2024

Paisa App

WHITE PAPER

Contents

Paisa Application

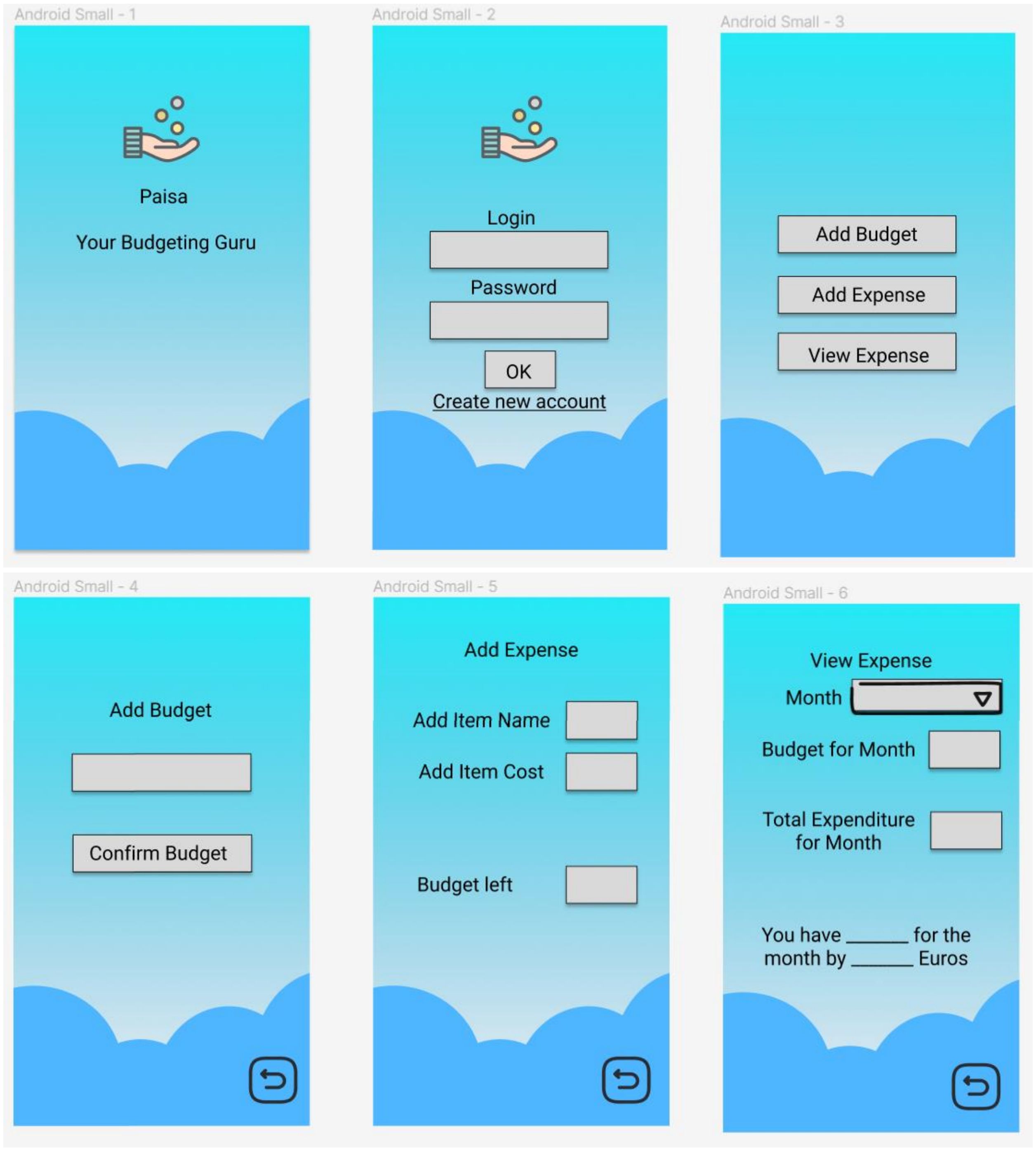
(White Paper)

1. Description

This budgeting application is designed to assist individuals in managing their finances effectively. With this application, users can input their monthly budget, track expenditures, and monitor progress toward their financial goals. The app aims to prompt users to be mindful of their spending.
The specific features of this application are:

- Budget Creation:
    - Users can add their monthly budget, which is editable. This feature ensures that the application can keep track of expenditures and indicates whether users are over or under budget, encouraging them to spend less or save money.
- Expenditure Capture:
    - Users can add each expense when it occurs so that the application can keep track of the same.
- Expense tracking:
    - Users can view past expenses, including the date of occurrence, to monitor spending history. Additionally, users can see the total expenses incurred up to the current date, providing a comprehensive overview. This feature helps users understand their total expenditure to date and manage future expenses based on this data.
- Over/Under Budget Alert:
    - Users receive alerts indicating whether they are over or under budget, facilitating better financial planning for the upcoming month.

2. **GUI Sketches**

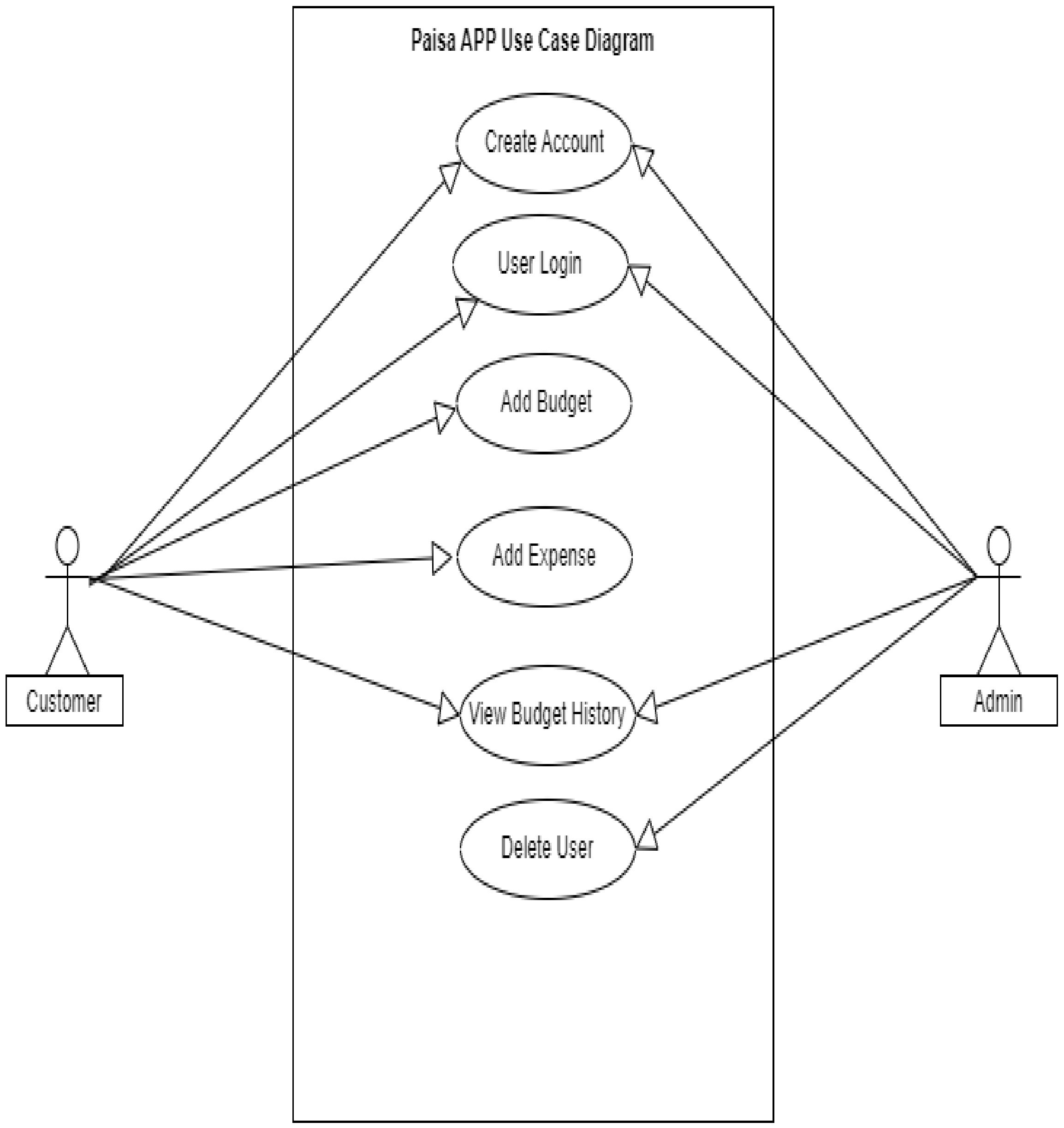


3. <u>Graphical User Interface Description:</u>

A clean and minimalistic user interface for easy access to different sections like User Login, Dashboard, Add Expenses, Add Budgets, and View Expense. The interface is designed to be responsive, ensuring a smooth and enjoyable user experience on various screen sizes. The UI features are below,

