# Exercise 2:

**Scenario 1:**

**For Automating the online application using Java programming language with Appium to complete the E2E flow with Page object model approach.**

**//BaseTest.java - This class will handle the setup part, Test cases execution and Tear down part for the application.**

import io.appium.java\_client.AppiumDriver;

import io.appium.java\_client.android.AndroidDriver;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.junit.After;

import org.junit.Before;

public class BaseTest {

protected AppiumDriver driver;

protected LoginPage loginPage;

protected ShowcaseScreenr showcaseScreen;

protected CartScreen cartScreenr ;

protected Address&PurchaseScreen address&PurchaseScreen ;

**@Before**

public void setUp() throws Exception {

//desiredCap for setUp

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability("platformName", "Android");

capabilities.setCapability("deviceName", "Android Emulator");

capabilities.setCapability("appPackage", "com.mercari.app");

capabilities.setCapability("appActivity", "com.mercari.app.MainActivity");

//localhost

driver = new AndroidDriver<>(new URL("http://127.0.0.1:4723/wd/hub"), capabilities);

}

**@Test**

Public void successfulItemsPurchase () throws Exception {

**//login page**

loginPage.login(“Valid-User”,”TestPassword”);

//wait for the screen to load

waitForScreenLoad(TextMatcher.withText(“Cart”));  
}

**//ShowScreen page**

showcaseScreen.openFirstItem();

itemScreen.addToCart();

waitForScreenLoad(TextMatcher.withText(“Showcase”));

showcaseScreen.openSecondItem();

itemScreen.addToCart();

waitForScreenLoad(TextMatcher.withText(“Cart”));

**//CartScreen** - Bonus one

verifyItemQuantity(2) - // To verify the Quantity of Items added to Cart

**//Proceed to checkout, address and Confirm purchase**

cartScreen.startPurchase();

address&PurchaseScreen.filladdress(“TestName”, “Test Address”, “Test Number”);

address&PurchaseScreen.ClickContinueButton();

waitForScreenLoad(TextMatcher.withText(“Confirm purchase”));

//Confirm Purchase

address&PurchaseScreen.ClickConfirmPurchaseButton();

//Confirm Success message

address&PurchaseScreen.verifySuccessMessage(“Success”);

**@After**

//Performs cleanUp after a test case execution completes.

public void tearDown() {

if (driver != null) {

driver.quit();

}

}

}

**PageObjects:**

**//Each screen is separated by page object class for code re-usability.**

**Login screen:**

import io.appium.java\_client.AppiumDriver;

import io.appium.java\_client.MobileElement;

import org.openqa.selenium.By;

import org.openqa.selenium.support.ui.WebDriverWait;

//All methods extends BaseClass to perform setUp() using @Before and TearDown using @After attributes.

public class LoginScreen extends BaseScreen {

private By usernameBar = By.id("com.example.app:id/username");

private By passwordBar = By.id("com.example.app:id/password");

private By loginButton = By.id("com.example.app:id/login\_button");

public LoginScreen(AppiumDriver driver) {

//Always points to the main driver

super(driver);

}

//Method for login test case

public void loginPage(String username, String password) {

MobileElement usernameEl = (MobileElement) driver.findElement(usernameBar);

MobileElement passwordEl = (MobileElement) driver.findElement(passwordBar);

MobileElement loginButtonEl = (MobileElement) driver.findElement(loginButton);

usernameEl.sendKeys(username);

passwordEl.sendKeys(password);

loginButtonEl.click();

}

}

**ShowCaseScreen**:

import io.appium.java\_client.AppiumDriver;

import io.appium.java\_client.MobileElement;

import org.openqa.selenium.By;

public class ShowcaseScreen extends BaseScreen {

private By firstItem = By.id("com.example.app:id/first\_item");

private By secondItem = By.id("com.example.app:id/second\_item");

public ShowcaseScreen(AppiumDriver driver) {

super(driver);

}

public void openFirstItem() {

MobileElement firstItem = (MobileElement) driver.findElement(firstItem);

firstItem.click();

