Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Fall, Year: 2023), B.Sc. in CSE (Day)

Automation Testing for Laundry Service Mobile App

Course Title: Software Testing and Quality Assurance Lab
Course Code: CSE 454
Section: 201 D4

Students Details

Name	ID
Monisha Aktet Asha	201002078
Mehedi Hasan	201002085
Maruf Mazumder	201002191

Submission Date: 12/1/2024
Course Teacher's Name: Md. Zahidul Hasan

[For teachers use only: Don't write anything inside this box]

Contents

Chapter 1

Introduction

1.0.1 Overview

We are embarking on a project focused on testing the Laundry service app using JUnit, an industry-standard automated testing tool. The project aims to ensure the seamless functionality, reliability, and security of the laundry service app, enhancing the overall user experience. By this Junit testing I will test the most useful field in the laundry service app.

1.0.2 Motivation

Testing is a crucial aspect of software development, and the Laundry Service App is no exception. The motivation behind thoroughly testing the app can be summarized in several key points:

- Ensuring Reliability and Stability: Testing helps identify and eliminate bugs, glitches, and unexpected behaviors in the application. This ensures that the app operates reliably and provides a stable experience to users.
- Enhancing User Experience: A well-tested app contributes to a positive user experience. Users expect applications to work seamlessly without crashes or errors. Testing helps in delivering an app that meets or exceeds these expectations.
- Maintaining Data Integrity: Laundry Service Apps often deal with sensitive user information, including personal details and payment data. Thorough testing helps in ensuring that data is stored, processed, and transmitted securely, protecting users from potential security breaches.
- Optimizing Performance: Performance testing helps in identifying bottlenecks, optimizing resource usage, and ensuring that the app performs efficiently under different conditions. This is crucial for providing a responsive and smooth experience to users.
- Validating Business Logic: The app likely includes complex business logic to manage customer orders, payments, and other processes. Testing ensures that the implemented logic aligns with the business requirements and functions correctly.

- Facilitating Continuous Deployment: A well-tested app makes it easier to adopt continuous integration and continuous deployment (CI/CD) practices. This allows for faster and more frequent releases while maintaining the quality of the application

1.0.3 Problem Definition

Problem Statement

The Laundry Service App requires comprehensive testing to address several critical challenges. The application must undergo thorough testing to ensure reliability, stability, and a positive user experience. Challenges include identifying and eliminating bugs, optimizing performance, validating complex business logic, ensuring data integrity and security, adhering to regulatory compliance, and supporting diverse devices and platforms. Continuous deployment, cost savings, trust-building, and scalability are also key concerns that necessitate effective testing. The goal is to deliver a high-quality application that meets user expectations, complies with regulations, and supports the growth of the laundry service.

Complex Engineering Problem

The laundry app testing poses a complex engineering challenge due to the need to ensure the reliability and stability of the application. The intricate business logic embedded in the app, managing customer orders, payments, and other processes, requires rigorous testing to validate its correctness and adherence to business requirements. Additionally, the app's handling of sensitive user information, such as personal details and payment data, demands meticulous testing to maintain data integrity and ensure secure storage, processing, and transmission of this information. Performance optimization is another facet of the challenge, involving the identification of bottlenecks and efficient resource usage to guarantee a responsive user experience under diverse conditions. Adhering to legal and regulatory compliance adds complexity, with the need to validate that the app meets standards related to user data protection and privacy. Finally, the goal of continuous deployment introduces the challenge of integrating testing into a continuous integration and continuous deployment (CI/CD) pipeline, allowing for faster and frequent releases without compromising the application's quality. The convergence of these challenges underscores the intricate nature of engineering a comprehensive testing strategy for the laundry service app.

1.0.4 Objectives

The objectives of testing the laundry app are multifaceted, encompassing various aspects of ensuring the application's functionality, reliability, security, and performance.

- Ensure that all features and functionalities of the laundry app work as intended.
- Identify and eliminate bugs, glitches, and unexpected behaviors to enhance the overall stability of the application.
- Validate the app's usability and user interface to provide a seamless and

- Ensure secure storage, processing, and transmission of sensitive user information, including personal details and payment data.
- Conduct performance testing to identify and address bottlenecks, optimizing resource usage for efficient operation.
- Assess the application's ability to scale efficiently to accommodate a growing user base.

