```java
package ecommerceautomation.tests;

import ecommerceautomation.base.BaseTest;
import ecommerceautomation.pages.CartPage;
import ecommerceautomation.pages.HomePage;
import ecommerceautomation.pages.ProductPage;
import org.testng.Assert;
import org.testng.annotations.Test;

public class CartTest extends BaseTest {

    @Test
    public void verifyCartOperations() {
        // Step 1: Navigate to the Home Page (handled by BaseTest @BeforeMethod)
        HomePage homePage = new HomePage(getDriver(), wait);

        // Step 2: Navigate to Products Page and add a product to the cart
        ProductPage productPage = homePage.navigateToProductsPage();
        String productName = "Blue Top";
        CartPage cartPage = productPage.searchProduct(productName);

        // Step 3: Verify product is in cart and its quantity
        Assert.assertTrue(cartPage.isProductInCart(productName),
                          "Expected product to be present in the cart: " + productName);

        // This assertion might be redundant if the test flow only adds one product.
        // int quantity = cartPage.getProductQuantity();
        // Assert.assertEquals(quantity, 1, "Expected cart quantity to be 1.");

        // Step 4: Remove the product
        cartPage.removeProduct(productName);

        // Step 5: Verify product was removed
        Assert.assertFalse(cartPage.isProductInCart(productName),
                           "Product should no longer be in the cart after removal: " + pr
oductName);
    }

    @Test
    public void proceedToCheckoutFlow() {
        // Step 1: Navigate to Products Page and add a product to the cart
        HomePage homePage = new HomePage(getDriver(), wait);
        ProductPage productPage = homePage.navigateToProductsPage();
        String productName = "Blue Top";
        CartPage cartPage = productPage.searchProduct(productName);

        // Step 2: Proceed to checkout from the Cart Page
        cartPage.proceedToCheckout();

        // Step 3: Verify the redirection
        String currentUrl = getDriver().getCurrentUrl();
        Assert.assertTrue(currentUrl.contains("checkout") || currentUrl.contains("login")
,
                "After clicking 'Proceed to Checkout', the user should be on the checkout
 or login page. Current URL: " + currentUrl);
    }

}
```

```java
package ecommerceautomation.tests;

import ecommerceautomation.base.BaseTest;
import ecommerceautomation.pages.HomePage;
import ecommerceautomation.pages.ProductPage;
import ecommerceautomation.pages.LoginPage;
import org.testng.Assert;
import org.testng.annotations.Test;

public class HomePageTest extends BaseTest {

    @Test
    public void verifyUserIsLoggedIn() {
        // Step 1: Assume a successful login has occurred and we are on the home page.
        // For a real test, you would perform the login first.
        // E.g., LoginPage loginPage = new LoginPage(getDriver(), wait);
        // HomePage homePage = loginPage.login("test@example.com", "password123");

        // Step 2: Instantiate the HomePage object
        HomePage homePage = new HomePage(getDriver(), wait);

        // Step 3: Assert that the user is logged in
        Assert.assertTrue(homePage.isUserLoggedIn(), "User should be logged in and the 'logged-in' indicator should be visible.");
    }

    @Test
    public void navigateToProductsPage() {
        // Step 1: Start on the home page
        HomePage homePage = new HomePage(getDriver(), wait);

        // Step 2: Navigate to the products page using the HomePage method
        ProductPage productPage = homePage.navigateToProductsPage();

        // Step 3: Assert that the URL has changed to the products page URL
        Assert.assertTrue(getDriver().getCurrentUrl().contains("products"), "URL should contain 'products' after navigating to the products page.");

        // Optional Step 4: Verify a unique element on the ProductPage to confirm navigation
        // Assert.assertTrue(productPage.isProductsHeaderVisible(), "Products header should be visible.");
    }

    @Test
    public void verifyLogoutFunctionality() {
        // Step 1: Assume a user is logged in
        // In a real scenario, this would follow a successful login test.
