N26 Challenge Solution

Task 01 Solution:

1) I have created exploratory document and explained the test cases and scenarios well. Attached exploratory and test plan document as well.
2) Major functionalities are working fine. Yes I found the bug, I have mentioned below all the detail.

Bug Detail:

Bug Title: When user click on selected page from Hamburg then nothing happens. Hamburg is not closing.
Bug ID: 001
Build Number: Version Number 1.16
Severity: Medium
Priority: Medium
Assigned to: Developer
Reported By: Raza
Reported On: 04/06/2022
Status: New/Open/Active (Depends on the Tool you are using)
Environment: Android
Description: When user click on selected page from hamburg then nothing happens. Hamburg is not closing. Ideally it should redirect to the detail page.
Steps to Reproduce:
1) Install the application.
2) Firstly check which page and account is selected. E.g Month is selected
3) Open Hamburg and click on Month button again.
4) Nothing happens, Hamburg is not closing.

Expected Result: On selecting Page and account hamburg should close and page should redirect to selected page.
Actual Result: Currently Hamburg is not closing.

3) I will explore Home page and Income Section first because home page have all the details that will display to user and income section have details about adding new income with different payment method. For end user these two modules are important.
4) Testing time is depend upon complexity level. QA need to execute all scenarios and test cases for maximum coverage. If in cases of automation scripts then QA also need time to setup framework and execute scripts. I have plan 2-4 hour for each character.

5) I will migrate these risks for this type of application
   - Identification & Impact of this app
   - Developing strategies
   - Acceptance requirement and analysis

Task 02 Solution:

✓ **Framework & Approach:**

I have automated the mobile application with Appium and Selenium TestNG Framework. Additionally, Page object model design pattern used for reusability and maintainability purpose.

Page Object model:

Page Object Model, also known as POM, is a design pattern in Selenium that creates an object repository for storing all web elements. It is useful in reducing code duplication and improves test case maintenance. In Page Object Model, consider each web page of an application as a class file.

Maven:

Maven is also used to manage a project's build, reporting and documentation from a central piece of information and a project will be easy shared without having to lots of manual libraries downloading**.**

Appium:

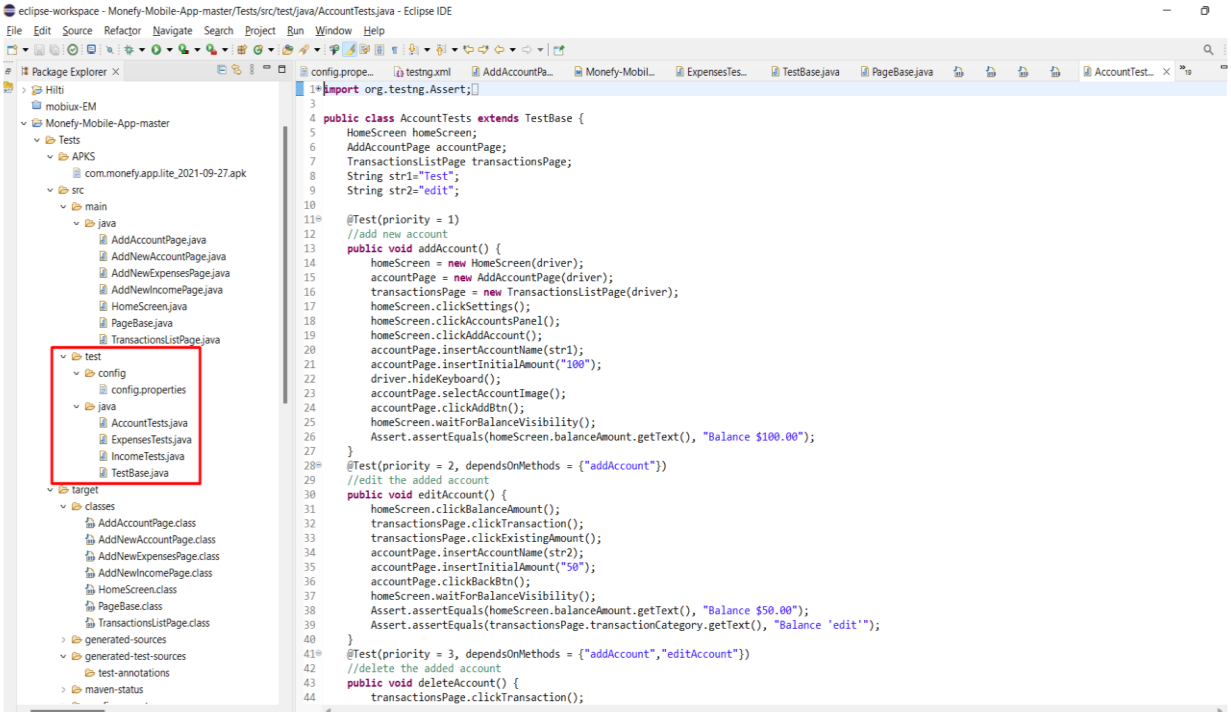
Appium is an opensource mobile testing tool. It is a platform independent so, it can be used on any machine and it does not need any enhancing in a tested application, so a developer does not need to customize his app. to be tested by Appium. Appium supported multi programming languages like Java, C#, PHP, Python and Ruby. Appium can test native mobile apps, hybrid mobile apps and browser apps. Appium can be used with multiple frameworks like data driven framework and behavior driven.

