topic	Money Management Mobile Application. (PHASE_02)
Group Name	IT_WD_G_02.02

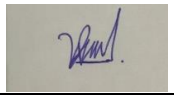
Project Declaration

We, the members of IT_WD_G_02.02, hereby declare that our group project is entirely authentic and original. We have conducted thorough research and analysis to ensure that our work is not plagiarized or copied from any other sources.

We have followed all guidelines provided by our LIC and have complied with all ethical and academic standards.

We take full responsibility for the authenticity of our work and understand the implications of academic dishonesty. We have worked collaboratively to produce this project, and each member has contributed to the best of their abilities.

We hereby affirm that our project represents our honest effort and commitment to academic integrity, and we take pride in presenting it as our own.



Group Leader (Signature)
Waseek M.L.

Git Repository

https://github.com/Shashin99/Budgetary.git

LinkedIn Link

https://www.linkedin.com/posts/waseek-lareef-981267217_money-uiux-frontend-activity-7060351384675647488-qrHH?utm_source=share&utm_medium=member_desktop

Description

Budgetary is a money management mobile application implemented to help the individuals and the households for better management of their financial activities during this economic crisis to make them the practice of accurate spending.

This application mainly focuses on four features such as income flow, expenses, bill categorization and budget planning and saving goals. Budgetary is a minimal and a reliable application where users can view their current balance and the features on the dashboard.

The "Income" function allows to store their cash flows during the month and the application calculates the total income concurrently. The users can plan their monthly cost of operations categorically (such as monthly food cost, transportation, household, education, etc.) regards to their income.

The users can add their bills, bill details, and make the payments via the application. They can add the daily expenses made via online banking or cash as this acts as a daily cash diary. Then the system displays the current available balance after adding the expenditure.

As a result, this application allows the user to manage the expenses and improve the savings to help the individuals workout through the crisis by displaying the total expenses in their profile.

Functions

IT21377358 - Profile management and Income management

The profile management function allows the users to create a user account, login and manage the profile as individual. The registered users can login to the system with the login credentials. The users can add their name, contact number, Goal and create the account by the email address and password details to their profile. The users store their monthly income flows descriptively and display the total income. The total income is required for the expenses, budgeting features of the app. The user can manage (edit, delete) the monthly income by clicking the income in the view page.

IT21189876 - Budget management

Budget management is one of the main functions of the system. It helps users to use their money sparingly and systematically. Users can create their budget plan related to their income. The app gives the user some already-implemented budgeting categories, accordingly it can be updated or can create a new category one by clicking "Add new plan". If we want, a category plan can be added to the budget plan and deleted if unnecessary.

IT21355196 - **Bill management**

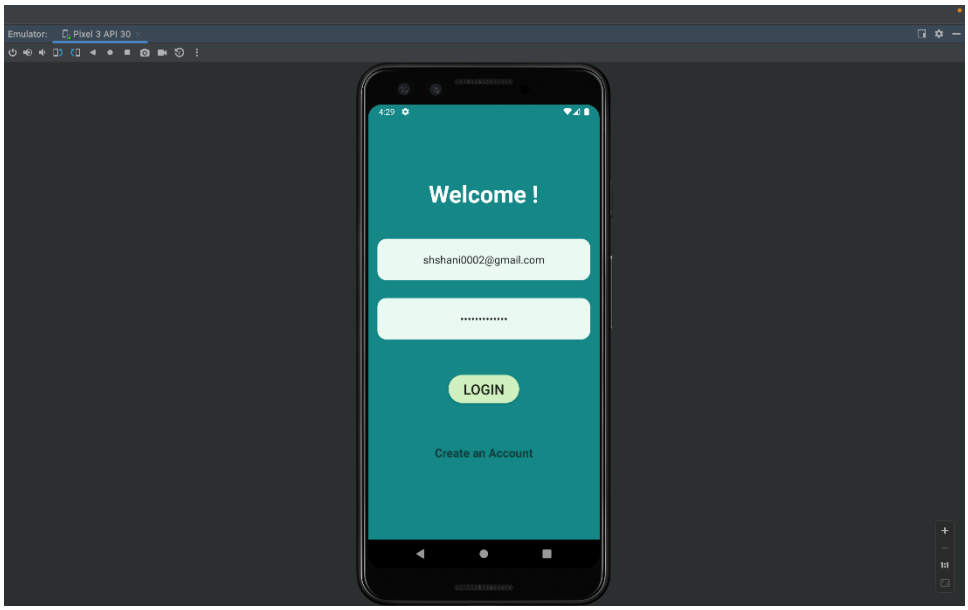
Bill Management function allows users to add the monthly payable bill under categories (electricity, water, utilities…). User can save the bill details. Users can update the details at the point of making the payment. After making the payment, a successful message is popped and the transaction details are saved in the "transaction" interface, the customer can view and delete the transaction details. But the users cannot edit the transaction details. The entire process related to Bill Management is user friendly while pop messages are displayed related to each step.

IT21374524 - **Expense management**

The expenses management function is used to maintain user's expenses, and this acts as a cash diary. For that user must be registered to the system. User can Add expenses, view expenses, update expenses and delete expenses. While adding expenses, users want to choose a category, add the details to the expenses and calculate the current balance simultaneously. The user can add the expenses made through cash. If the user deletes or updates an expense the Balance is updated.

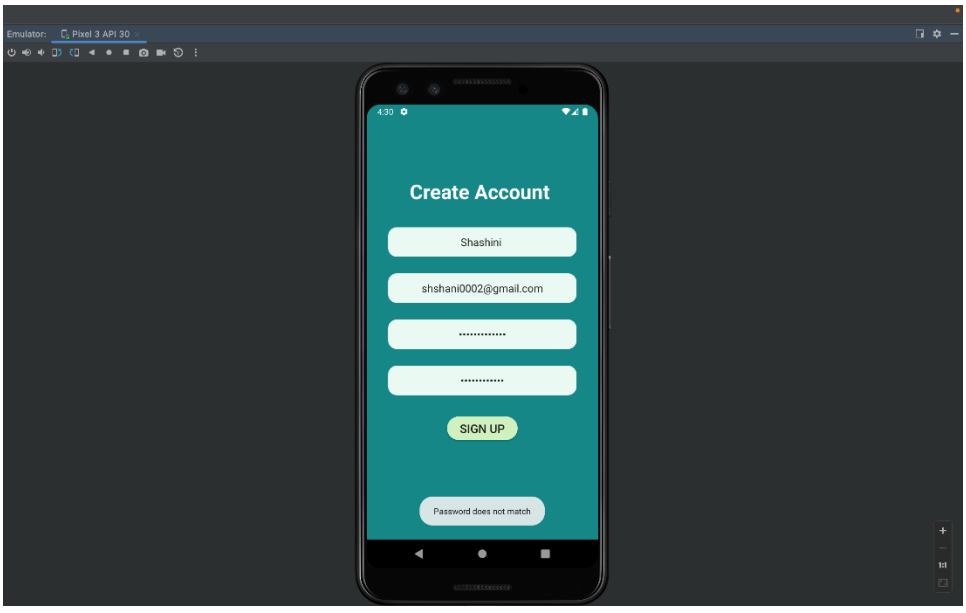
Common Interfaces

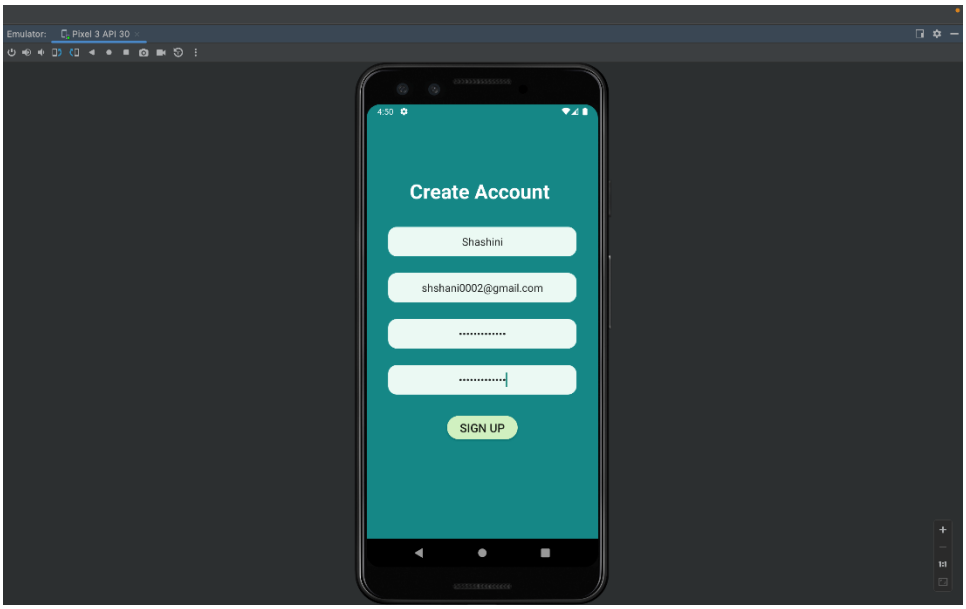
Login Page: The user should provide a valid email and a password to use this application