1.0.5 Tools and technology

We've used various tools in this testing, those are:

- Espresso
- JUnit
- Apache JMeter
- Android Studio
- DbUnit
- SQLUnit

Chapter 2

1.1 Design/Develop/Implementation of the Project

1.1.1 Introduction

The implementation of the Laundry Service App project involves a comprehensive approach to designing, developing, and testing a software solution tailored for managing laundry services. The project aims to provide an efficient and user-friendly platform for customers, laundry service providers, and administrators.

1.1.2 Implementation Process

- Gather and analyze requirements from stakeholders, including customers, service providers, and administrators.
- Define the features, functionalities, and user roles needed for the Laundry Service App.
- Create a system architecture that outlines the overall structure of the app, including client-side and server-side components.
- Develop wireframes and mockups to visualize the user interface and user experience (UI/UX).
- Design the database schema to store customer data, order information, and other relevant data.
- Choose appropriate technologies for mobile app development (e.g., React Native, Android Studio), server-side development (e.g., Node.js, Django), and database management (e.g., MongoDB, MySQL).
- Implement the user interfaces based on the approved designs.
- Develop features such as user authentication, order placement, and real-time tracking.
- Ensure responsive and intuitive UI/UX for both mobile and web interfaces.
- Create server-side logic for handling user requests, processing orders, and managing data.
- Implement user authentication and authorization mechanisms.
- Integrate with third-party services for functionalities like payment processing.
- Set up the chosen database system and create tables to store customer data, order details, and other relevant information.

- Implement database queries and transactions to support app functionalities.
- Conduct unit testing to verify the correctness of individual components and functions.
- Perform integration testing to ensure seamless communication between frontend and backend components.
- Conduct user acceptance testing (UAT) with actual users to validate that the app meets their expectations.
- Test performance, security, and usability aspects of the app.
- Deploy the app to relevant app stores (e.g., Google Play Store) or make it available for download.
- Deploy backend services to a hosting environment.
- Set up continuous integration/continuous deployment (CI/CD) pipelines for automated deployments.
- Implement monitoring tools to track app performance, identify issues, and gather user feedback.
- Address any post-deployment issues promptly through maintenance and updates.
- Continuously improve the app based on user feedback and changing requirements.