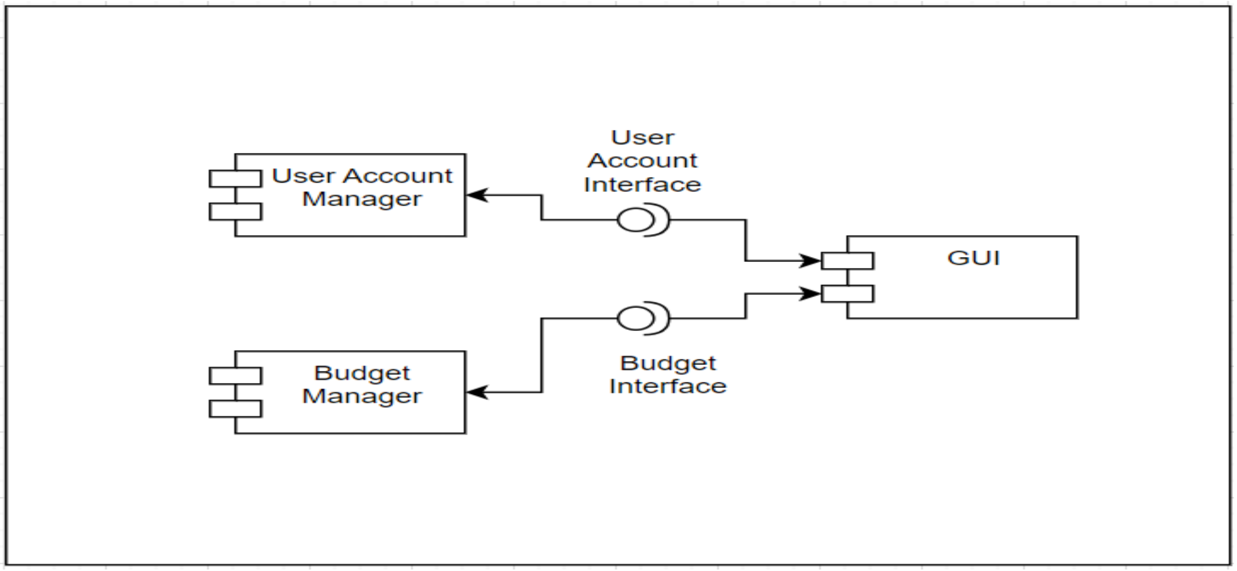
1. **User Login:** The primary function of user login is to authenticate and verify the identity of the user. It ensures that only authorized individuals gain access to the application's features and data. Users provide their unique credentials (such as username/email and password) during the login process to prove their identity.
2. **Dashboard:** It has easy access to different sections like Add Expense, Add Budget, and View Expense.
3. **Add Budget:** It serves as a fundamental function that enables users to set and manage their financial allocations every month. The primary purpose of the "Add Budget" feature is to allow users to set spending limits. Users can allocate a predefined amount of money they intend to spend within a certain period, typically monthly.
4. **Add Expense:** It serves as a core function that allows users to record and track their expenditures efficiently. Users can add their expenses when they make a purchase. The primary function is to enable users to log and record their expenses in real-time. Users can input details such as the name, and cost.
5. **View Expense:** Enables users to access and review their recorded expenses in a comprehensive and organized manner. Users can view their current spending against set limits. Users can search for specific expenses by using search filters for the “Month”. It assists in analyzing spending patterns, managing finances effectively, and making informed decisions to achieve financial goals.

4. Use Case Diagram:



5. Software Structure Components Overview

5.1 Component Diagram



5.2 API

We have 2 distinct API's in our Application.

Budget Manager:
The Budget Manager is employed by internal components related to budget. With this API, users can add their budgets and expenses, and also view previously added budgets, along with the remaining amount from the budget.

```
public interface BudgetManager {

   long addBudget(Context context, float budgetAmount, String month, String
customerName);
   CustomerBudget getBudget(Context context, String customerName, String
month);
   float getBalanceLeft(Context context, String customerName, String month);
   long addExpense(Context context, String item, float itemAmount, String month,
String customerName);
}
```

UserAccountInterface:
The UserAccountInterface API is designed for adding new users and authenticating them to grant access to the application.

```
public interface UserAccountInterface {
   long addCustomer(Context context, String userName, String password);
   boolean authenticateCustomer(Context context,String userName, String
password);
}
```

5.3Data Base

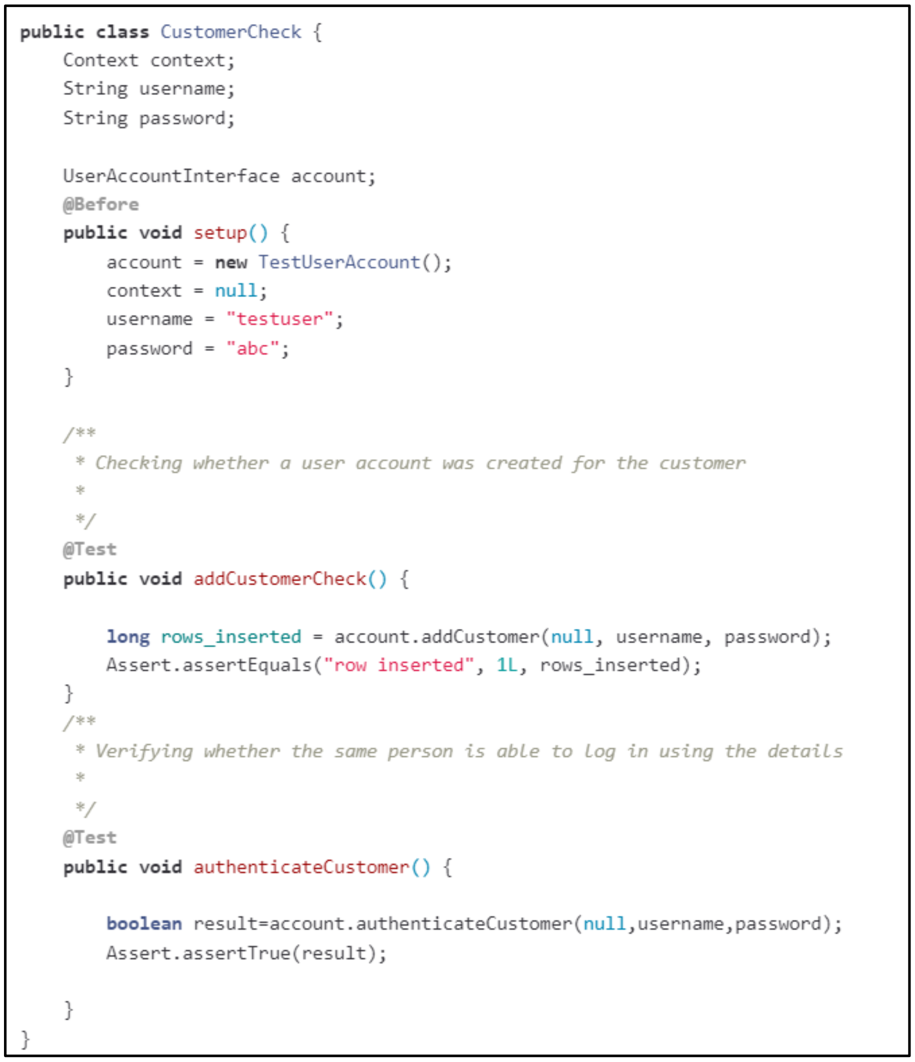
The implementation of the application has been done of the SQLite Database. This Relational Database was chosen as it has cross-platform compatibility, transactional, and requires no complex configurations.