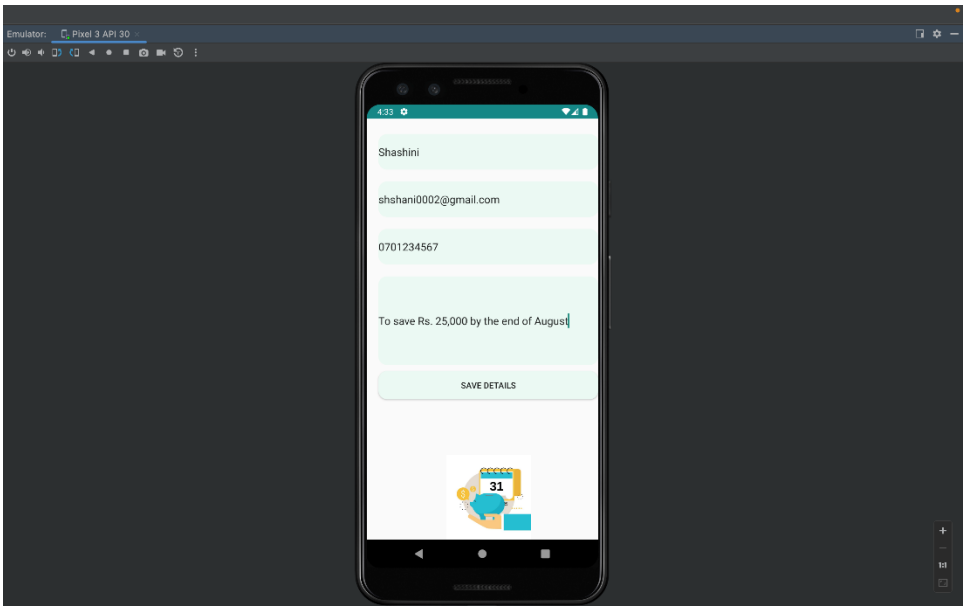
If the user is not a registered user, they can create their own account by entering the Username, email, password and they should confirm the password

Create Account Page:





Inserting User Details: After the account is created successfully they can insert the user details and set a goal where this can be read in their user profile



Member Wise Descriptions

IT21377358 - Profile management and Income management

INCOME MANAGEMENT

Instrumented Testing –

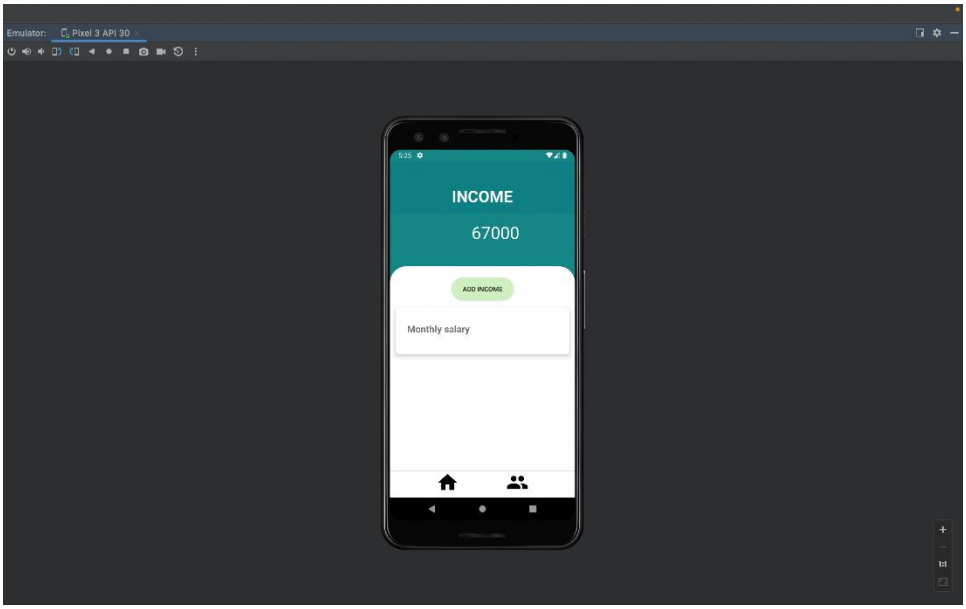
Check whether When clicking Income button that page redirect to the all income page.

```
package com.example.assignmentmad

import androidx.test.core.app.ActivityScenario
import androidx.test.espresso.Espresso
import androidx.test.espresso.action.ViewActions
import androidx.test.espresso.assertion.ViewAssertions
import androidx.test.espresso.intent.Intents
import androidx.test.espresso.intent.matcher.IntentMatchers
import androidx.test.espresso.matcher.ViewMatchers
import androidx.test.ext.junit.rules.ActivityScenarioRule
import androidx.test.ext.junit.runners.AndroidJUnit4
import androidx.test.platform.app.InstrumentationRegistry
import com.example.budgetapp.activities.FetchingBill
import com.example.budgetapp.activities.MainActivity
import org.junit.*
import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)
class ViewBillTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        Assert.assertEquals("com.example.appsquad", appContext.packageName)
    }

    @get:Rule
    val activityScenarioRule = ActivityScenarioRule(MainActivity::class.java)
    private lateinit var activityScenario: ActivityScenario<MainActivity>
    @Before
    fun setUp() {
        activityScenario = activityScenarioRule.scenario
        activityScenario.onActivity { activity ->
            // Do any setup you need for your activity here
        }
    }

    @Before
    fun setUp() {
```

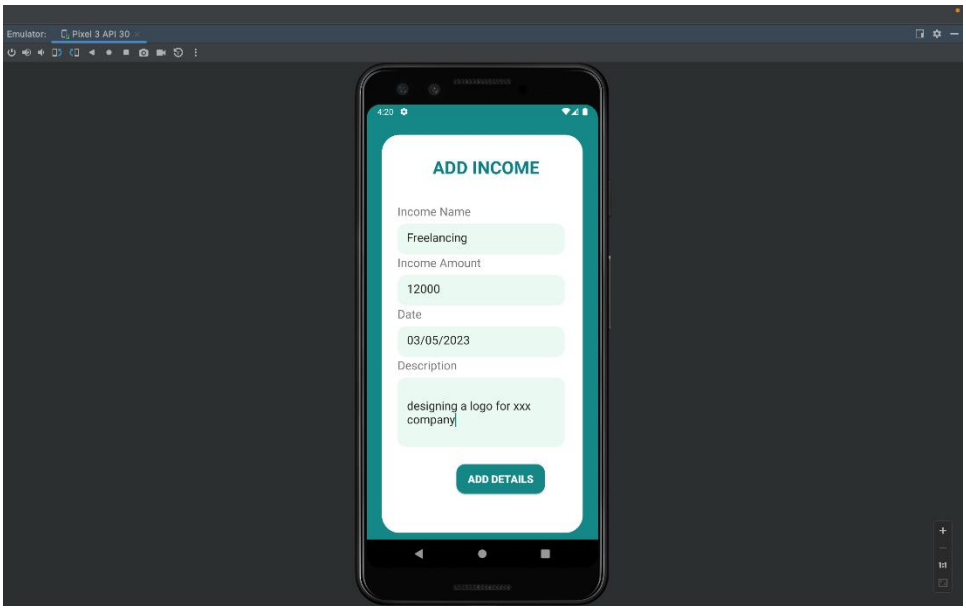
```
        activityScenario = activityRule.activity
        activityScenario.onActivity { activity ->
            // Do any setup you need for your activity here
        }
    }

    @Test
    fun testButton(){


        Intents.init()
        Espresso.onView(ViewMatchers.withId(R.id.editIncomebtn))
            .check(ViewAssertions.matches(ViewMatchers.isDisplayed()))

Espresso.onView(ViewMatchers.withId(R.id.editIncomebtn)).perform(ViewActions.click());


        Intents.intended(IntentMatchers.hasComponent(IncomeModel::class.java.name))
    }
    @After
    fun tearDown() {
        activityScenario.close()
        }
}
```

The user can manage the income by clicking the “INCOME” button on the home page. The user gets directed to the income page. The user can add, View, update or delete the income details.