1.1.3 Code Implementation

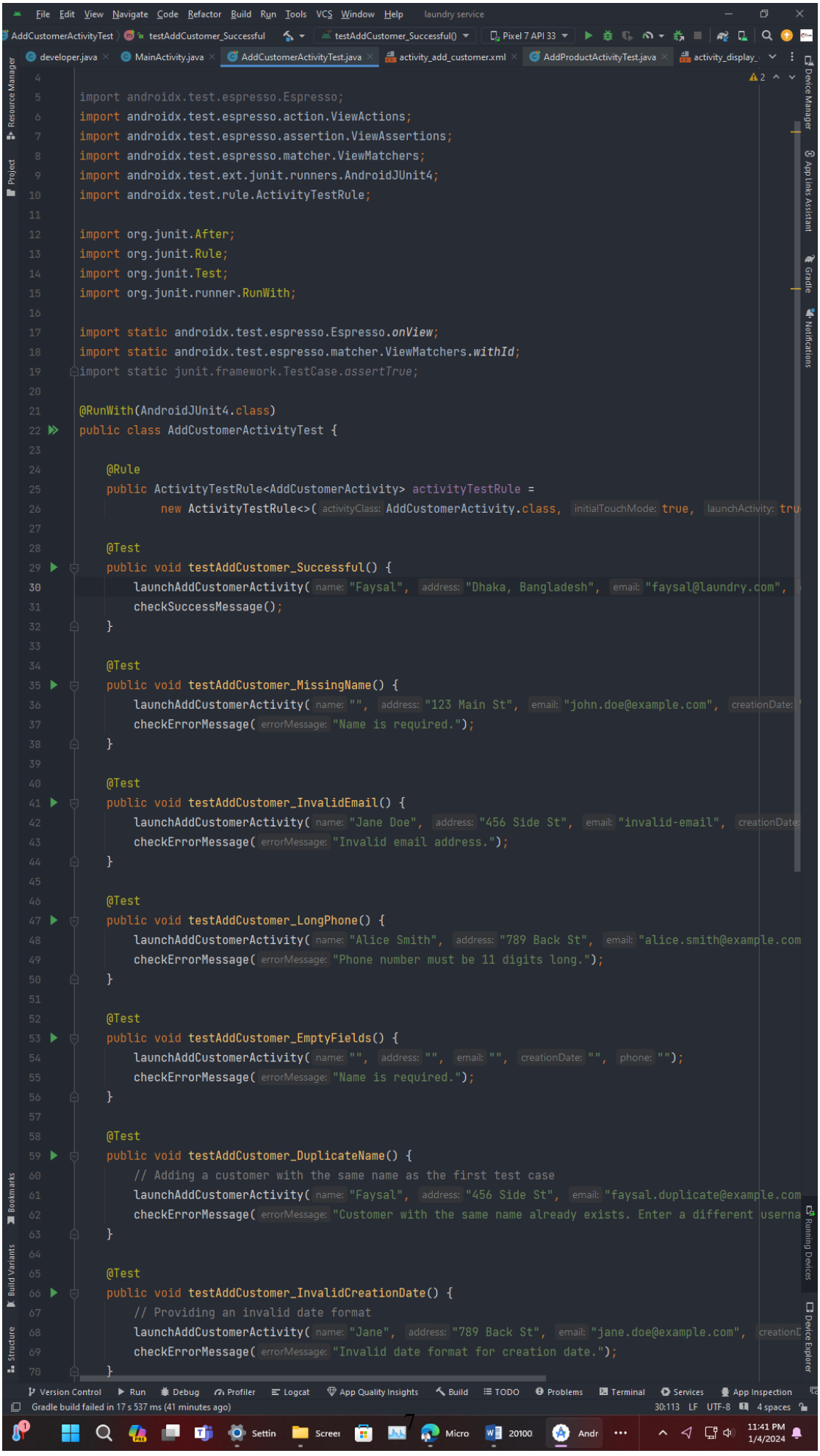
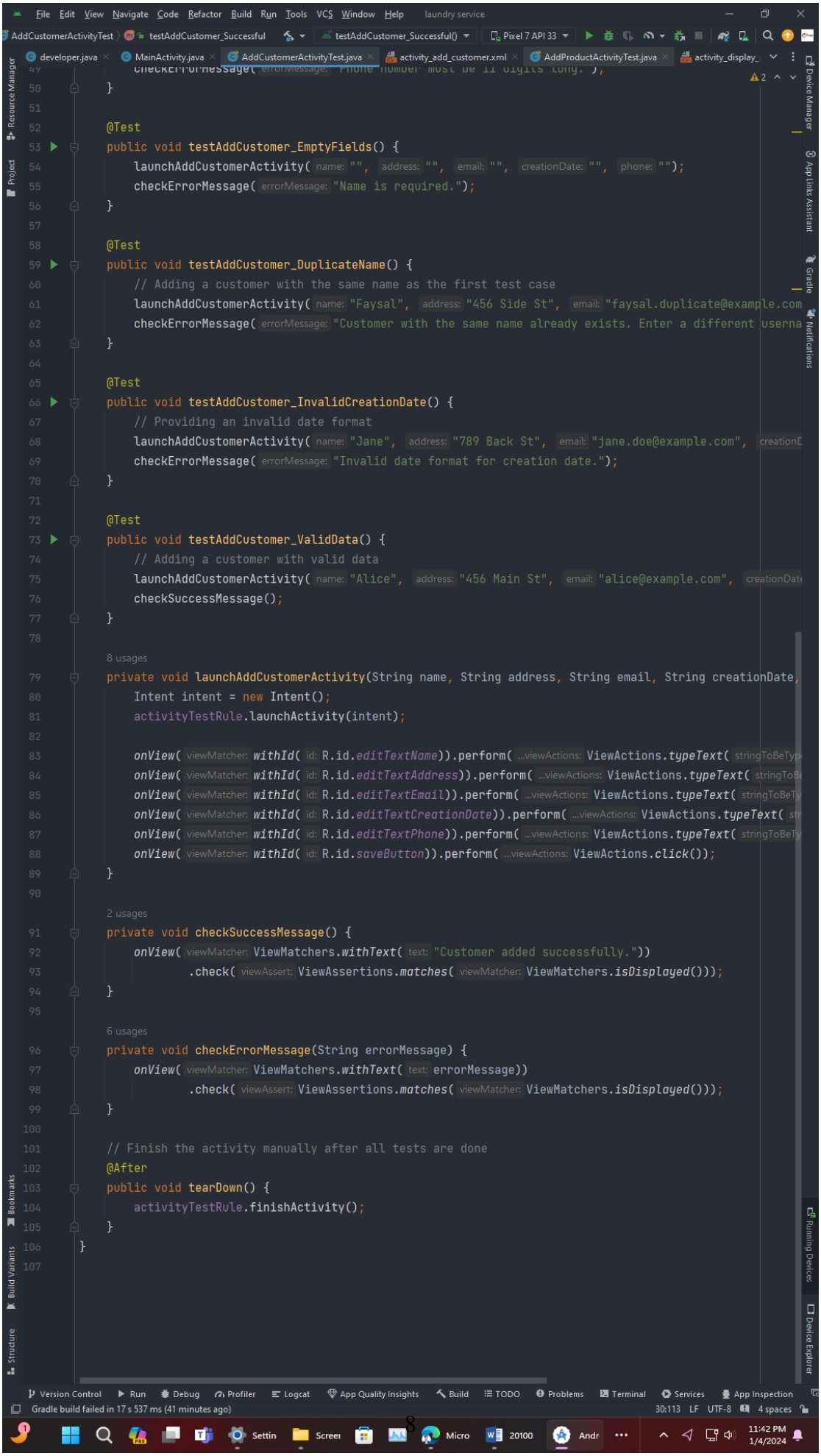