        HomePage homePage = new HomePage(getDriver(), wait);

        // Step 2: Perform the logout action
        LoginPage loginPage = homePage.logout();

        // Step 3: Assert that the user has been logged out by checking for a login-related element
        Assert.assertTrue(loginPage.isLoginButtonVisible(), "After logout, the login button should be visible.");
    }
}
```

```java
package ecommerceautomation.tests;

import ecommerceautomation.base.BaseTest;
import ecommerceautomation.pages.CartPage;
import ecommerceautomation.pages.HomePage;
import ecommerceautomation.pages.ProductPage;
import org.testng.Assert;
import org.testng.annotations.Test;

public class ProductTest extends BaseTest {

    @Test
    public void searchAndAddProductToCart() {
        // Step 1: Start from the home page (handled by BaseTest @BeforeMethod)
        HomePage homePage = new HomePage(getDriver(), wait);

        // Step 2: Navigate to the products page, search for a product, and go to the car
t page
        String productName = "Blue Top";
        CartPage cartPage = homePage.navigateToProductsPage().searchProduct(productName);

        // Step 3: Verify that the product is in the cart
        Assert.assertTrue(cartPage.isProductInCart(productName),
                          "Product should be in the cart");
    }
}
```

```java
package ecommerceautomation.tests;

import ecommerceautomation.base.BaseTest;
import ecommerceautomation.pages.HomePage;
import ecommerceautomation.pages.LoginPage;
import ecommerceautomation.utils.ExcelUtils;
import org.testng.Assert;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class LoginTest extends BaseTest {

    @DataProvider(name = "validLoginData")
    public Object[][] getValidLoginData() throws Exception {
        ExcelUtils excel = new ExcelUtils(config.getProperty("user.data.path"), "ValidLog
inUsers");
        return excel.getAllRowsAsObjectArray();
    }

    @DataProvider(name = "invalidLoginData")
    public Object[][] getInvalidLoginData() throws Exception {
        ExcelUtils excel = new ExcelUtils(config.getProperty("user.data.path"), "InvalidL
oginUsers");
        return excel.getAllRowsAsObjectArray();
    }

    @Test(dataProvider = "validLoginData")
    public void validUserLoginTest(String email, String password) {
        LoginPage loginPage = new LoginPage(getDriver(), wait);

        Object nextPage = loginPage.login(email, password);

        Assert.assertTrue(nextPage instanceof HomePage, "Login failed for valid user.");

        HomePage homePage = (HomePage) nextPage;
        Assert.assertTrue(homePage.isUserLoggedIn(), "User is not logged in after success
ful login.");
        homePage.logout();
    }

    @Test(dataProvider = "invalidLoginData")
    public void invalidUserLoginTest(String email, String password, String expectedMessag
e) {
        LoginPage loginPage = new LoginPage(getDriver(), wait);

        Object nextPage = loginPage.login(email, password);

        Assert.assertTrue(nextPage instanceof LoginPage, "Login succeeded for invalid use
r.");

        LoginPage returnedLoginPage = (LoginPage) nextPage;
        String actualMessage = returnedLoginPage.getLoginErrorMessage();
        Assert.assertEquals(actualMessage, expectedMessage, "Incorrect error message for
invalid login.");
    }
}
```

```java
package ecommerceautomation.tests;
import ecommerceautomation.base.BaseTest;
import ecommerceautomation.pages.SignupPage;
import ecommerceautomation.pages.AccountInformationPage;
import ecommerceautomation.pages.HomePage;
import ecommerceautomation.pages.LoginPage;
import ecommerceautomation.utils.ExcelUtils;
import org.testng.Assert;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class SignupPageTest extends BaseTest {

    @DataProvider(name = "registerUserData")
    public Object[][] getRegisterUserData() throws Exception {
        ExcelUtils excel = new ExcelUtils(config.getProperty("user.data.path"), "Register
Users");
        return excel.getAllRowsAsObjectArray();
    }

    @Test(dataProvider = "registerUserData")
    public void registerNewUserTest(String name, String email, String password, String ti
tle, String day, String month,
                                    String year, String firstname, String lastname,
String company, String address,
                                    String country, String state, String city, Strin
g zip, String mobileNo) {

        SignupPage signupPage = new SignupPage(getDriver(), wait);
        signupPage.clickOnSignupLoginLink();
        signupPage.enterNameAtSignUp(name);
        signupPage.enterEmailAtSignUp(email);

        // This check is to ensure test data is clean.