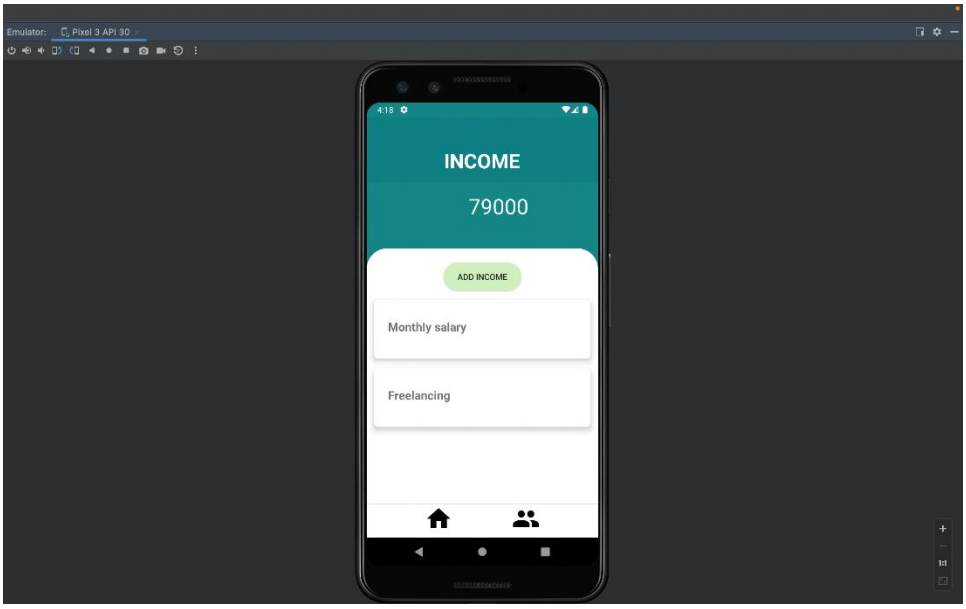
Income Page: The currently added income records are displayed with their names and the total income is displayed



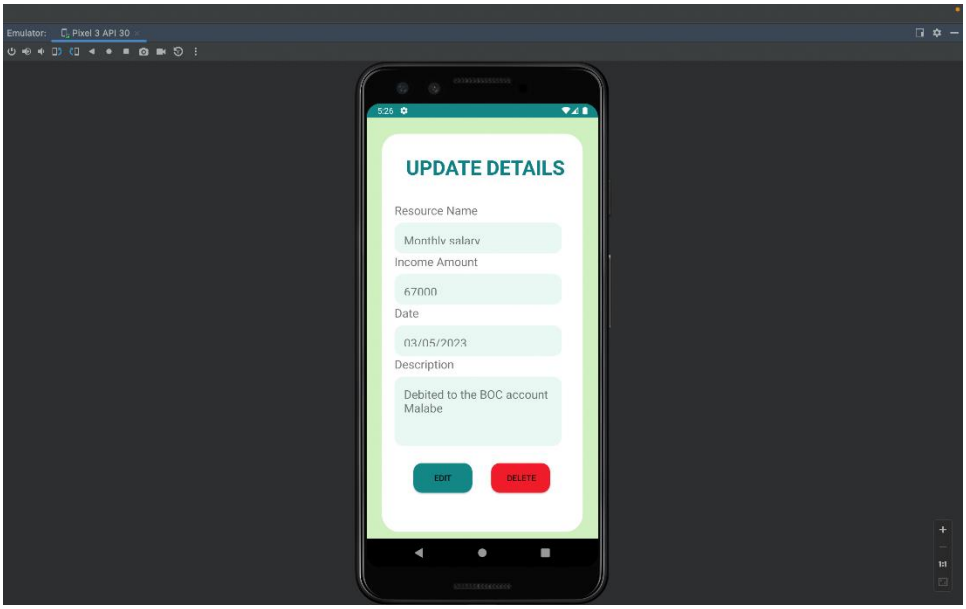
Adding Income: When the user needs to add a new income, by clicking the “Add Income” button, they get navigated to the “ADD INCOME” page and can add a new income record



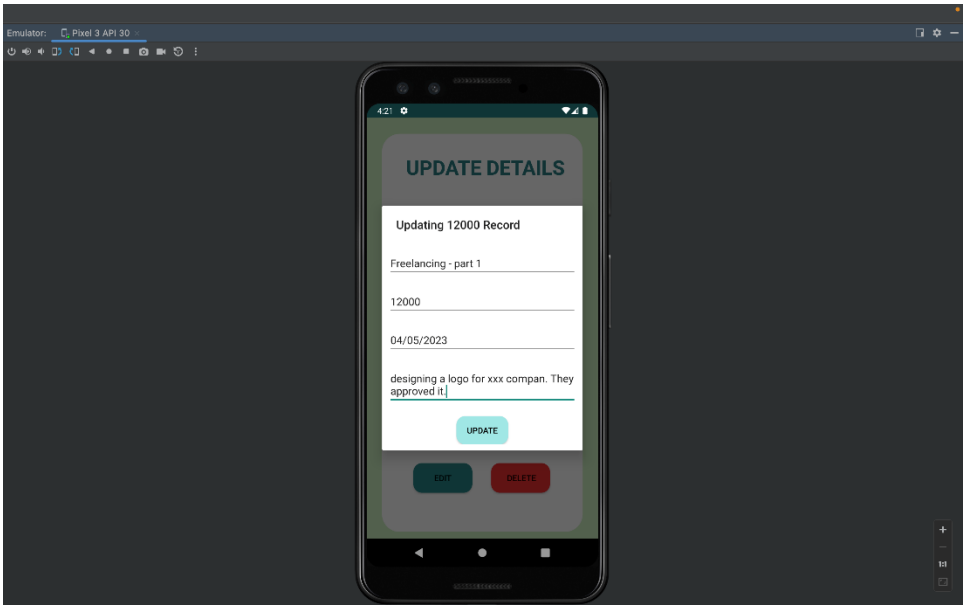
Then it is displayed in the Income page

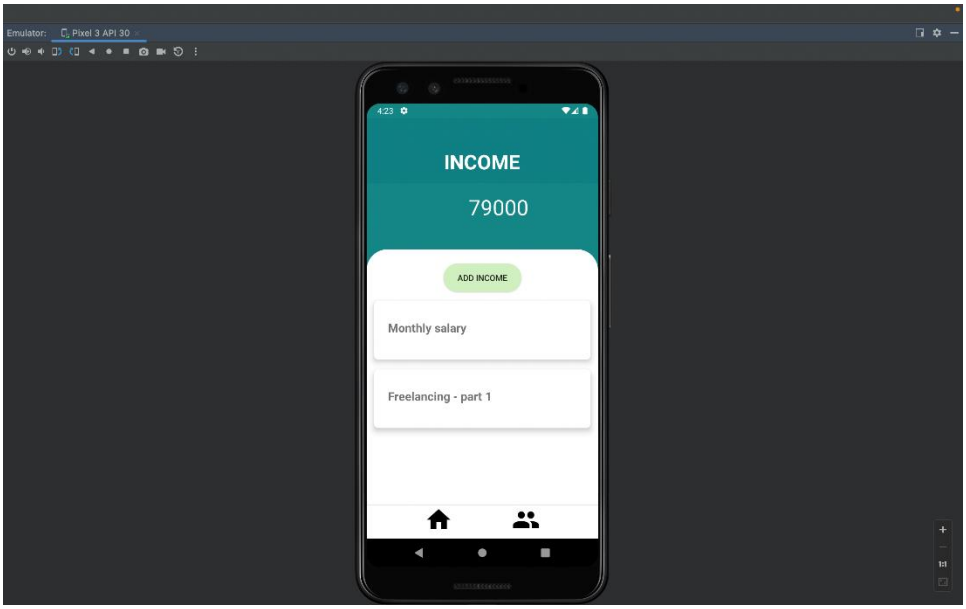


Viewing an income record: The user can view the record when clicking the text of the income record