5.4 Test Cases

5.4.1 Unit Test case:

Component1- **UserAccountInterface**:

In this set of unit tests, we aimed to verify the creation of a new account for a new customer. This was achieved by checking whether a new row is inserted with the account details of the customer, and subsequently, confirming if the same customer can be authenticated by logging in with the newly created account details



```
public class CustomerCheck {
    Context context;
    String username;
    String password;

    UserAccountInterface account;
    @Before
    public void setup() {
        account = new TestUserAccount();
        context = null;
        username = "testuser";
        password = "abc";
    }

    /**
     * Checking whether a user account was created for the customer
     *
     */
    @Test
    public void addCustomerCheck() {

        long rows_inserted = account.addCustomer(null, username, password);
        Assert.assertEquals("row inserted", 1L, rows_inserted);
    }
    /**
     * Verifying whether the same person is able to log in using the details
     *
     */
    @Test
    public void authenticateCustomer() {

        boolean result=account.authenticateCustomer(null,username,password);
        Assert.assertTrue(result);

    }
}
```

1. **addCustomerCheck**:

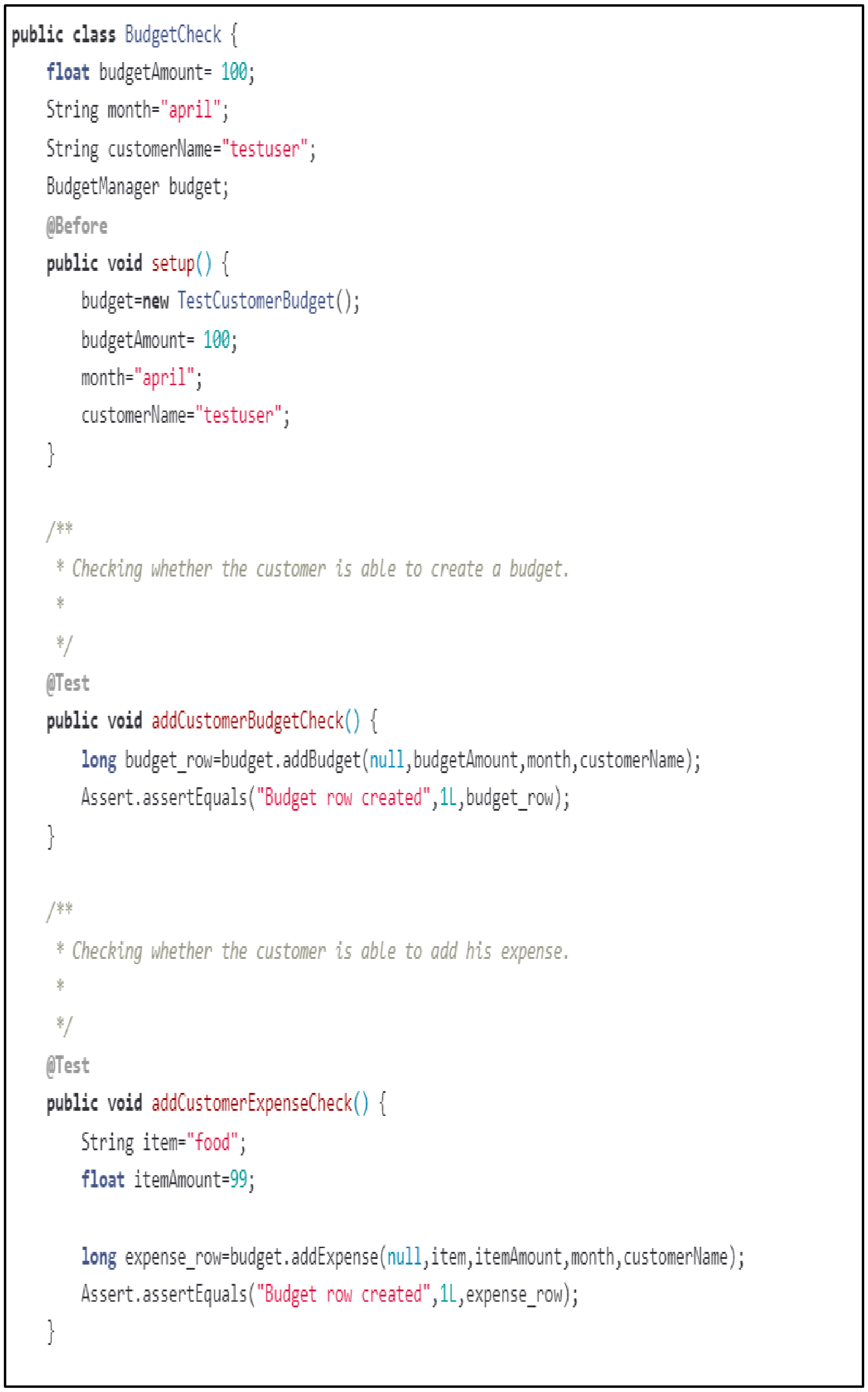
    - Requirement - Verify whether a user account is created for our customer.
    - Method - Call addCustomer on the account object, and the test asserts that the number of rows inserted is equal to 1.

2. **authenticateCustome**r:

    - Requirement - Verify whether a customer, whose account was created, can log in using a username and password.

Component2- **BudgetManagerInterface**

In this set of unit tests, we verify whether the customer can perform the following actions: add a budget for their total monthly expenses, add an expense for the month, view the balance left for expenses based on their set budget, and retrieve a specific customer's budget accurately