        if (signupPage.isEmailAlreadyRegistered()) {
                System.out.println("email already resgistered");
                Assert.fail("Test data issue: The email " + email + " is already register
ed. Please use a unique email for a new user test.");
        }

        AccountInformationPage accountInfoPage = signupPage.clickOnSignupButton();
        accountInfoPage.enterAccountInformation(title, password, day, month, year);
        accountInfoPage.enterAddressInformation(firstname, lastname, company, address, co
untry, state, city, zip, mobileNo);

        HomePage homePage = accountInfoPage.clickOnCreateAccountButton();

        homePage.logout();
    }
}
```

```java
package ecommerceautomation.tests;

import ecommerceautomation.base.BaseTest;
import ecommerceautomation.pages.*;
import ecommerceautomation.utils.ExcelUtils;
import org.testng.Assert;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class EcommerceShoppingTest extends BaseTest {

    @DataProvider(name = "loginData")
    public Object[][] getLoginData() throws Exception {
        ExcelUtils excel = new ExcelUtils(config.getProperty("user.data.path"), "LoginUse
rs");
        return excel.getAllRowsAsObjectArray();
    }

    @Test(groups = { "regression" }, dataProvider = "loginData")
    public void fullEndToEndFlow(String email, String password, String expectedResult) {
        // Step 1: Login with valid credentials
        if (expectedResult.equalsIgnoreCase("success")) {
            LoginPage loginPage = new LoginPage(getDriver(), wait);
            Object nextPage = loginPage.login(email, password);
            Assert.assertTrue(nextPage instanceof HomePage, "Expected HomePage object on
successful login.");

            HomePage homePage = (HomePage) nextPage;

            // Step 2: Add a product to the cart
            ProductPage productPage = homePage.navigateToProductsPage();
            String productName = "Blue Top";
            CartPage cartPage = productPage.searchProduct(productName);

            // Step 3: Verify the product is in the cart
            Assert.assertTrue(cartPage.isProductInCart(productName),
                    "Expected product to be present in the cart: " + productName);

            // Step 4: Proceed to checkout
            CheckoutPage checkoutPage = cartPage.proceedToCheckout();

            // Step 5: Place the order
            PaymentPage paymentPage = checkoutPage.placeOrder();

            // Step 6: Enter payment details (you would need a data provider for this)
            // For a complete test, you would get card details from a data provider
            String nameOnCard = "John Doe";
            String cardNumber = "4000 0000 0000";
            String cvc = "123";
            String expiryMonth = "12";
            String expiryYear = "2025";
            paymentPage.enterCardDetails(nameOnCard, cardNumber, cvc, expiryMonth, expiry
Year);
            OrderConfirmationPage orderConfirmationPage = paymentPage.confirmOrder();

            // Step 7: Assert order confirmation
            String actualMessage = orderConfirmationPage.getOrderConfirmationMessage();
            Assert.assertEquals(actualMessage, "Order Placed!", "Order confirmation messa
ge is incorrect.");

            // Step 8: Logout
            HomePage finalHomePage = orderConfirmationPage.clickOnContinueButton();
            finalHomePage.logout();

        } else {
            // Test for invalid login if needed
            System.out.println("Skipping end-to-end flow for invalid login credentials.")
;
        }
    }
}
```

```java
package ecommerceautomation.tests;

import ecommerceautomation.base.BaseTest;
import ecommerceautomation.pages.CartPage;
import ecommerceautomation.pages.CheckoutPage;
import ecommerceautomation.pages.HomePage;
import ecommerceautomation.pages.OrderConfirmationPage;
import ecommerceautomation.pages.PaymentPage;
import ecommerceautomation.pages.ProductPage;
import ecommerceautomation.utils.ExcelUtils;
import org.testng.Assert;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

import java.util.List;

public class PaymentTest extends BaseTest {

    @DataProvider(name = "paymentData")
    public Object[][] getPaymentData() throws Exception {
        ExcelUtils excel = new ExcelUtils(config.getProperty("card.details.path"), "CardD
etails");
        List<String[]> allRows = excel.getAllRows();

        Object[][] cardData = new Object[allRows.size()][5];
        for (int i = 0; i < allRows.size(); i++) {
            String[] row = allRows.get(i);
            cardData[i][0] = row[0]; // nameOnCard
            cardData[i][1] = row[1]; // cardNumber
            cardData[i][2] = row[2]; // cvc
            cardData[i][3] = row[3]; // expiryMonth
            cardData[i][4] = row[4]; // expiryYear
        }
        return cardData;
    }

    @Test(dataProvider = "paymentData")
    public void completePaymentTest(String nameOnCard, String cardNumber, String cvc, Str
ing expiryMonth, String expiryYear) {
        // Step 1: Start from the home page and add a product to the cart
        HomePage homePage = new HomePage(getDriver(), wait);
        ProductPage productPage = homePage.navigateToProductsPage();