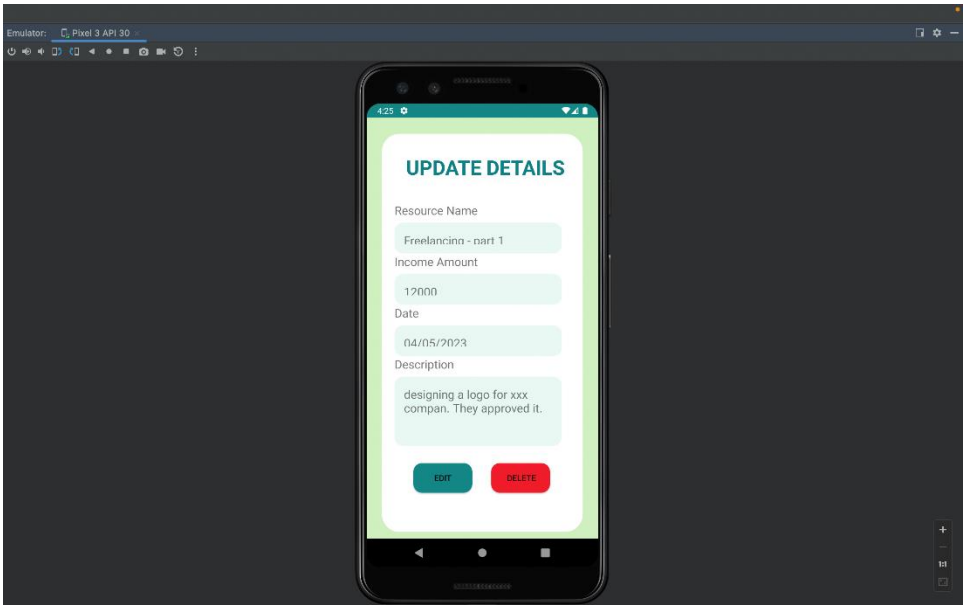


Updating Income: when the user clicks the “EDIT” button the record can be updated and it is displayed in the income page.

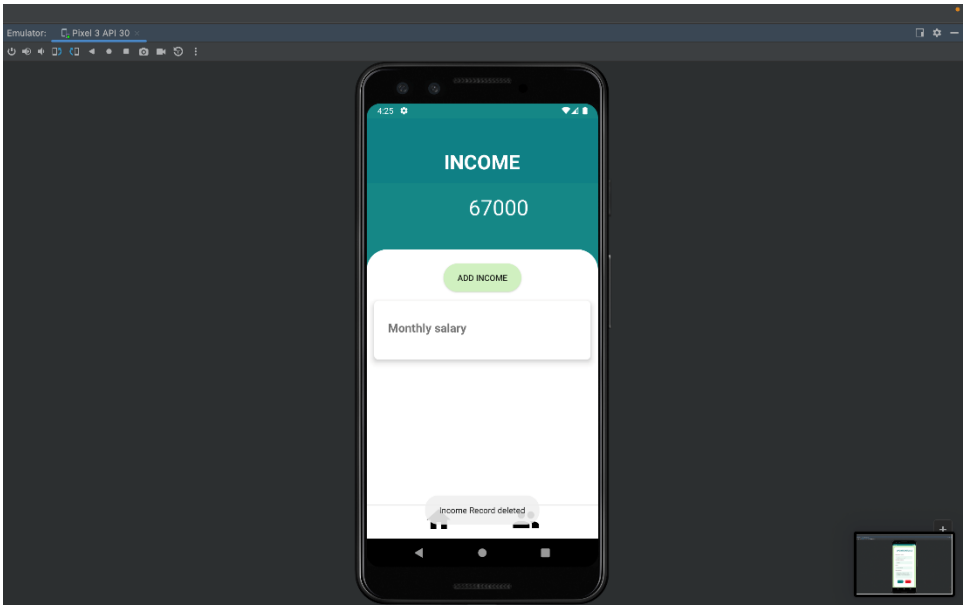




Deleting Income: when the user clicks the “DELETE” button the record gets deleted

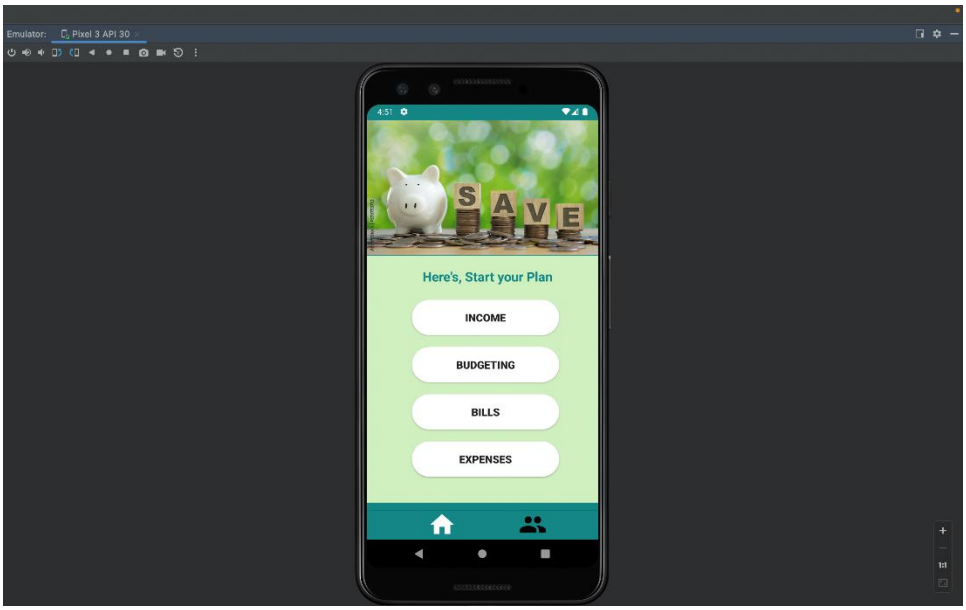


A toast message is displayed, and they get back to the “INCOME” page where the total income gets amended.

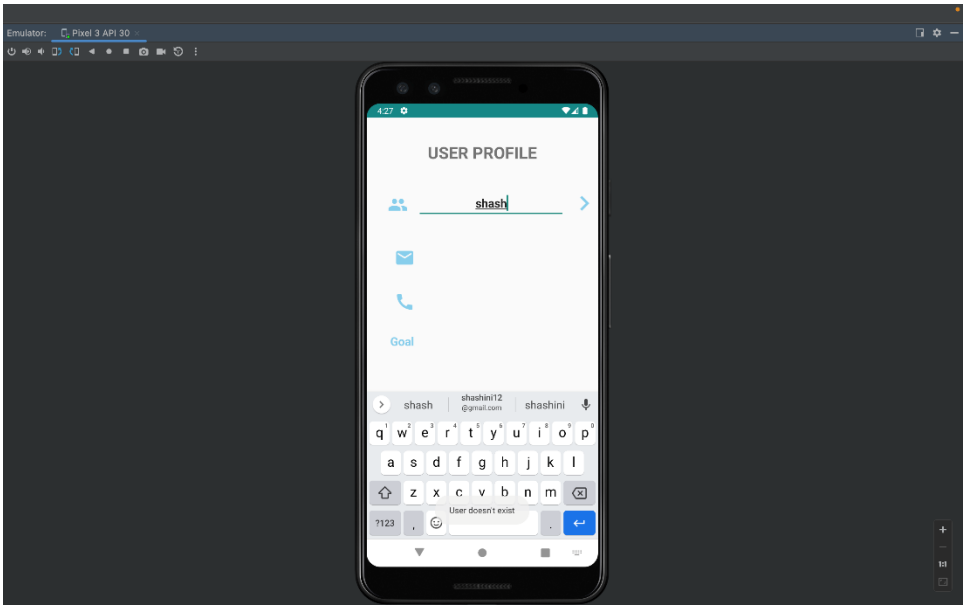


PROFILE MANAGEMENT

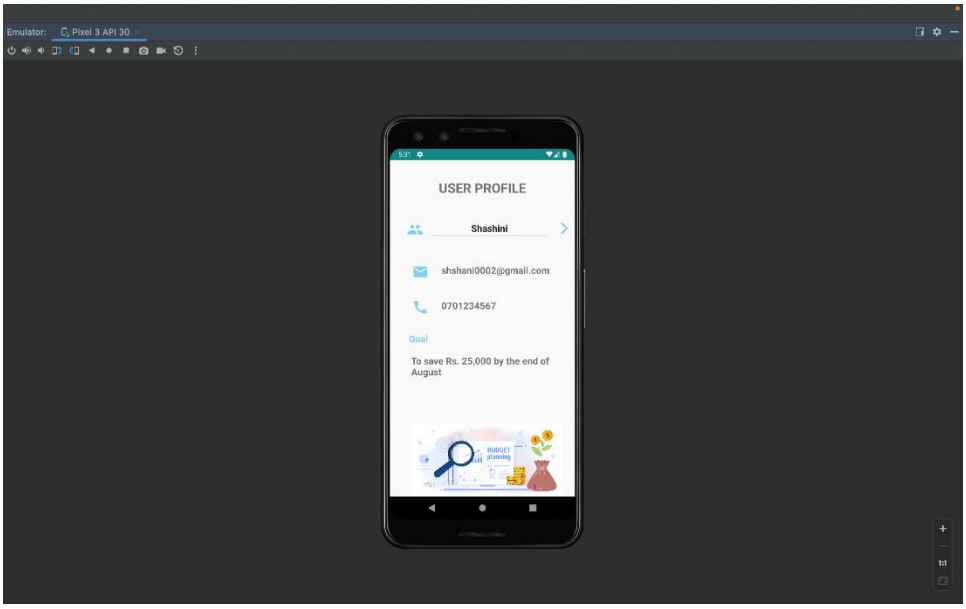
The user can navigate to the USER PROFILE page by clicking the user vector image on the footer. The user can view their Goal and user details, but they can edit or delete any record.