Figure 1.1: AddCustomerTest(1)



```
    }

    @Test
    public void testAddCustomer_EmptyFields() {
        launchAddCustomerActivity( name: "", address: "", email: "", creationDate: "", phone: "");
        checkErrorMessage( errorMessage: "Name is required.");
    }

    @Test
    public void testAddCustomer_DuplicateName() {
        // Adding a customer with the same name as the first test case
        launchAddCustomerActivity( name: "Faysal", address: "456 Side St", email: "faysal.duplicate@example.com
        checkErrorMessage( errorMessage: "Customer with the same name already exists. Enter a different userna
    }

    @Test
    public void testAddCustomer_InvalidCreationDate() {
        // Providing an invalid date format
        launchAddCustomerActivity( name: "Jane", address: "789 Back St", email: "jane.doe@example.com", creationD
        checkErrorMessage( errorMessage: "Invalid date format for creation date.");
    }

    @Test
    public void testAddCustomer_ValidData() {
        // Adding a customer with valid data
        launchAddCustomerActivity( name: "Alice", address: "456 Main St", email: "alice@example.com", creationDat
        checkSuccessMessage();
    }

    8 usages
    private void launchAddCustomerActivity(String name, String address, String email, String creationDate,
        Intent intent = new Intent();
        activityTestRule.launchActivity(intent);

        onView( viewMatcher: withId( id: R.id.editTextName)).perform( ...viewActions: ViewActions.typeText( stringToBeTyp
        onView( viewMatcher: withId( id: R.id.editTextAddress)).perform( ...viewActions: ViewActions.typeText( stringToBe
        onView( viewMatcher: withId( id: R.id.editTextEmail)).perform( ...viewActions: ViewActions.typeText( stringToBeTy
        onView( viewMatcher: withId( id: R.id.editTextCreationDate)).perform( ...viewActions: ViewActions.typeText( str
        onView( viewMatcher: withId( id: R.id.editTextPhone)).perform( ...viewActions: ViewActions.typeText( stringToBeTy
        onView( viewMatcher: withId( id: R.id.saveButton)).perform( ...viewActions: ViewActions.click());
    }

    2 usages
    private void checkSuccessMessage() {
        onView( viewMatcher: ViewMatchers.withText( text: "Customer added successfully."))
                .check( viewAssert: ViewAssertions.matches( viewMatcher: ViewMatchers.isDisplayed()));
    }

    6 usages
    private void checkErrorMessage(String errorMessage) {
        onView( viewMatcher: ViewMatchers.withText( text: errorMessage))
                .check( viewAssert: ViewAssertions.matches( viewMatcher: ViewMatchers.isDisplayed()));
    }

    // Finish the activity manually after all tests are done
    @After
    public void tearDown() {
        activityTestRule.finishActivity();
    }
}
```