Automated Test Cases:

1) Account Page test cases
   Add New Account with Validation
   Edit Account Detail With Validation
   Delete the account with Validation

2) Expense Page Test Cases
   Add New Expense with Validation
   Edit Expenses and Validate the updated expenses
   Delete the expenses with validation

3) Income Page Test Cases
Add New Income with Validation
Edit Income with validation
Delete Added Income with Validation



```java
import org.testng.Assert;

public class AccountTests extends TestBase {
    HomeScreen homeScreen;
    AddAccountPage accountPage;
    TransactionsListPage transactionsPage;
    String str1="Test";
    String str2="edit";

    @Test(priority = 1)
    //add new account
    public void addAccount() {
        homeScreen = new HomeScreen(driver);
        accountPage = new AddAccountPage(driver);
        transactionsPage = new TransactionsListPage(driver);
        homeScreen.clickSettings();
        homeScreen.clickAccountsPanel();
        homeScreen.clickAddAccount();
        accountPage.insertAccountName(str1);
        accountPage.insertInitialAmount("100");
        driver.hideKeyboard();
        accountPage.selectAccountImage();
        accountPage.clickAddBtn();
        homeScreen.waitForBalanceVisibility();
        Assert.assertEquals(homeScreen.balanceAmount.getText(), "Balance $100.00");
    }
    @Test(priority = 2, dependsOnMethods = {"addAccount"})
    //edit the added account
    public void editAccount() {
        homeScreen.clickBalanceAmount();
        transactionsPage.clickTransaction();
        transactionsPage.clickExistingAmount();
        accountPage.insertAccountName(str2);
        accountPage.insertInitialAmount("50");
        accountPage.clickBackBtn();
        homeScreen.waitForBalanceVisibility();
        Assert.assertEquals(homeScreen.balanceAmount.getText(), "Balance $50.00");
        Assert.assertEquals(transactionsPage.transactionCategory.getText(), "Balance 'edit'");
    }
    @Test(priority = 3, dependsOnMethods = {"addAccount","editAccount"})
    //delete the added account
    public void deleteAccount() {
        transactionsPage.clickTransaction();
```

<u>Task 03 Solution:</u>

- ✓ Framework & Approach:

I have used Rest-Assured Library to automate Rest API end points with Maven. The benefits of using rest-assured is It removes the need for writing a lot of boilerplate code required to set up an HTTP connection, send a request and receive and parse a response.

These test cases primarily focusses on correct functionality of endpoints.