If the user entered the username incorrectly a toast message is displayed



When the username is accurate the user details are retrieved in the USER PROFILE



IT21189876 - **Budget management**

If the user needs to add a new plan they need to click “ADD NEW PLAN” button, then they get directed to “Add New plan “UI.

If they need to edit or delete a record, select the record using a checkbox and click the edit button.

- Instrumented Testing –

Check whether When clicking add button in add new plan ui All added categories and amount redirect to the all budget plan page.

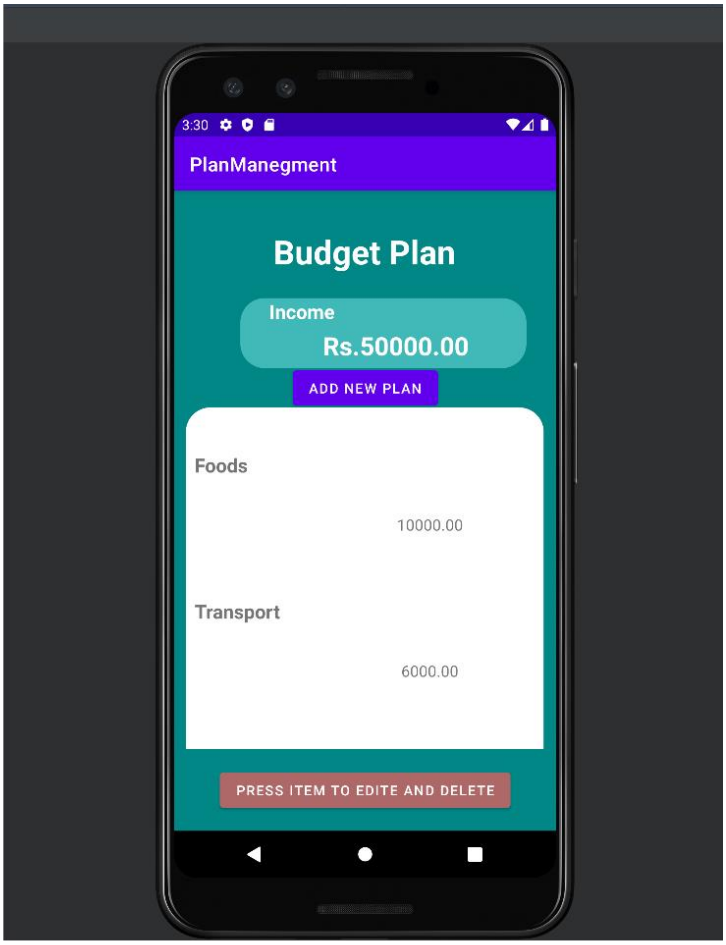
```
package com.example.budgetapp

import androidx.test.core.app.ActivityScenario
import androidx.test.espresso.Espresso
import androidx.test.espresso.action.ViewActions
```

```
import androidx.test.espresso.assertion.ViewAssertions
import androidx.test.espresso.intent.Intents
import androidx.test.espresso.intent.matcher.IntentMatchers
import androidx.test.espresso.matcher.ViewMatchers
import androidx.test.ext.junit.rules.ActivityScenarioRule
import androidx.test.ext.junit.runners.AndroidJUnit4
import androidx.test.platform.app.InstrumentationRegistry
import com.example.budgetapp.activities.FetchingBill
import com.example.budgetapp.activities.MainActivity
import org.junit.*
import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)
class ViewBillTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        Assert.assertEquals("com.example.appsquad", appContext.packageName)
    }

    @get:Rule
    val activityScenarioRule = ActivityScenarioRule(MainActivity::class.java)
    private lateinit var activityScenario: ActivityScenario<MainActivity>
    @Before
    fun setUp() {
        activityScenario = activityScenarioRule.scenario
        activityScenario.onActivity { activity ->
            // Do any setup you need for your activity here
        }
    }

    @Before
    fun setUp() {
        activityScenario = activityRule.activity
        activityScenario.onActivity { activity ->
            // Do any setup you need for your activity here
        }
    }

    @Test
    fun testButton(){


        Intents.init()
        Espresso.onView(ViewMatchers.withId(R.id.btnViewBill))
            .check(ViewAssertions.matches(ViewMatchers.isDisplayed()))

Espresso.onView(ViewMatchers.withId(R.id.btnViewBill)).perform(ViewActions.click());


        Intents.intended(IntentMatchers.hasComponent(FetchingBill::class.java.name))
```

```
    }
    @After
    fun tearDown() {
        activityScenario.close()
        }
}
```

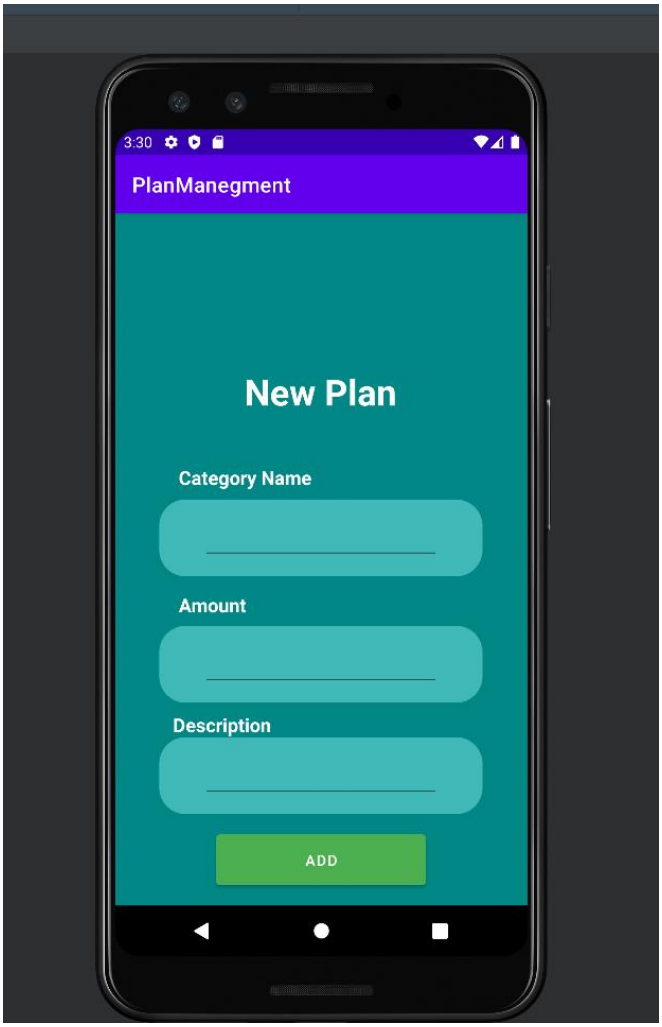
Budget Plan:

Users can see their plans.