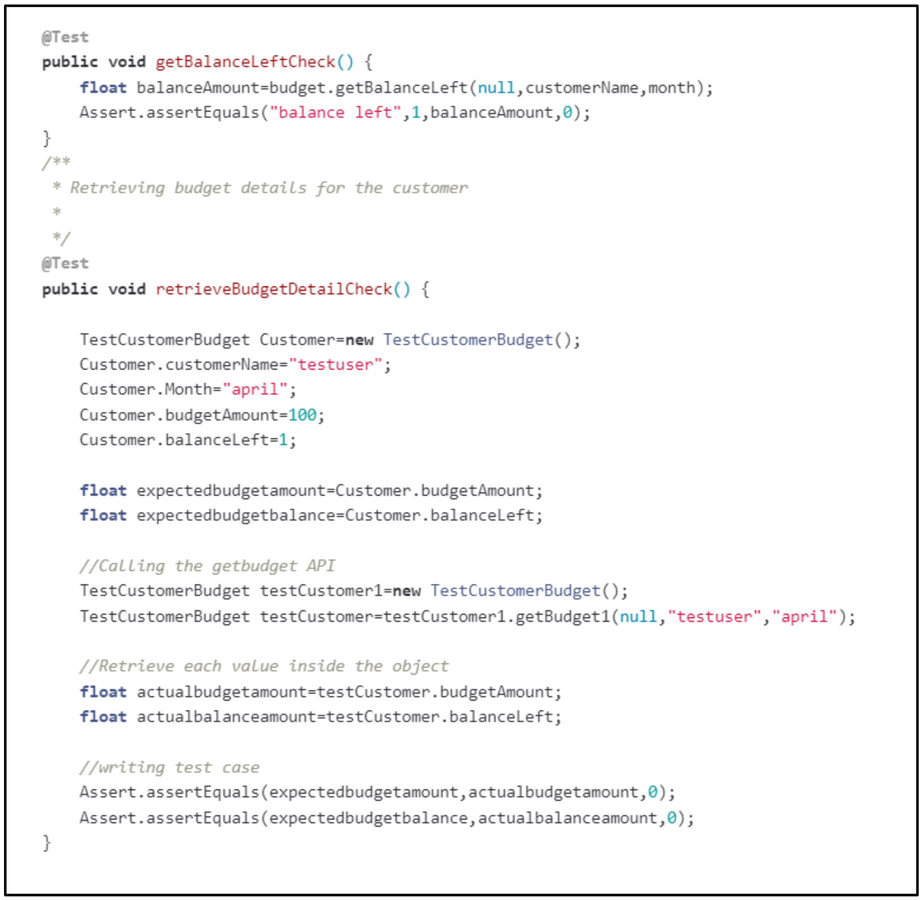


```
public class BudgetCheck {
    float budgetAmount= 100;
    String month="april";
    String customerName="testuser";
    BudgetManager budget;
    @Before
    public void setup() {
        budget=new TestCustomerBudget();
        budgetAmount= 100;
        month="april";
        customerName="testuser";
    }

    /**
     * Checking whether the customer is able to create a budget.
     *
     */
    @Test
    public void addCustomerBudgetCheck() {
        long budget_row=budget.addBudget(null,budgetAmount,month,customerName);
        Assert.assertEquals("Budget row created",1L,budget_row);
    }

    /**
     * Checking whether the customer is able to add his expense.
     *
     */
    @Test
    public void addCustomerExpenseCheck() {
        String item="food";
        float itemAmount=99;

        long expense_row=budget.addExpense(null,item,itemAmount,month,customerName);
        Assert.assertEquals("Budget row created",1L,expense_row);
    }
```



```
@Test
public void getBalanceLeftCheck() {
    float balanceAmount=budget.getBalanceLeft(null,customerName,month);
    Assert.assertEquals("balance left",1,balanceAmount,0);
}
/**
 * Retrieving budget details for the customer
 *
 */
@Test
public void retrieveBudgetDetailCheck() {

    TestCustomerBudget Customer=new TestCustomerBudget();
    Customer.customerName="testuser";
    Customer.Month="april";
    Customer.budgetAmount=100;
    Customer.balanceLeft=1;

    float expectedbudgetamount=Customer.budgetAmount;
    float expectedbudgetbalance=Customer.balanceLeft;

    //Calling the getbudget API
    TestCustomerBudget testCustomer1=new TestCustomerBudget();
    TestCustomerBudget testCustomer=testCustomer1.getBudget1(null,"testuser","april");

    //Retrieve each value inside the object
    float actualbudgetamount=testCustomer.budgetAmount;
    float actualbalanceamount=testCustomer.balanceLeft;

    //writing test case
    Assert.assertEquals(expectedbudgetamount,actualbudgetamount,0);
    Assert.assertEquals(expectedbudgetbalance,actualbalanceamount,0);
}
```

1. **addCustomerBudgetCheck**:

   - Requirement - It verifies whether a customer can create a budget.
   - Method - Call addBudget on the budget object, and the test asserts that the returned budget row is equal to 1.

2. **addCustomerExpenseCheck**:

   - Requirement - Verify whether the customer can add an expense for the month.
   - Method - Call addExpense on the budget object, and the test asserts that the returned expense row equals 1.

3. **getBalanceLeftCheck**:

   - Requirement - Verify if the customer can check their balance after expenditure based on their set budget.
   - Method - Call getBalanceLeft on the budget object, and the test asserts that the returned balance amount is equal to 1, with the previously checked expense added as 99 and the already set budget as 100.

4. **retrieveBudgetDetailCheck**:

   - Requirement - Verify if the leftover budget and budget for the month can be retrieved correctly.
   - Method - Set up a TestCustomerBudget object with expected values, call the getBudget1 method on another TestCustomerBudget object, and then compare the expected and actual values.

5.4.2 Integration Test:

Test Case ID	**Test Case objective**	**Test case description**	**Expected result**
1	Check the interface link between login page and budget dashboard	Enter Login details and click on the login button	The user should be redirected to the Budget Menu
2	Check whether details entered in the login page are reflected in the dashboard screen	Enter login details and click on the login button	The username of the user should be displayed as a header in the Budget Dashboard

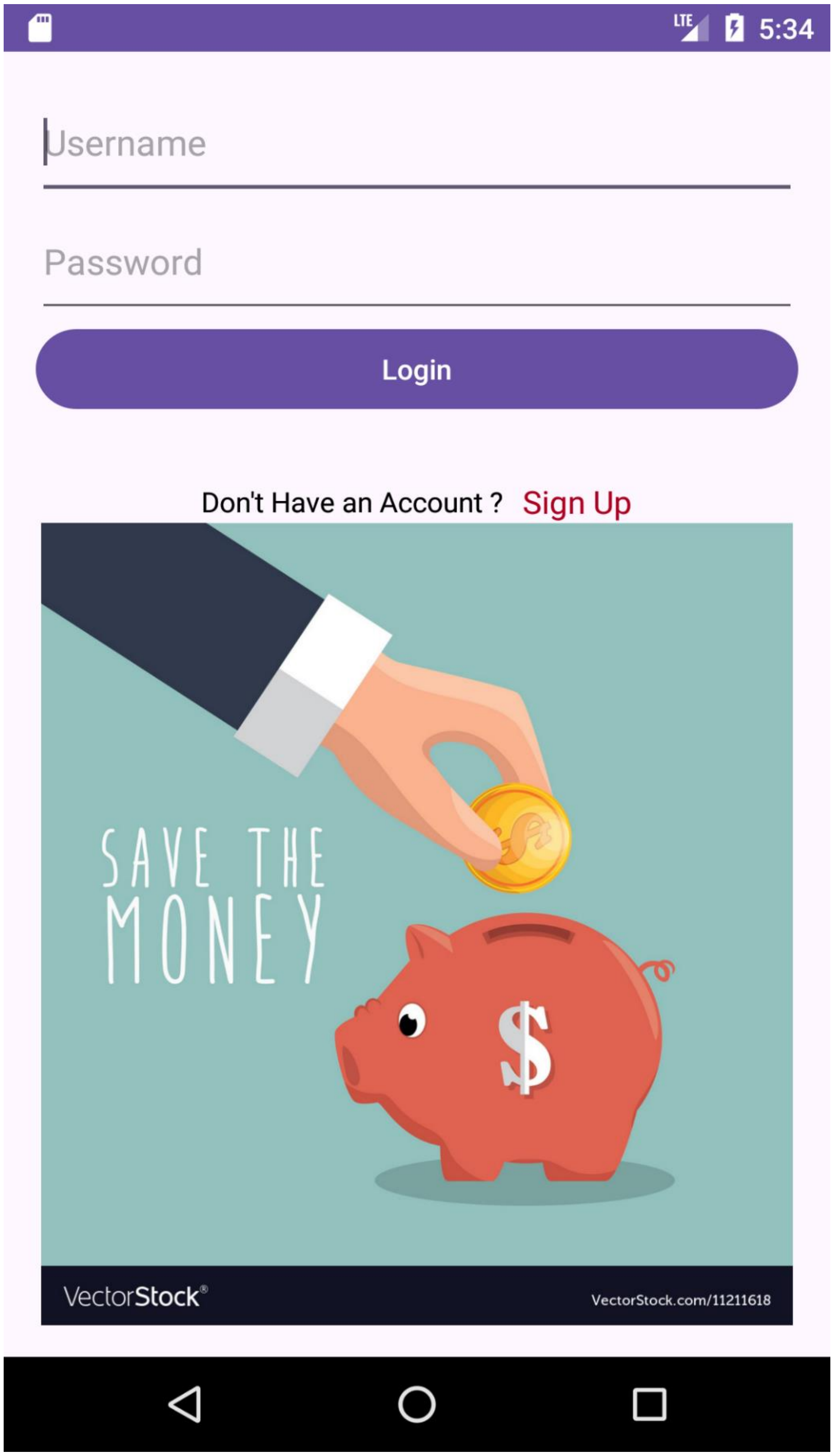
5.4.3 End-to-End Test Case:

Test Case ID:1
Test Case Description:

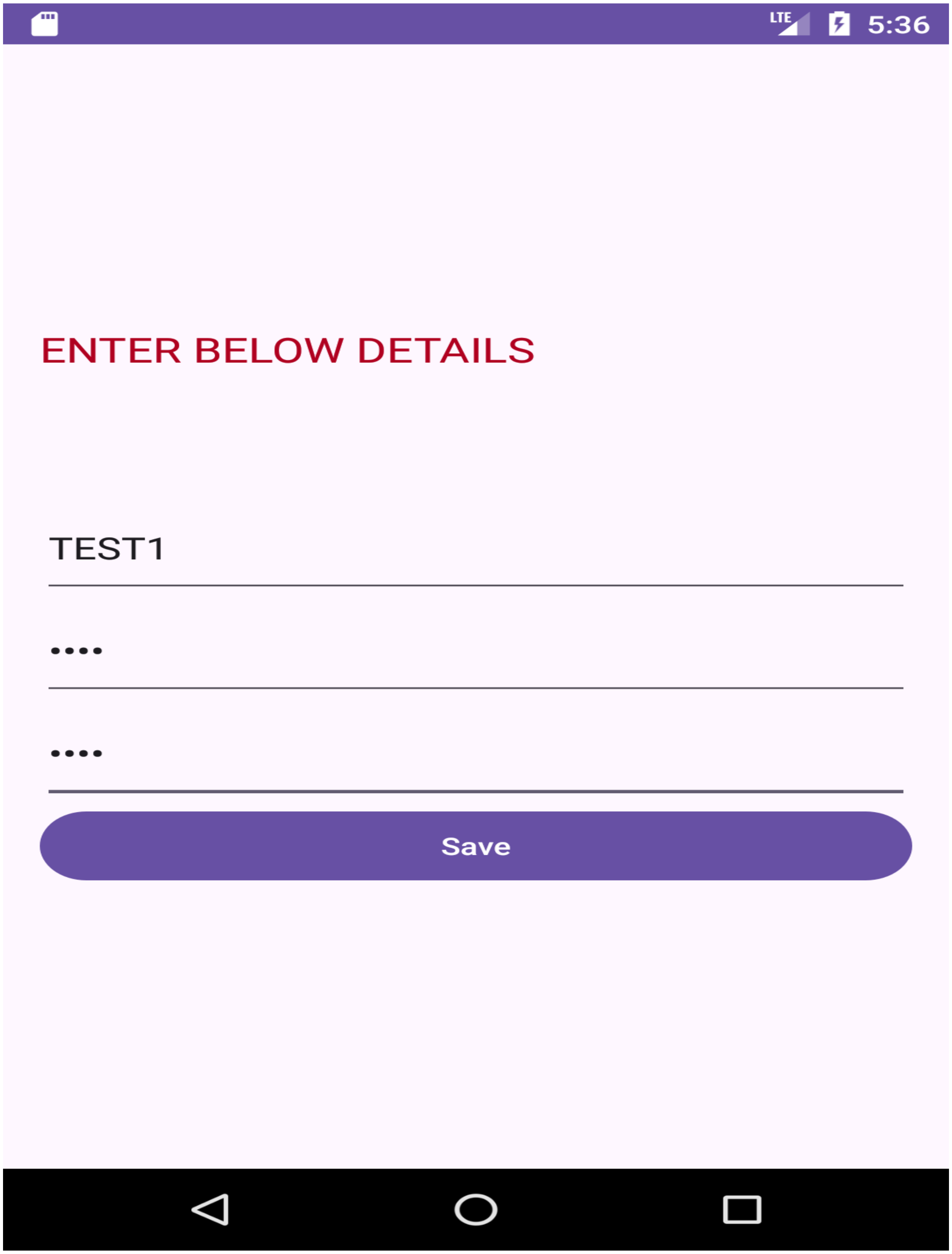
1. Create a new account for the user.
2. Log in to the app.
3. Upon user login, the user must be able to view the budget dashboard, where options to (add budget, add expense, view budget) are visible.
4. Click on "Add Budget" to redirect to the budget creation screen and create a new budget.
5. Click on "Add Expense" to redirect to the expense logging screen, where the user can log monthly expenses. Each time an expense is added, the user should be able to view the remaining balance from the budget.
6. If the user attempts to add an expense that exceeds the budget, a toast message should be displayed, indicating that the expense exceeds the allocated budget for the month.
7. After adding multiple expenses, click on "View Budget" to see the total budget amount and the money spent for the month.

Expected Result: All steps should be successfully completed, and the "View Budget" screen should display accurate information based on the created budget and logged expenses.