        CartPage cartPage = productPage.searchProduct("Blue Top");

        // Step 2: Proceed to checkout and place the order
        CheckoutPage checkoutPage = cartPage.proceedToCheckout();
        PaymentPage paymentPage = checkoutPage.placeOrder();

        // Step 3: Enter payment details and confirm the order
        paymentPage.enterCardDetails(nameOnCard, cardNumber, cvc, expiryMonth, expiryYear
);
        OrderConfirmationPage orderConfirmationPage = paymentPage.confirmOrder();

        // Step 4: Assert that the order confirmation message is displayed
        String confirmationMessage = orderConfirmationPage.getOrderConfirmationMessage();
        Assert.assertEquals(confirmationMessage, "Order Placed!", "The order confirmation
 message is incorrect.");
    }
}
```

```
package ecommerceautomation.tests;

import ecommerceautomation.base.BaseTest;
import ecommerceautomation.pages.AccountInformationPage;
import ecommerceautomation.pages.HomePage;
import ecommerceautomation.pages.SignupPage;
import ecommerceautomation.utils.ExcelUtils;
import org.testng.Assert;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class AccountInformationPageTest extends BaseTest {

    @DataProvider(name = "registerUserData")
    public Object[][] getRegisterUserData() throws Exception {
        ExcelUtils excel = new ExcelUtils(config.getProperty("user.data.path"), "Register
Users");
        return excel.getAllRowsAsObjectArray();
    }

    @Test(dataProvider = "registerUserData")
    public void registerNewUserTest(String name, String email, String password, String ti
tle,
                                    String day, String month, String year, String firstna
me,
                                    String lastname, String company, String address, Stri
ng country,
                                    String state, String city, String zip, String mobileN
o) {

        // Step 1: Start the signup process on the SignupPage
        SignupPage signupPage = new SignupPage(getDriver(), wait);
        signupPage.clickOnSignupLoginLink();
        signupPage.enterNameAtSignUp(name);
        signupPage.enterEmailAtSignUp(email);

        // Step 2: Continue to the account information page
        AccountInformationPage accountInfoPage = signupPage.clickOnSignupButton();

        // Step 3: Fill out account and address information on the AccountInformationPage
        accountInfoPage.enterAccountInformation(title, password, day, month, year);
        accountInfoPage.enterAddressInformation(firstname, lastname, company, address, co
untry, state, city, zip, mobileNo);

        // Step 4: Click create account and navigate to the home page
        HomePage homePage = accountInfoPage.clickOnCreateAccountButton();

        // Step 5: Assert that account creation was successful
        Assert.assertTrue(homePage.isUserLoggedIn(), "Account creation failed or user not
 logged in.");
    }
}
```

```java
package ecommerceautomation.tests;

import ecommerceautomation.base.BaseTest;
import ecommerceautomation.pages.HomePage;
import ecommerceautomation.pages.ProductPage;
import ecommerceautomation.pages.CartPage;
import ecommerceautomation.pages.CheckoutPage;
import ecommerceautomation.pages.PaymentPage;
import ecommerceautomation.pages.OrderConfirmationPage;
import ecommerceautomation.utils.ExcelUtils;
import org.testng.Assert;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

import java.util.Arrays;
import java.util.List;

public class CheckoutTest extends BaseTest {

    @DataProvider(name = "checkoutData")
    public Object[][] getCheckoutData() throws Exception {
        // Updated to get the file path from a config file for better maintainability
        ExcelUtils excel = new ExcelUtils(config.getProperty("user.data.path"), "Register
Users");
        List<String[]> allRows = excel.getAllRows();

        Object[][] data = new Object[allRows.size()][];
        for (int i = 0; i < allRows.size(); i++) {
            data[i] = allRows.get(i); // each row contains: name, address, city, zipcode,
 country
        }
        return data;
    }

    @Test(dataProvider = "checkoutData")
    public void placeOrderTest(String name, String address, String city, String zipcode,
String country) {
        // Step 1: Start from the home page and add a product to the cart
        HomePage homePage = new HomePage(getDriver(), wait);
        ProductPage productPage = homePage.navigateToProductsPage();
        CartPage cartPage = productPage.searchProduct("Blue Top");

        // Step 2: Proceed to checkout from the Cart Page
        CheckoutPage checkoutPage = cartPage.proceedToCheckout();

        // Step 3: Compare delivery and billing addresses
        List<String> deliveryDetails = checkoutPage.getDeliveryAddressDetails();
        List<String> billingDetails = checkoutPage.getBillingAddressDetails();

        Assert.assertEquals(deliveryDetails, billingDetails, "Delivery and billing addres
ses should match.");

        // Step 4: Place the order and navigate to the payment page
        PaymentPage paymentPage = checkoutPage.placeOrder();

        // Optional: You could add a test to fill out payment details here
        // paymentPage.enterCardDetails(...)