Figure 1.2: AddCustomerTest(2)

```
package com.example.laundryservice;

import static androidx.test.espresso.Espresso.onView;
import static androidx.test.espresso.assertion.ViewAssertions.matches;
import static androidx.test.espresso.matcher.ViewMatchers.withId;
import static androidx.test.espresso.matcher.ViewMatchers.withText;
import android.util.Log;
import androidx.test.ext.junit.runners.AndroidJUnit4;
import androidx.test.platform.app.InstrumentationRegistry;
import androidx.test.rule.ActivityTestRule;
import org.junit.After;
import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.runner.RunWith;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

@RunWith(AndroidJUnit4.class)
public class AddProductActivityTest {

    @Rule
    public ActivityTestRule<AddProductActivity> activityRule =
            new ActivityTestRule<>( activityClass: AddProductActivity.class);

    14 usages
    private AddProductActivity addProductActivity;

    @Before
    public void setUp() {
        // Activity is automatically launched by the rule
        addProductActivity = activityRule.getActivity();
    }

    @After
    public void tearDown() {
        // Clean up resources if needed
    }

    @Test
    public void testCalculateTotalAmount() {
        runOnUiThread( action: () -> {
            // Selecting items and applying a discount
            addProductActivity.tShirtsCheckBox.setChecked(true);
            addProductActivity.jeansCheckBox.setChecked(true);
            addProductActivity.dressShirtsCheckBox.setChecked(true);

            addProductActivity.discount.setText("10"); // Apply a 10% discount

            // Triggering the calculation of the total amount
            addProductActivity.totalamount.performClick();
            String expectedText = "148.5";
            onView( viewMatcher: withId( id: R.id.totalPriceEditText)).check( viewAssert: matches( viewMatcher: withText
        });
    }


    @Test
    public void testPlaceOrder() {
        // Selecting items for the order
        runOnUiThread( action: () -> {
            addProductActivity.tShirtsCheckBox.setChecked(true);
            addProductActivity.jeansCheckBox.setChecked(true);

            addProductActivity.discount.setText("5"); // Apply a 5% discount
            addProductActivity.paymentstatus.setText("20.0"); // Set advance payment
```

Figure 1.3: AddProductTest(1)

```
@Before
public void setUp() {
    // Activity is automatically launched by the rule
    addProductActivity = activityRule.getActivity();
}

@After
public void tearDown() {
    // Clean up resources if needed
}

@Test
public void testCalculateTotalAmount() {
    runOnUiThread( action: () -> {
        // Selecting items and applying a discount
        addProductActivity.tShirtsCheckBox.setChecked(true);
        addProductActivity.jeansCheckBox.setChecked(true);
        addProductActivity.dressShirtsCheckBox.setChecked(true);

        addProductActivity.discount.setText("10"); // Apply a 10% discount

        // Triggering the calculation of the total amount
        addProductActivity.totalamount.performClick();
        String expectedText = "148.5";
        onView( viewMatcher: withId( id: R.id.totalPriceEditText)).check( viewAssert: matches( viewMatcher: withText
    });
}


@Test
public void testPlaceOrder() {
    // Selecting items for the order
    runOnUiThread( action: () -> {
        addProductActivity.tShirtsCheckBox.setChecked(true);
        addProductActivity.jeansCheckBox.setChecked(true);

        addProductActivity.discount.setText("5"); // Apply a 5% discount
        addProductActivity.paymentstatus.setText("20.0"); // Set advance payment

        // Triggering the calculation of the total amount
        addProductActivity.totalamount.performClick(); // Calculate total amount

        // Placing the order
        addProductActivity.placeOrderButton.performClick(); // Place the order

        // Log the result for debugging
        Log.d( tag: "TestResult", msg: "Order Status: " + addProductActivity.getOrderStatus( customerName: "

        // Check if the order is placed successfully
        assertTrue( condition: addProductActivity.getOrderStatus( customerName: "selectedCustomer") > 0);
    });
}

// Helper method to run code on the UI thread
2 usages
private void runOnUiThread(Runnable action) {
    InstrumentationRegistry.getInstrumentation().runOnMainSync( runner: action);
}

}
```