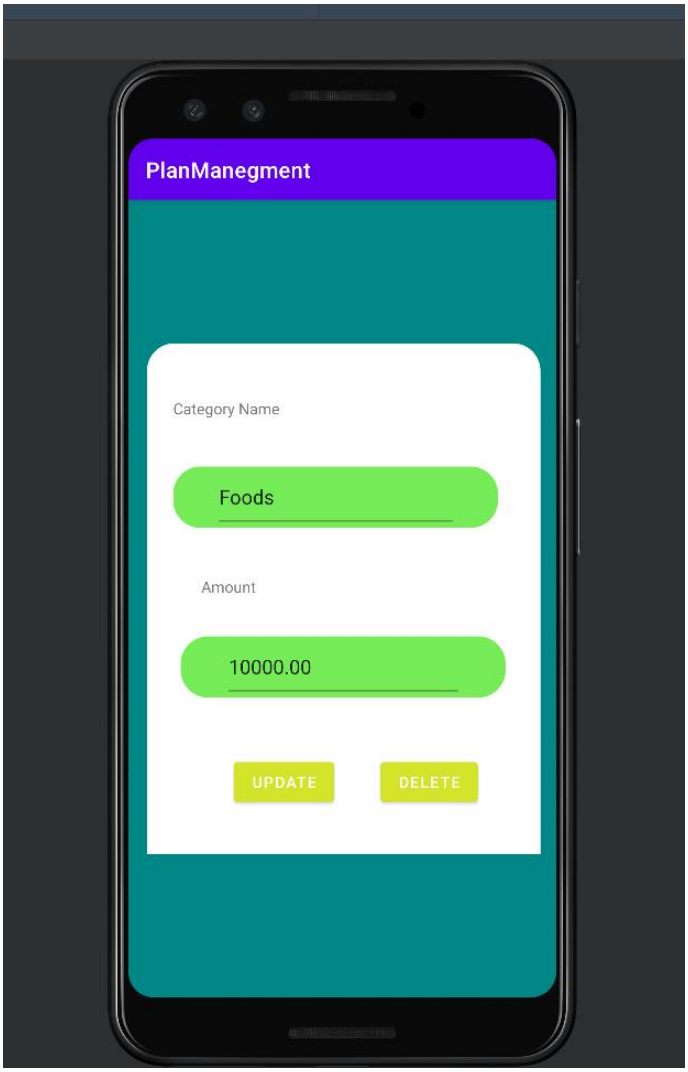
Add New Plan:

Users can add a new plan by inserting a new category, amount to allocate and a brief description.



Edit Plan Details:

Users can modify or remove their plan details.



IT21355196 - **Bill Management**

The user can view the bills they have according to the categories, if they have a new bill or a bill under a new category they can add it by clicking ”ADD NEW BILL”. The user can edit the bills in the categories by clicking the arrow button.

The user can view a record by clicking the bill record, then it gets directed to the bill page. There they can view the details and make the payment by clicking “PAY” button or delete the record by clicking “REMOVE” button.

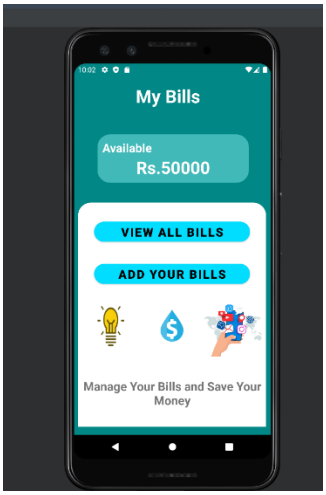
Instrumented Testing –

Check whether When clicking View All Bills button that page redirect to the all bills page

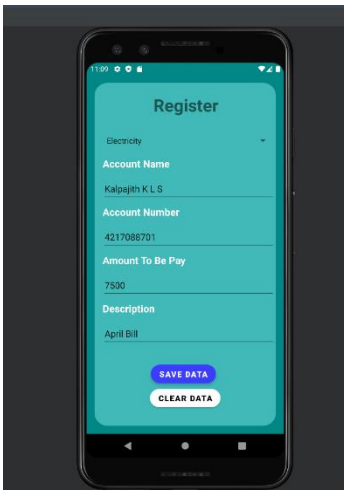
```
package com.example.budgetapp

import androidx.test.core.app.ActivityScenario
import androidx.test.espresso.Espresso
import androidx.test.espresso.action.ViewActions
import androidx.test.espresso.assertion.ViewAssertions
import androidx.test.espresso.intent.Intents
import androidx.test.espresso.intent.matcher.IntentMatchers
import androidx.test.espresso.matcher.ViewMatchers
import androidx.test.ext.junit.rules.ActivityScenarioRule
import androidx.test.ext.junit.runners.AndroidJUnit4
import androidx.test.platform.app.InstrumentationRegistry
import com.example.budgetapp.activities.FetchingBill
import com.example.budgetapp.activities.MainActivity
import org.junit.*
import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)
class ViewBillTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        Assert.assertEquals("com.example.appsquad", appContext.packageName)
    }

    @get:Rule
    val activityScenarioRule = ActivityScenarioRule(MainActivity::class.java)
    private lateinit var activityScenario: ActivityScenario<MainActivity>
```

```
    @Before
    fun setUp() {
        activityScenario = activityScenarioRule.scenario
        activityScenario.onActivity { activity ->
            // Do any setup you need for your activity here
        }
    }

    @Before
    fun setUp() {
        activityScenario = activityRule.activity
        activityScenario.onActivity { activity ->
            // Do any setup you need for your activity here
        }
    }

    @Test
    fun testButton(){


        Intents.init()
        Espresso.onView(ViewMatchers.withId(R.id.btnViewBill))
            .check(ViewAssertions.matches(ViewMatchers.isDisplayed()))

Espresso.onView(ViewMatchers.withId(R.id.btnViewBill)).perform(ViewActions.click());


        Intents.intended(IntentMatchers.hasComponent(FetchingBill::class.java.name))
    }
    @After
    fun tearDown() {
        activityScenario.close()
        }
}
```

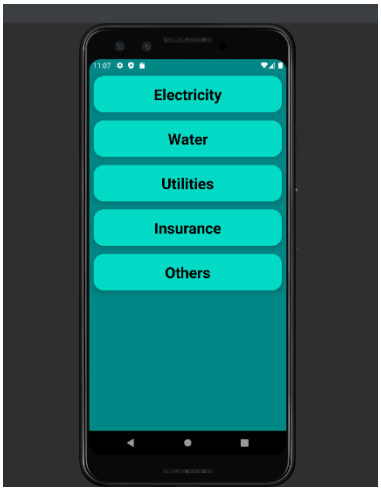
1.User gets redirect to the My Bills Home Page after clicking the ‘BILLS’ button in the User Profile



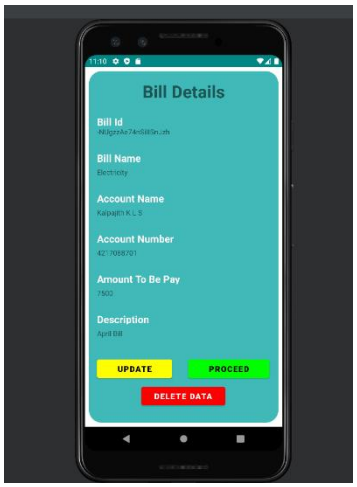
2. User can see their bills by clicking ‘VIEW ALL BILLS’ button and can add the new bill by clicking ‘ADD YOUR BILLS’ button.

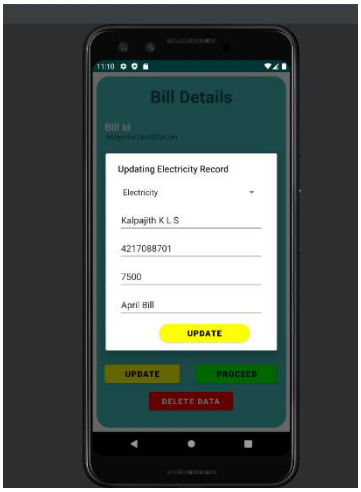


3.User can Register the bill by entering relevant details. (Insert)

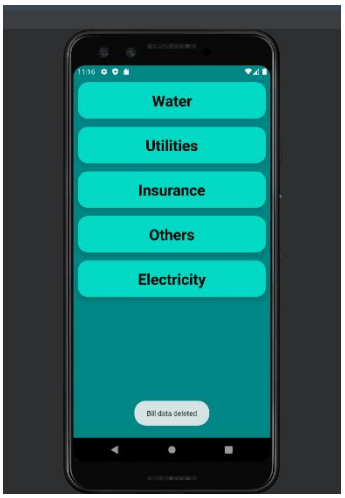


4.After registering the Bill to the System that bill retrieve to the All-Bills UI.