Test result screenshot:
Application start screen



Create a new account for the user



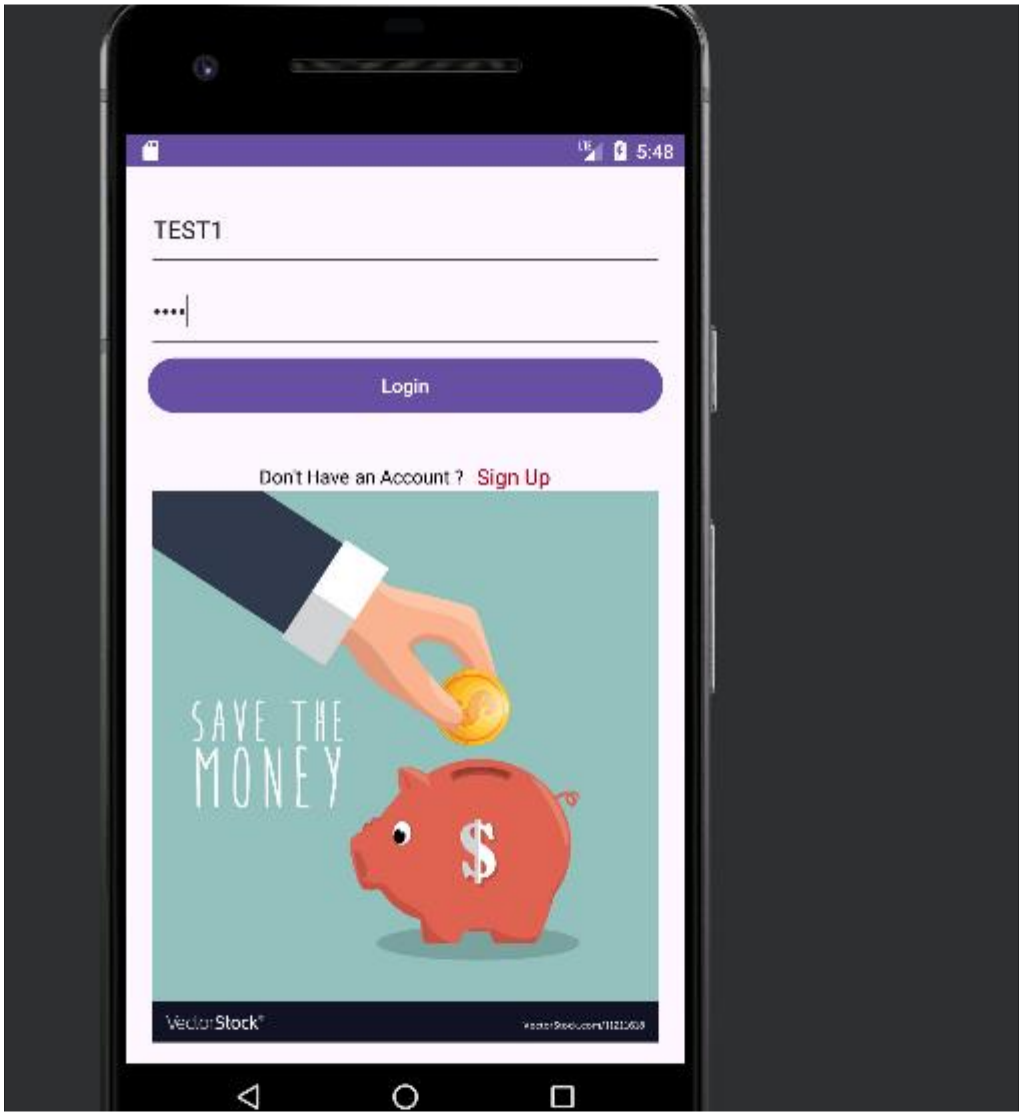
ENTER BELOW DETAILS

TEST1

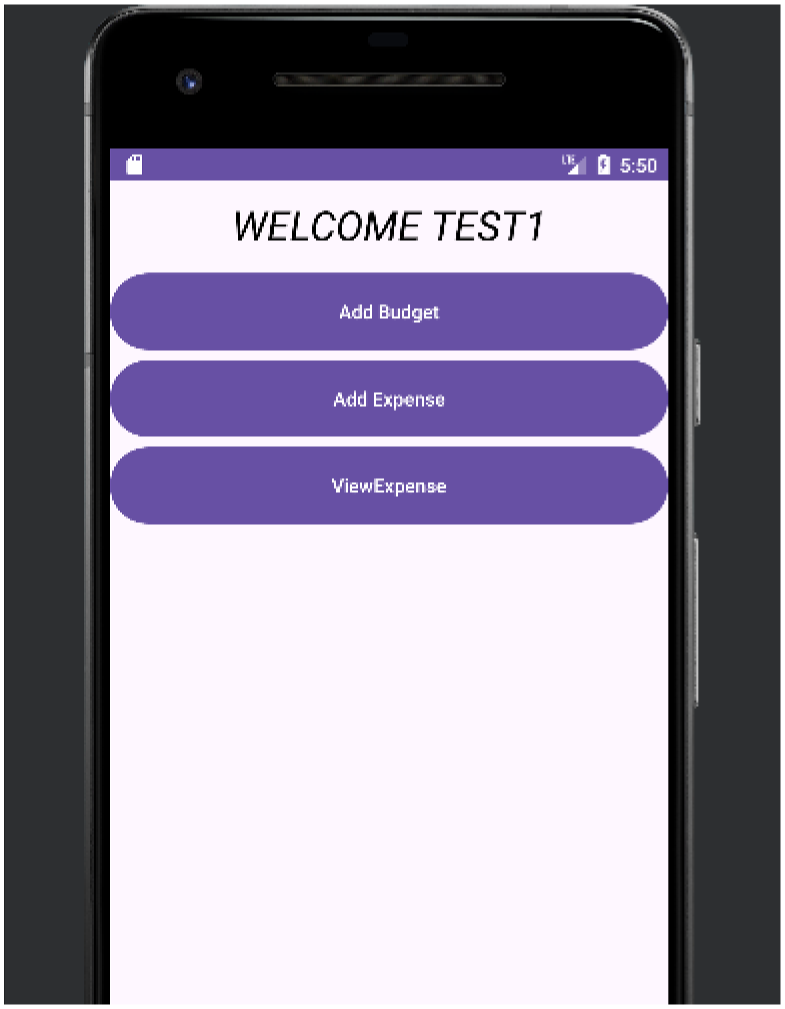
••••

••••

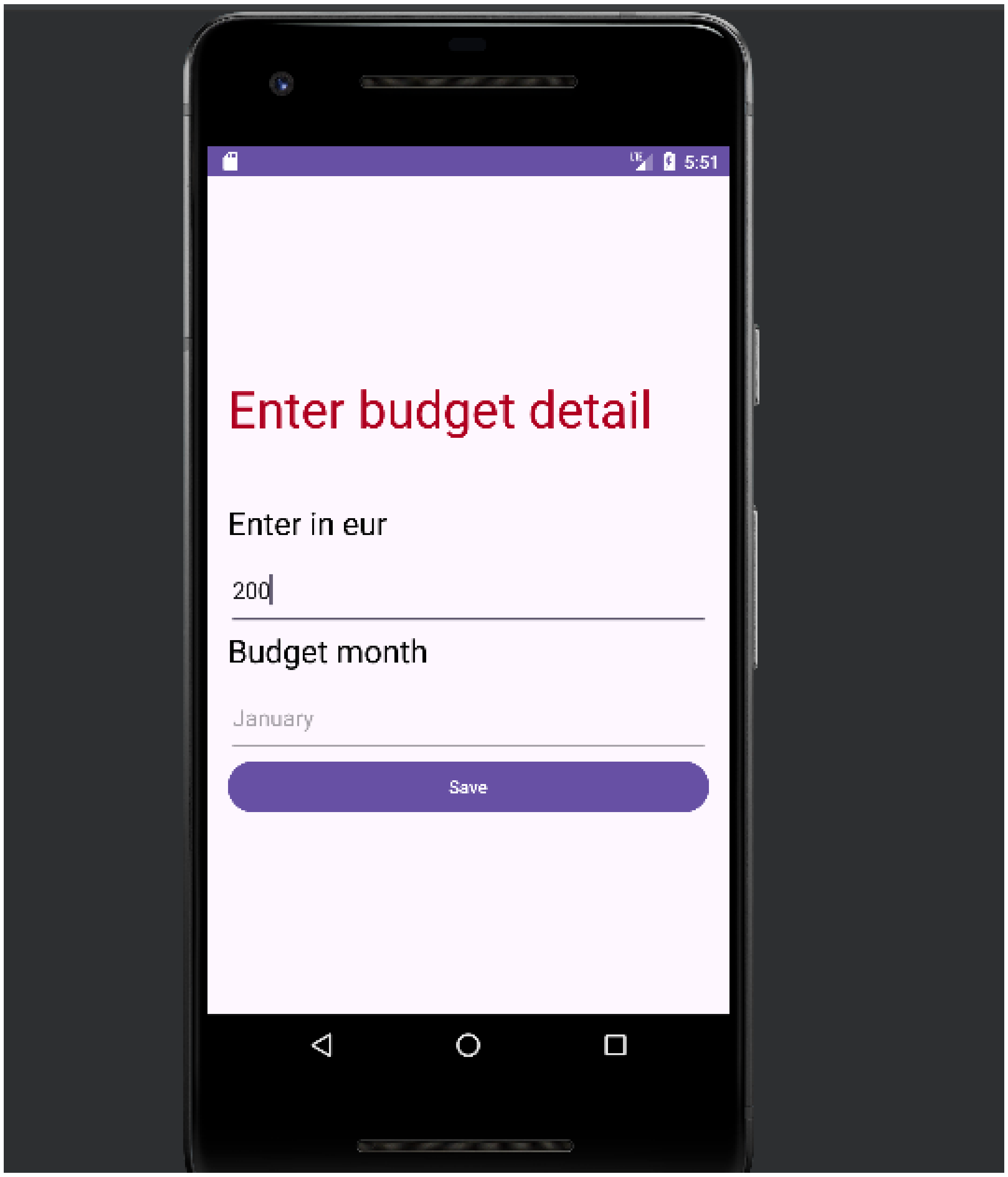
Enter user login detail in login page:



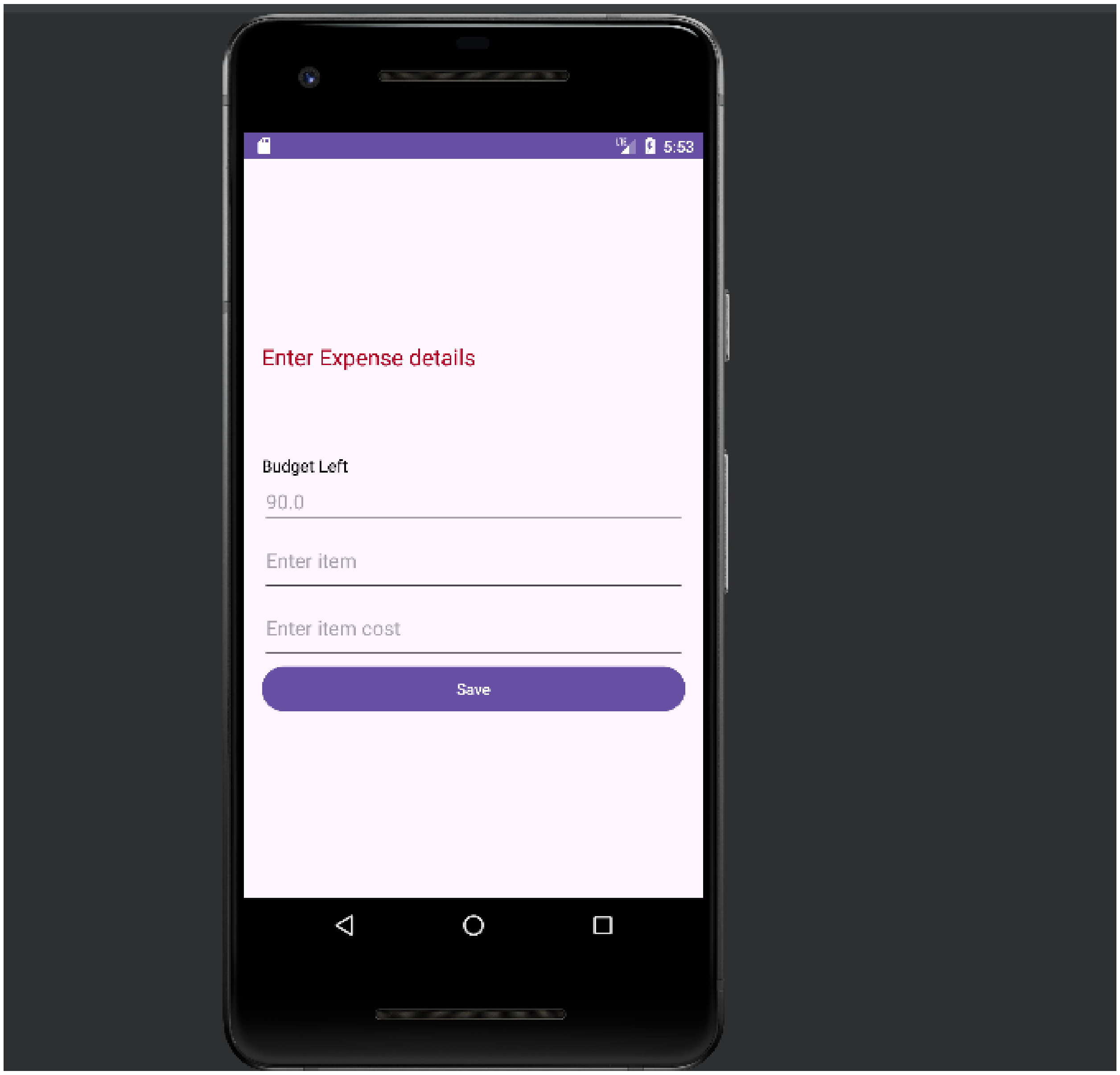
Upon user login, the user must be able to view the budget dashboard, where options to (add budget, add expense, view budget) are visible



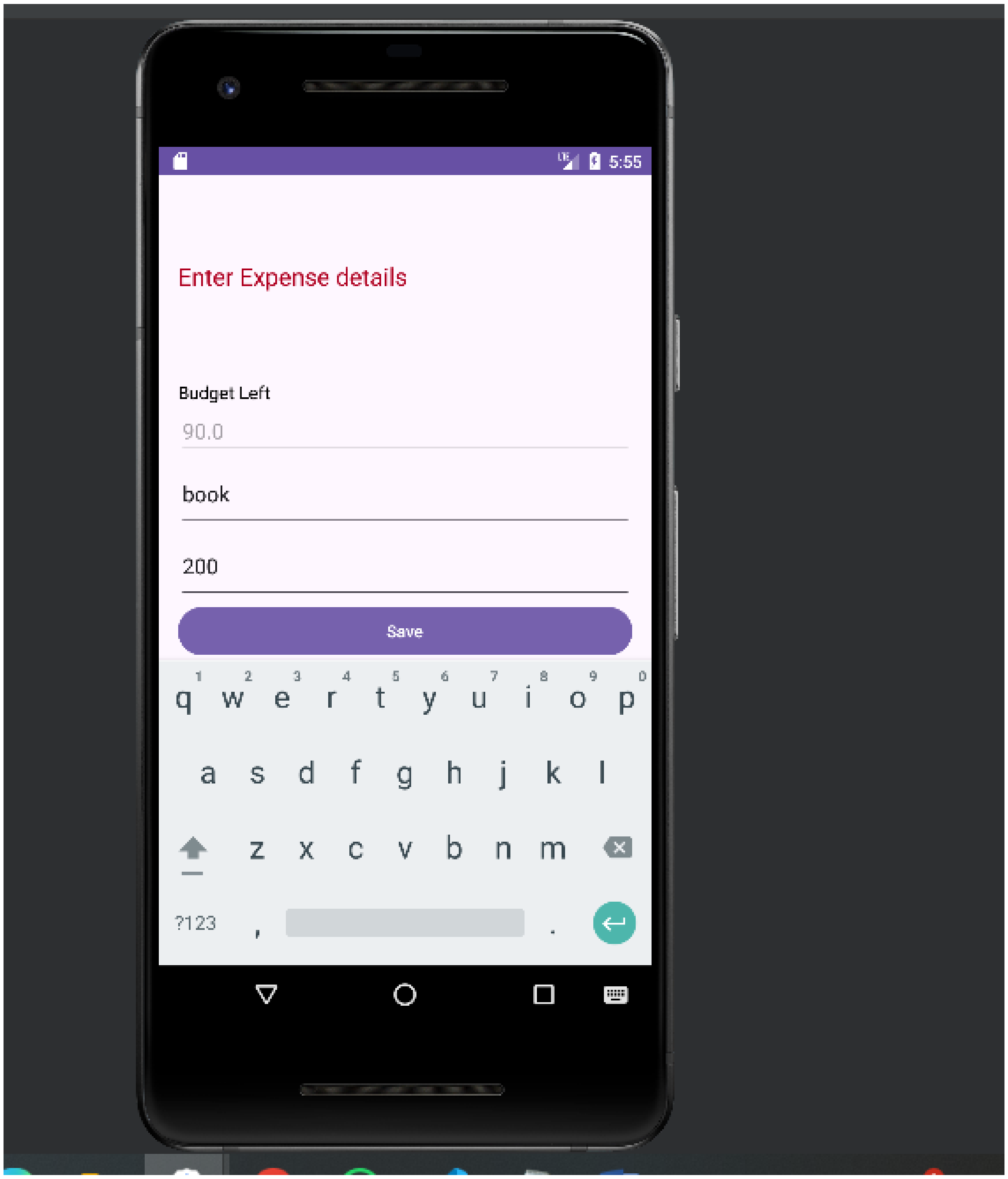
Click on "Add Budget" to redirect to the budget creation screen and create a new budget.