}

public void openSecondItem() {

MobileElement secondItem = (MobileElement) driver.findElement(secondItem);

secondItem.click();

}

}

**ItemScreen**  
  
import io.appium.java\_client.AppiumDriver;

import io.appium.java\_client.MobileElement;

import org.openqa.selenium.By;

public class ItemScreen extends BaseScreen {

private By addToCartButton = By.id("com.example.app:id/add\_to\_cart\_button");

public ItemScreen(AppiumDriver driver) {

super(driver);

}

public void addToCart() {

MobileElement addToCart = (MobileElement) driver.findElement(addToCartButton);

addToCart.click();

}

}

**CartScreen:**

import io.appium.java\_client.AppiumDriver;

import io.appium.java\_client.MobileElement;

import org.openqa.selenium.By;

public class CartScreen extends BaseScreen {

private By cartItemCount = By.id("com.example.app:id/cart\_item\_count");

private By startPurchaseButton = By.id("com.example.app:id/start\_purchase\_button");

public CartScreen(AppiumDriver driver) {

super(driver);

}

//Methods verifies the item quantity and throws error message in console on not matching.

public void verifyItemQuantity(int expectedQuantity) {

MobileElement itemCountEl = (MobileElement) driver.findElement(cartItemCount);

int itemCount = Integer.parseInt(itemCountEl.getText());

if (itemCount != expectedQuantity) {

throw new AssertionError("Expected item count: " + expectedQuantity + " but got: " + itemCount);

}

}

public void startPurchase() {

MobileElement startPurchaseButtonEl = (MobileElement) driver.findElement(startPurchaseButton);

startPurchaseButtonEl.click();

// Wait for transition to Address Screen

WaitUtils.waitForScreenToLoad(driver, "address\_screen", 10);

}

}

**Address Screen and Purchase Screen:**

import io.appium.java\_client.AppiumDriver;

import io.appium.java\_client.MobileElement;

import org.openqa.selenium.By;

public class Address&PurchaseScreen extends BaseScreen {

private By fillAddressButton = By.name("Address Space");

private By continueButton = By.name("ContinuePurchase");

private By PurchaseButton = By.name("Purchase");

Private By SuccessMessageString = By.name(“com.success.message”)

public CartScreen(AppiumDriver driver) {

super(driver);

**}**

public void filladdress(String name, String Address, String number){

MobileElement fillAddress = (MobileElement) driver.findElement(fillAddressButton);

firstItem.sendKeys(name,Address,number);

}

public void ClickContinueButton() {

MobileElement continueOption = (MobileElement) driver.findElement(ContinueButton);

continueOption.click();

}

public void ClickConfirmPurchaseButton() {

MobileElement ContinuePurchase = (MobileElement) driver.findElement(PurchaseButton);

ContinuePurchase.click();

}

Public void verifySuccessMessage(String messageString) {

MobileElement successMessage = (MobileElement) driver.findElement(SuccessMessageString);

String message = successMessagel.getText());

if (!message.equalsIgnorecase( messageString)) {

throw new AssertionError("Expected message is: " +messageString + " but got: " + message );

}

**Scenario 2:**

1. **To handle screen transitions:**

we use **WaitUtils.waitForScreenToLoad(**) to handle slow network conditions and ensure that we are interacting with the correct screen before proceeding.

i. Internal Method: waitForScreenToLoad() method is implemented in WaitUtils.java and is used across different screen classes accordingly.

ii. Logging: The screen that has been loaded is logged to the console each time the method completes successfully.

1. **Before Test starts:**

Yes it is vital, where we have setUp.java() method before the actual test begins, where we initialize the Appium driver and also here app can be mocked to test slow network condition. Also if needed reporter, date and Device details can be recorded before the test begin to display in end report.

1. **Cart Quantity Assertion:**

The cart item count is important to ensure the items are correctly added to the cart. The verifyItemQuantity() method in CartScreen.java ensures that the cart contains the expected number of items as added by user, also it throws assertion error on failing to match the exact value in console, which helps user to move forward to purchase without mismatch in items added .