        // OrderConfirmationPage confirmationPage = paymentPage.confirmOrder();

        // Step 5: Assert that we are on the payment page
        String currentUrl = getDriver().getCurrentUrl();
        Assert.assertTrue(currentUrl.contains("payment"), "Expected to be on the payment
page after placing the order.");
    }
}
```

```
package ecommerceautomation.tests;

import ecommerceautomation.base.BaseTest;
import ecommerceautomation.pages.*;
import ecommerceautomation.utils.ExcelUtils;
import org.testng.Assert;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

import java.util.List;

public class OrderConfirmationTest extends BaseTest {

    @DataProvider(name = "orderData")
    public Object[][] getOrderData() throws Exception {
        ExcelUtils excel = new ExcelUtils(config.getProperty("card.details.path"), "CardD
etails");
        List<String[]> allRows = excel.getAllRows();

        // Expected columns: nameOnCard, cardNumber, cvc, expiryMonth, expiryYear
        Object[][] data = new Object[allRows.size()][5];
        for (int i = 0; i < allRows.size(); i++) {
            String[] row = allRows.get(i);
            data[i][0] = row[0]; // nameOnCard
            data[i][1] = row[1]; // cardNumber
            data[i][2] = row[2]; // cvc
            data[i][3] = row[3]; // expiryMonth
            data[i][4] = row[4]; // expiryYear
        }
        return data;
    }

    @Test(dataProvider = "orderData")
    public void verifyOrderConfirmation(String nameOnCard, String cardNumber, String cvc,
 String expiryMonth, String expiryYear) {
        // Step 1: Start from the home page and add a product to the cart
        HomePage homePage = new HomePage(getDriver(), wait);
        ProductPage productPage = homePage.navigateToProductsPage();
        CartPage cartPage = productPage.searchProduct("Blue Top");

        // Step 2: Proceed to checkout and place the order
        CheckoutPage checkoutPage = cartPage.proceedToCheckout();
        PaymentPage paymentPage = checkoutPage.placeOrder();

        // Step 3: Enter payment details and confirm the order
        paymentPage.enterCardDetails(nameOnCard, cardNumber, cvc, expiryMonth, expiryYear
);
        OrderConfirmationPage orderConfirmationPage = paymentPage.confirmOrder();

        // Step 4: Verify the order confirmation message
        String actualMessage = orderConfirmationPage.getOrderConfirmationMessage();
        Assert.assertEquals(actualMessage, "Order Placed!", "Order confirmation message i
s incorrect.");

        // Step 5: Download the invoice and continue
        orderConfirmationPage.downloadInvoice();
        orderConfirmationPage.clickOnContinueButton();

        // Step 6: Verify that we are redirected back to the home page
        Assert.assertTrue(getDriver().getCurrentUrl().contains(config.getProperty("baseUR
L")), "Not redirected to the home page after clicking 'Continue'.");
    }
}
```

```
package ecommerceautomation.utils;

import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class ExcelUtils {

    private Sheet sheet;

    public ExcelUtils(String filePath, String sheetName) throws Exception {
        if (filePath == null || filePath.isEmpty()) {
            throw new IllegalArgumentException("Excel file path cannot be null or empty."