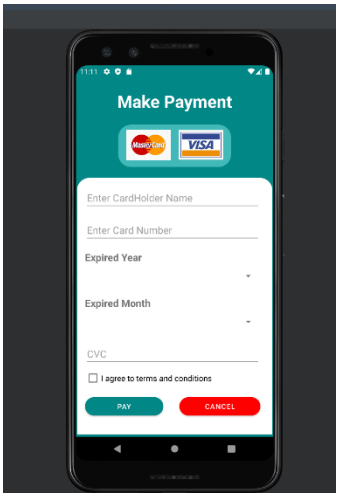




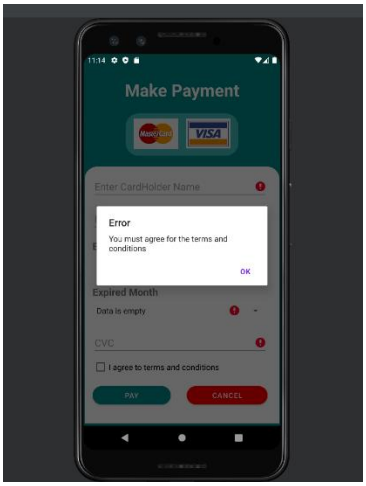
5.User can Update their bill according to their preference



6.User can delete the bill if it is not needed



7.If User Click the ‘Proceed’ button User enters to the Payment Page.



8. User must fill the Payment Page with correct valid details otherwise user cannot do the payment

IT21374524 - **Expense management**

Instrumented Testing –

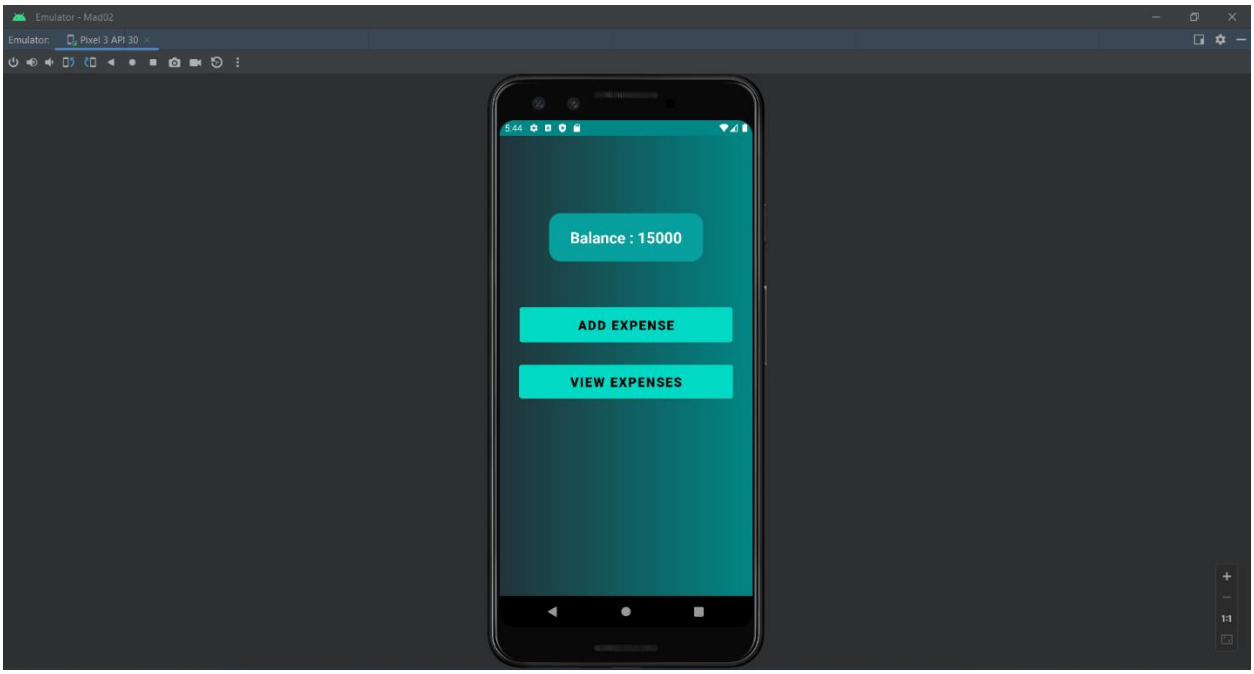
Check whether When clicking View Expenses button that page redirect to the All Expenses page.

```
package com.example.mad02

import androidx.test.core.app.ActivityScenario
import androidx.test.espresso.Espresso
import androidx.test.espresso.action.ViewActions
import androidx.test.espresso.assertion.ViewAssertions
import androidx.test.espresso.intent.Intents
```

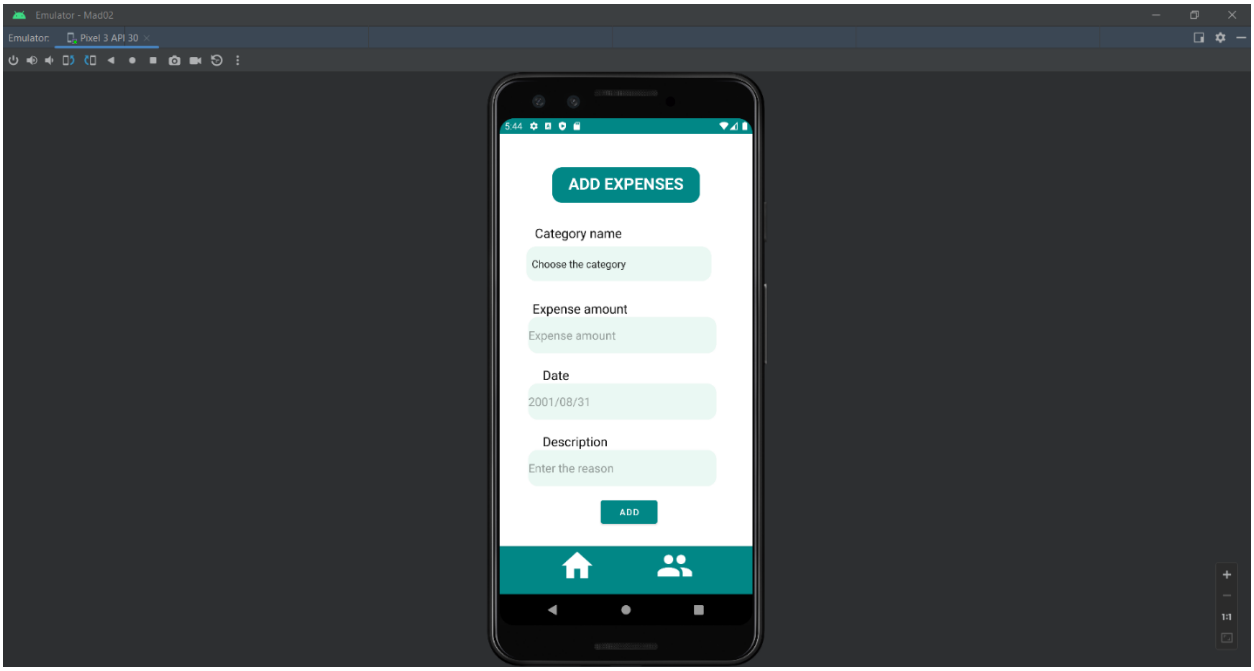
```
import androidx.test.espresso.intent.matcher.IntentMatchers
import androidx.test.espresso.matcher.ViewMatchers
import androidx.test.ext.junit.rules.ActivityScenarioRule
import androidx.test.ext.junit.runners.AndroidJUnit4
import androidx.test.platform.app.InstrumentationRegistry
import com.example.budgetapp.activities.FetchingBill
import com.example.budgetapp.activities.MainActivity
import org.junit.*
import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)
class ViewBillTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        Assert.assertEquals("com.example.appsquad", appContext.packageName)
    }

    @get:Rule
    val activityScenarioRule = ActivityScenarioRule(MainActivity::class.java)
    private lateinit var activityScenario: ActivityScenario<MainActivity>
    @Before
    fun setUp() {
        activityScenario = activityScenarioRule.scenario
        activityScenario.onActivity { activity ->
            // Do any setup you need for your activity here
        }
    }

    @Before
    fun setUp() {
        activityScenario = activityRule.activity
        activityScenario.onActivity { activity ->
            // Do any setup you need for your activity here
        }
    }

    @Test
    fun testButton(){


        Intents.init()
        Espresso.onView(ViewMatchers.withId(R.id.btnViewBill))
            .check(ViewAssertions.matches(ViewMatchers.isDisplayed()))

Espresso.onView(ViewMatchers.withId(R.id.btnViewBill)).perform(ViewActions.click());


        Intents.intended(IntentMatchers.hasComponent(FetchingBill::class.java.name))
    }
    @After
```

```
    fun tearDown() {
        activityScenario.close()
        }
}
```