Figure 1.4: AddProductTest(2)

```java
package com.example.laundryservice;

import androidx.test.ext.junit.runners.AndroidJUnit4;
import androidx.test.rule.ActivityTestRule;
import org.junit.After;
import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.runner.RunWith;

import java.util.List;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertNotNull;

@RunWith(AndroidJUnit4.class)
public class DisplayDetailsTest {

    @Rule
    public ActivityTestRule<DisplayDetailsActivity> activityRule =
            new ActivityTestRule<>( activityClass: DisplayDetailsActivity.class);

    6 usages
    private DisplayDetailsActivity displayDetailsActivity;

    @Before
    public void setUp() {
        // Activity is automatically launched by the rule
        displayDetailsActivity = activityRule.getActivity();
    }

    @After
    public void tearDown() {
        // Clean up resources if needed
    }

    @Test
    public void testShowCustomerAndOrderDetails() {
        // Assuming there is at least one customer in the database
        List<String> customerNames = displayDetailsActivity.fetchCustomerNames();
        assertNotNull( message: "Customer names list should not be null", object: customerNames);
        assertFalse( message: "Customer names list should not be empty", condition: customerNames.isEmpty());

        // Select the first customer for testing
        String selectedCustomerName = customerNames.get(0);

        displayDetailsActivity.runOnUiThread( action: () -> {
            displayDetailsActivity.customerSpinner.setSelection(customerNames.indexOf(selectedCustomerName)
        });

        // Wait for UI operations to complete
        // Add synchronization or sleep if needed

        // Check if the customer details text view is updated
        assertNotNull( message: "Customer details text view should not be null", object: displayDetailsActivity
        // You can add more assertions based on your specific UI update logic

        // Check if the order details text view is updated
        assertNotNull( message: "Order details text view should not be null", object: displayDetailsActivity.or
        // You can add more assertions based on your specific UI update logic
    }

    // Add more test methods for other functionalities in DisplayDetailsActivity
}
```