Click on "Add Expense" to redirect to the expense logging screen, where the user can log monthly expenses. Each time an expense is added, the user should be able to view the remaining balance from the budget



If the user attempts to add an expense that exceeds the budget, a toast message should be displayed, indicating that the expense exceeds the allocated budget for the month



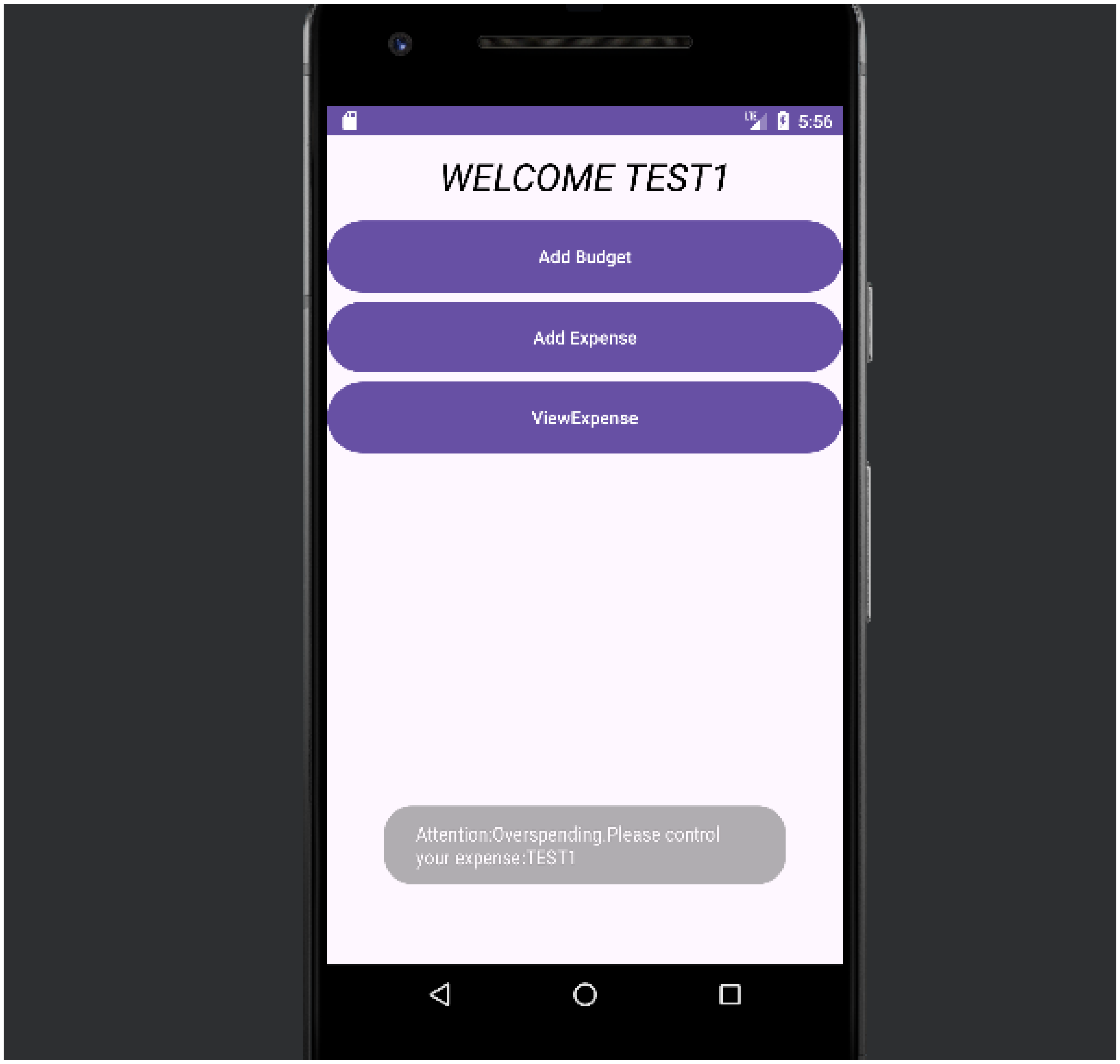
Enter Expense details

Budget Left

90.0

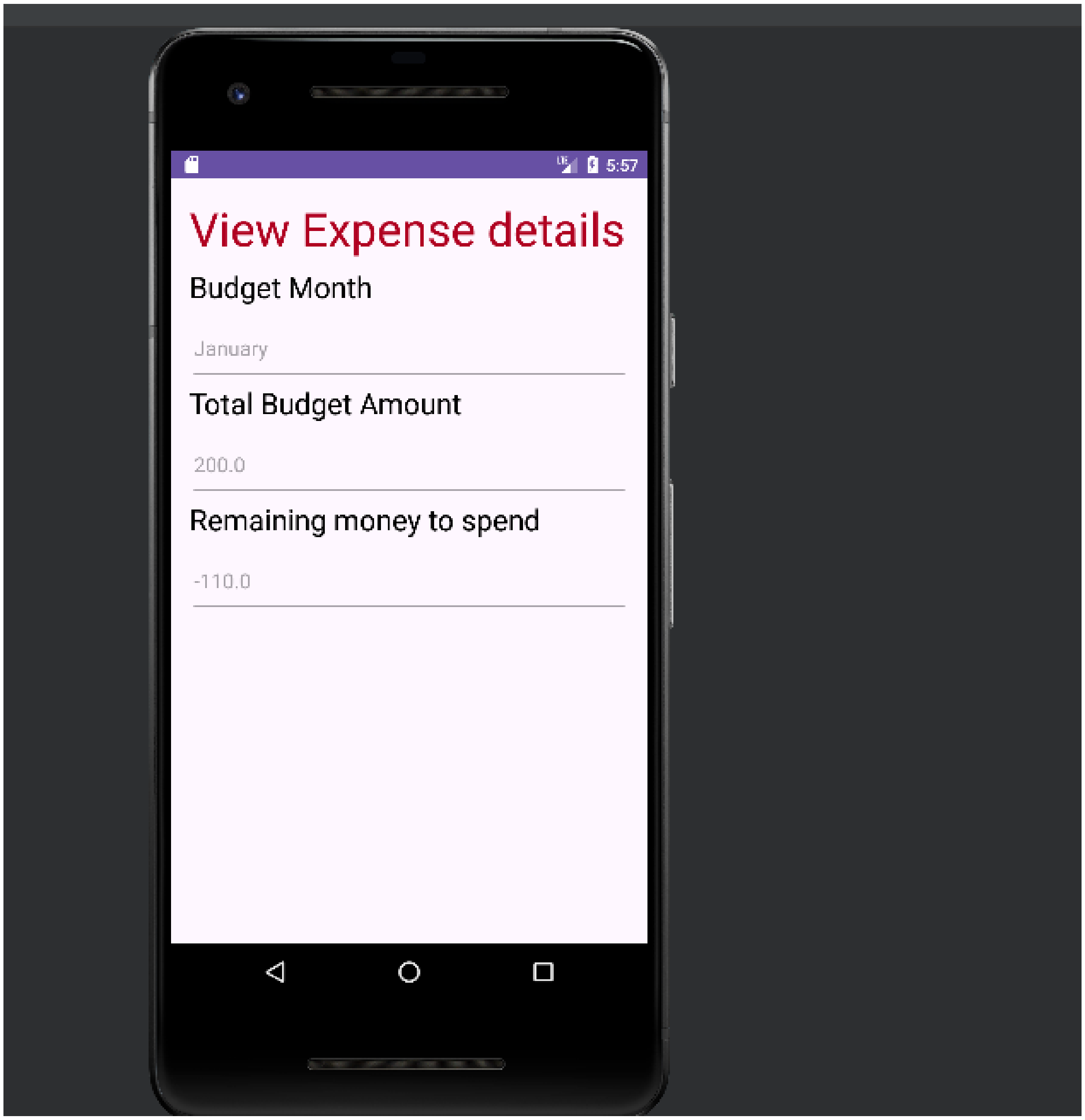
book

200



Attention:Overspending.Please control your expense:TEST1

After adding multiple expenses, click on "View Budget" to see the total budget amount and the money spent for the month



View Expense details

Budget Month

January

Total Budget Amount

200.0

Remaining money to spend

-110.0