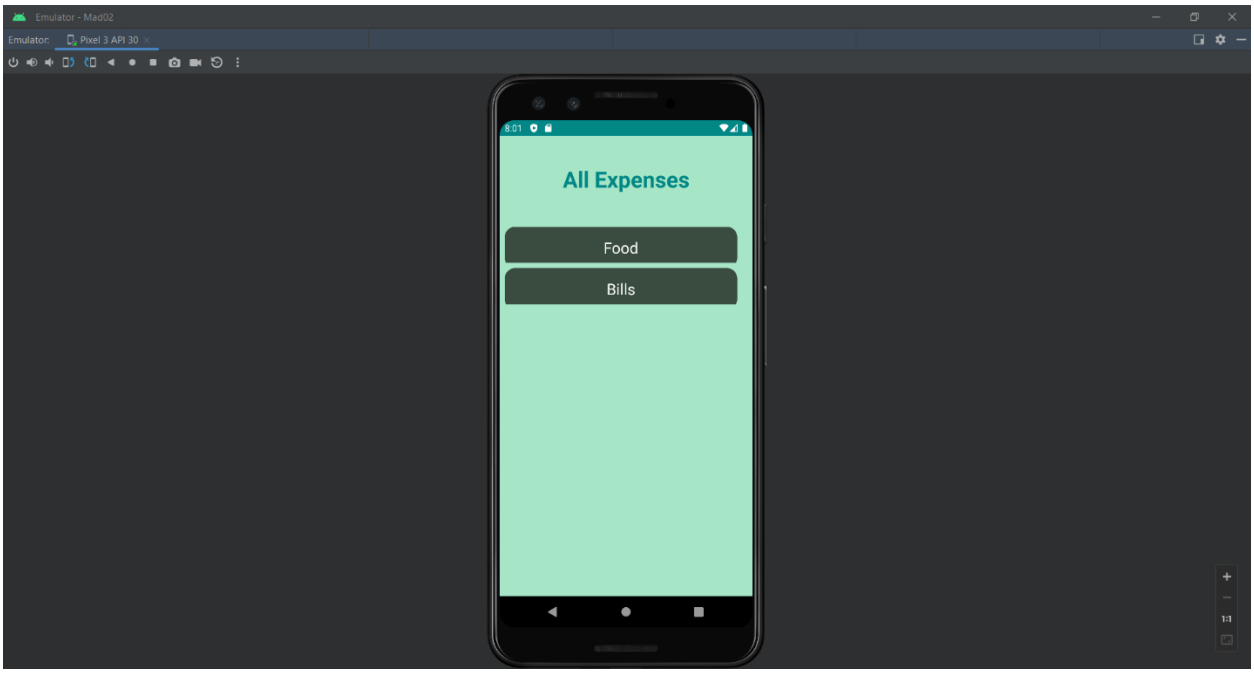
Expenses main: From here user can view the balance amount after conducting the expenses. The user can choose View expense to view the already saved expense records or Add expense to create a new expense.



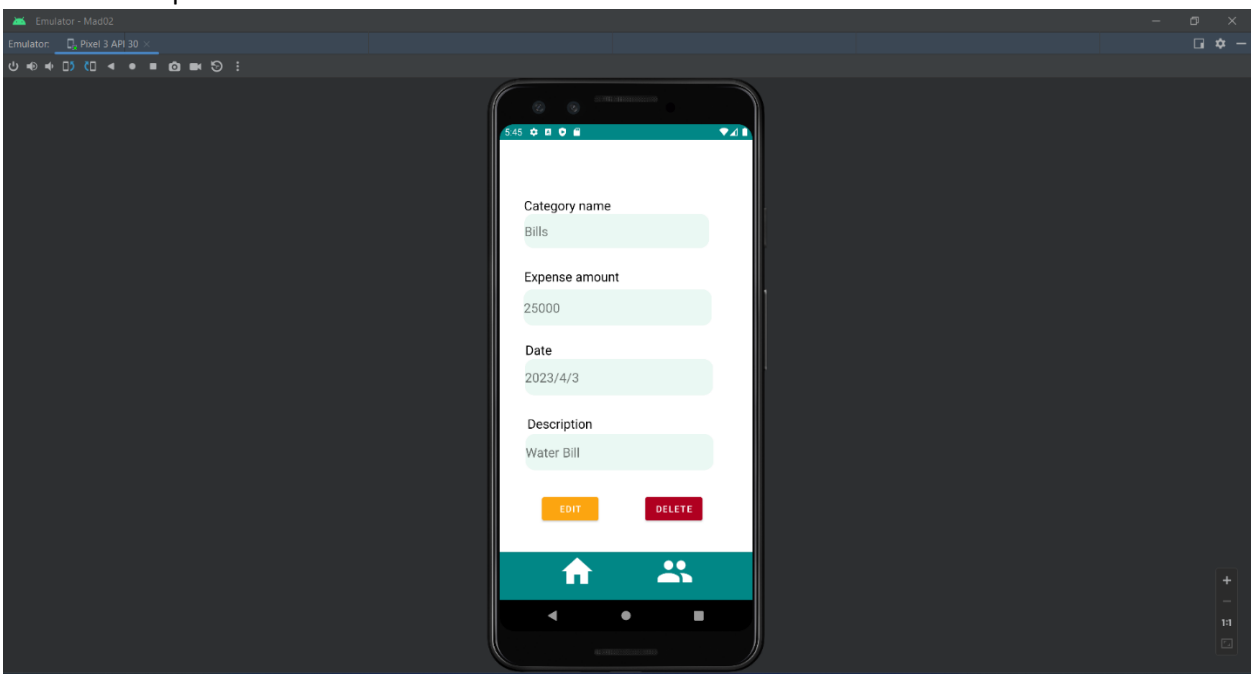
ADD: User add expense details



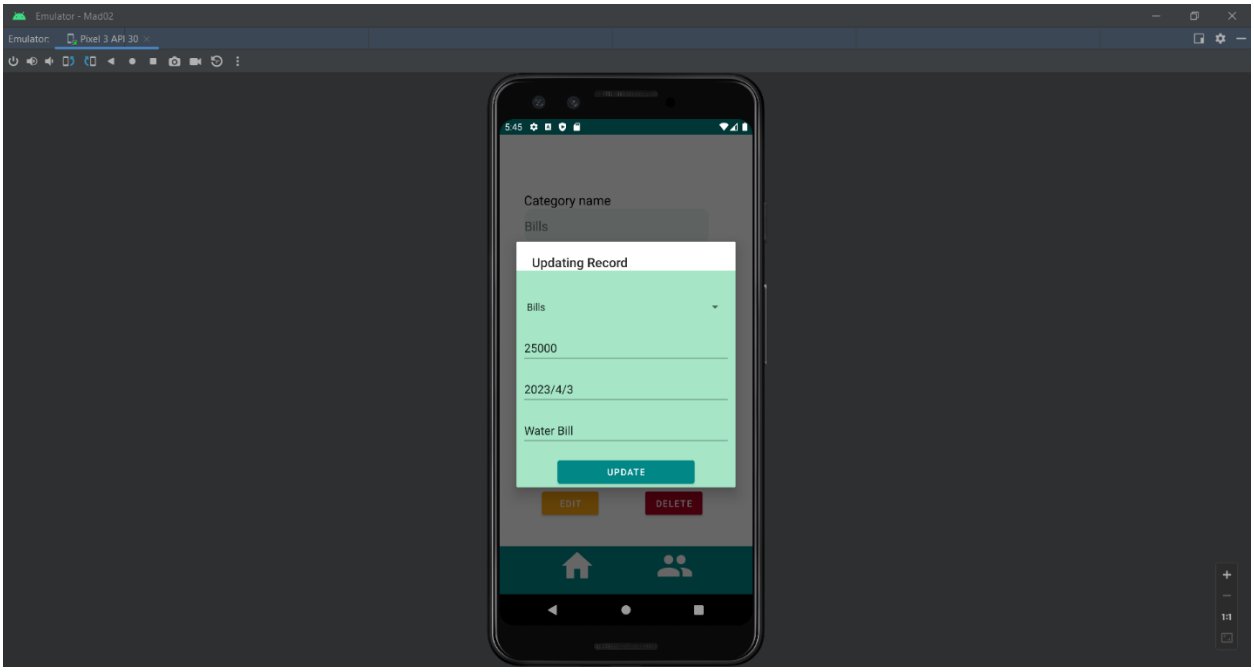
Read: User Views all the expenses made by the user.



Read: View expense's all the details and delete that record



Update: Update the record of expense.



Delete: When the user deletes the record the record gets deleted. And redirect to expense page.