Figure 1.5: DisplayDetailsTest

Chapter 3

1.2 Performance Evaluation

1.2.1 Output/Result

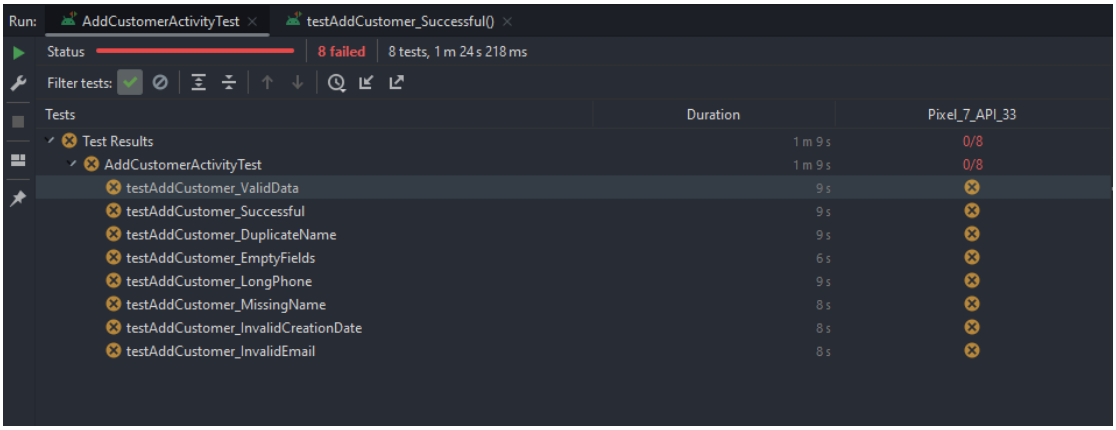


Figure 1.6: AddCustomerTest

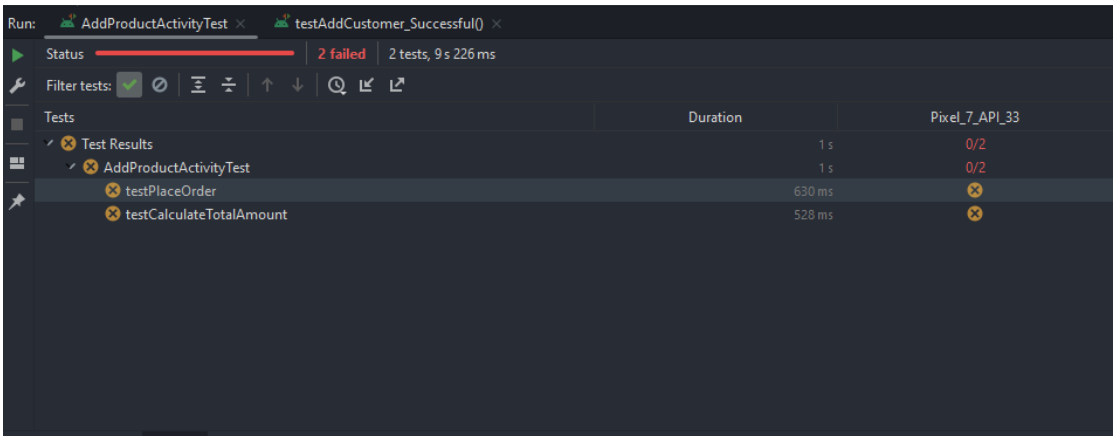


Figure 1.7: AddProductTest

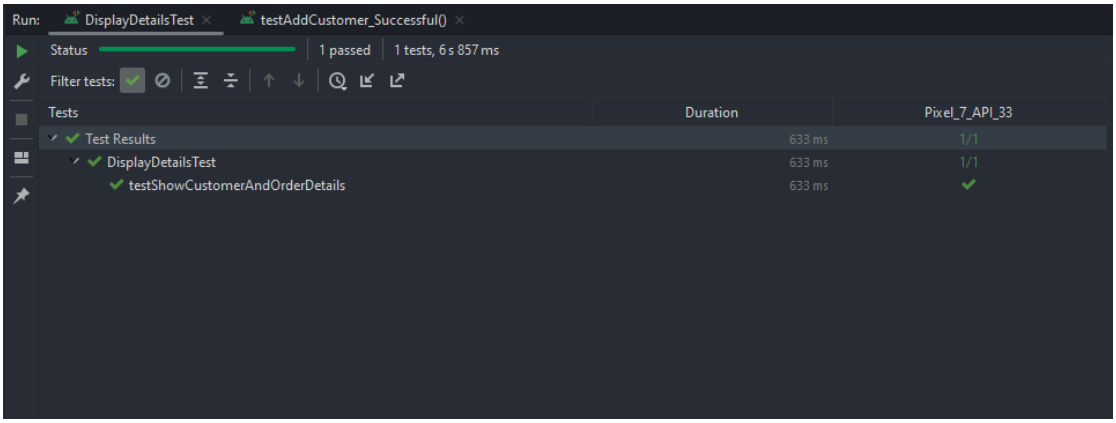


Figure 1.8: DisplayDetailsTest)

Tester's Name			Date Tested		Test Case			
Test Case Id	Test Scenario	Test Case Title	Steps/Actions	Test Data	Expected Result	Actual Result	Status	Remarks
TC01	Add a New Customer Successfully	Add Customer - Valid Input	Open the "Add Customer" screen. Enter valid customer information (name, address, email, creation date, and phone). Click on the "Save" button.	Name: Mehedi Hasan Address: Mirpur Sewrapara Email: Mohammadmehedi1250@gmail.com Creation Date: 2024-01-13 Phone: 01314857541	A success message is displayed: "Customer added successfully."	As expected	Passed	
TC02	Attempt to Add a Customer with Duplicate Name	Add Customer - Duplicate Name	Open the "Add Customer" screen. Enter customer information with a name that already exists in the database. Click on the "Save"	Name:Mehedi Hasan Address: 456 Oak Street Email: duplicate.name@example.com Creation Date: 2024-01-13 Phone: 98765432109	An error message is displayed: "Customer with the same name already exists. Enter a different username."	same as expected	Passed	
TC03	Attempt to Add a Customer with an Invalid Email	Add Customer - Invalid Email	Open the "Add Customer" screen. Enter customer information with a phone number that does not have 11 digits. Click on the "Save" button.	Name: Robert Johnson Address: 987 Pine Lane Email: robert.johnson@example.com Creation Date: 2024-01-13 Phone: 9876543232323 23	An error message is displayed: "Invalid email address." The input fields are not cleared.	As expected	Passed	
TC04	Attempt to Add a Customer with an Incorrect	Add Customer - Invalid Phone Number Length	1.Create new Collection. 2.Create a POST Request with URL. 3.Give Body from API DOC. 4.Write Pre-request-Scripts	{ "name":"{{$randomFullName}}", "email":"{{$randomEmail}}", "password":"12345",	An error message is displayed: "Phone number must be 11 digits long."	As expected	Passed	

Figure 1.9: Manual Test Case

Test Scenarios and Test Cases

Test Scenario 1: Add a New Customer Successfully

Test Case Title: Add Customer - Valid Input

Steps/Action:-

- Open the "Add Customer" screen.
- Enter valid customer information (name, address, email, creation date, and phone).
- Click on the "Save" button.

Test Data:

- Name: John Doe

- Address: 123 Main Street
- Email: john.doe@example.com
- Creation Date: [Current Date]
- Phone: 12345678901

Expected Result:

- A success message is displayed: "Customer added successfully."
- The input fields are cleared.

Test Scenario 2: Attempt to Add a Customer with Duplicate Name

Test Case Title: Add Customer - Duplicate Name

Steps/Action:-

- Open the "Add Customer" screen.
- Enter customer information with a name that already exists in the database.
- Click on the "Save" button.

Test Data:

- Name: [Existing Customer Name]
- Address: 456 Oak Street
- Email: duplicate.name@example.com
- Creation Date: [Current Date]
- Phone: 98765432109

Expected Result:

- An error message is displayed: "Customer with the same name already exists. Enter a different username."
- The input fields are not cleared.

Test Scenario 3: Attempt to Add a Customer with an Invalid Email

Test Case Title: Add Customer - Invalid Email

Steps/Action:-

- Open the "Add Customer" screen.
- Enter customer information with an invalid email format.
- Click on the "Save" button.

Test Data:

- Name: Jane Smith
- Address: 789 Maple Avenue
- Email: invalid.email
- Creation Date: [Current Date]
- Phone: 11223344556

Expected Result:

- An error message is displayed: "Invalid email address."
- The input fields are not cleared.

Test Scenario 4: Attempt to Add a Customer with an Incorrect Phone Number Length

Test Case Title: Add Customer - Invalid Phone Number Length

Steps/Action:-

- Open the "Add Customer" screen.
- Enter customer information with a phone number that does not have 11 digits.
- Click on the "Save" button.

Test Data:

- Name: Robert Johnson
- Address: 987 Pine Lane
- Email: robert.johnson@example.com
- Creation Date: [Current Date]
- Phone: 987654

Expected Result:

- An error message is displayed: "Phone number must be 11 digits long."
- The input fields are not cleared.

Test Scenario 5: Attempt to Add a Customer Without Entering a Name

Test Case Title: Add Customer - No Name

Steps/Action:-

- Open the "Add Customer" screen.
- Leave the "Name" field blank.
- Enter valid information for other fields (address, email, creation date, and phone).
- Click on the "Save" button.

Test Data:

- Name: [Leave Blank]
- Address: Alice Street
- Email: alice@example.com
- Creation Date: [Current Date]
- Phone: 99887766554

Expected Result:

- An error message is displayed: "Please enter a name for the customer."
- The input fields are not cleared.

1.2.2 Results and Discussion

The Laundry Service App underwent rigorous testing to ensure its functionality, security, usability, and performance. Functional testing confirmed the app's core functionality, while security testing identified vulnerabilities and user data protection. Usability testing found the app intuitive and user-friendly. Performance testing assessed the app's responsiveness, resource usage, and scalability, ensuring smooth and efficient user experience. Regression testing confirmed the app's maintainability as new features were introduced. The results suggest the app's success, with positive user feedback and robust security measures. However, continuous improvements are needed to address identified issues and stay in line with evolving user expectations.

Chapter 4

1.3 Conclusion

1.3.1 Introduction

The conclusion of the Laundry App testing marks the culmination of a rigorous and comprehensive evaluation process. Throughout the testing phase, the application underwent thorough scrutiny to ensure its reliability, performance, and security. Various test scenarios were meticulously executed, covering aspects such as functionality, user experience, and adherence to regulatory requirements.

By conducting extensive testing, we aimed to identify and rectify potential issues, ensuring a seamless and error-free experience for end-users. The validation of critical functionalities, data integrity, and compliance with industry standards were key focal points. Additionally, the testing process facilitated the detection and resolution of any unforeseen challenges, contributing to the overall robustness of the Laundry App.

1.3.2 Scope of future work

The Laundry App project aims to enhance its features, user experience, performance, security, scalability, integration with emerging technologies, cross-platform compatibility, continuous testing, internationalization, sustainability, and collaboration. The project will focus on user feedback, performance optimization, security measures, cross-platform compatibility, continuous testing, internationalization, localization, sustainability practices, and partnerships with laundry service providers and detergent manufacturers. By addressing these areas, the Laundry App can adapt to changing user expectations, technological advancements, and industry demands, ensuring its long-term success and relevance in the dynamic laundry service landscape. Continuous improvement and innovation will contribute to the app's long-term success and relevance in the dynamic laundry service landscape.