);
        }

        try (FileInputStream file = new FileInputStream(filePath);
             Workbook workbook = new XSSFWorkbook(file)) {
            sheet = workbook.getSheet(sheetName);
            if (sheet == null) {
                throw new RuntimeException("Sheet '" + sheetName + "' not found in Excel
file: " + filePath);
            }
        } catch (Exception e) {
            throw new RuntimeException("Failed to open or read Excel file: " + filePath,
e);
        }
    }

    // Returns all rows as list of string arrays, skipping blank rows
    public List<String[]> getAllRows() {
        List<String[]> data = new ArrayList<>();
        Iterator<Row> rowIterator = sheet.iterator();

        boolean header = true;
        while (rowIterator.hasNext()) {
            Row row = rowIterator.next();
            if (header) {
                header = false;
                continue; // skip header
            }

            int cellCount = row.getLastCellNum();
            String[] rowData = new String[cellCount];

            boolean isEmptyRow = true;
            for (int i = 0; i < cellCount; i++) {
                Cell cell = row.getCell(i, Row.MissingCellPolicy.CREATE_NULL_AS_BLANK);
                String value = cell.toString().trim();
                rowData[i] = value;
                if (!value.isEmpty()) {
                    isEmptyRow = false; // found actual data
                }
            }

            if (!isEmptyRow) { // â\234\205 only add non-empty rows
                data.add(rowData);
            }
        }
        return data;
    }

    public Object[][] getAllRowsAsObjectArray() {
        List<String[]> allRows = getAllRows();
        Object[][] data = new Object[allRows.size()][];
        for (int i = 0; i < allRows.size(); i++) {
            data[i] = allRows.get(i);
```

```
        }
        return data;
    }

    // â\234\205 Returns first non-empty row (ignores blanks)
    public String[] getFirstRow() {
        List<String[]> rows = getAllRows();
        if (rows.isEmpty()) {
            throw new RuntimeException("No valid data rows found in sheet.");
        }
        return rows.get(0);
    }
}
```

```java
package ecommerceautomation.utils;

import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

public class ActionsUtils {

    private WebDriver driver;

    public ActionsUtils(WebDriver driver) {
        this.driver = driver;
    }

    public void scrollToElement(WebElement element) {
        ((JavascriptExecutor) driver).executeScript("arguments[0].scrollIntoView(true);",
 element);
    }

    public void clickWithJS(WebElement element) {
        ((JavascriptExecutor) driver).executeScript("arguments[0].click();", element);
    }
}
```

```java
package ecommerceautomation.utils;

import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.time.Duration;
import java.util.List;

public class WaitUtils {

    private WebDriver driver;
    private WebDriverWait wait;
    private static final Duration DEFAULT_TIMEOUT = Duration.ofSeconds(15); // A reasonab
le default

    public WaitUtils(WebDriver driver) {
        this.driver = driver;
        this.wait = new WebDriverWait(driver, DEFAULT_TIMEOUT);
    }

    // You can keep an overloaded constructor if you need a custom wait for a specific sc
enario
    public WaitUtils(WebDriver driver, int timeoutInSeconds) {
        this.driver = driver;
        this.wait = new WebDriverWait(driver, Duration.ofSeconds(timeoutInSeconds));
    }

    // --------------------------------
    // Page load
    // --------------------------------
    public void waitForPageToLoad() {
        wait.until(webDriver -> ((JavascriptExecutor) webDriver)
                .executeScript("return document.readyState").equals("complete"));
    }

    // --------------------------------
    // Explicit waits (By overloads)
    // --------------------------------
    public WebElement waitForElementToBeVisible(By locator) {
        return wait.until(ExpectedConditions.visibilityOfElementLocated(locator));
    }

    public WebElement waitForElementToBeClickable(By locator) {
        return wait.until(ExpectedConditions.elementToBeClickable(locator));
    }

    public List<WebElement> waitForAllElementsToBeVisible(By locator) {
        return wait.until(ExpectedConditions.visibilityOfAllElementsLocatedBy(locator));
    }

    // --------------------------------
    // Explicit waits (Utility methods)
    // --------------------------------
    /**
     * Checks if an element is visible within a given timeout.
     * This is useful for handling optional or conditional elements.
     * @param locator The locator of the element.
     * @param timeoutInSeconds The timeout in seconds.
     * @return true if the element is visible, false otherwise.
     */
    public boolean isElementVisible(By locator, int timeoutInSeconds) {
        try {
            new WebDriverWait(driver, Duration.ofSeconds(timeoutInSeconds))
                    .until(ExpectedConditions.visibilityOfElementLocated(locator));
            return true;
        } catch (Exception e) {
            return false;
```

```
            }
        }
    }
```

```java
package ecommerceautomation.utils;

import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;

public class JsonUtils {

    private JSONObject jsonObject;

    public JsonUtils(String filePath) throws Exception {
        JSONParser parser = new JSONParser();
        FileReader reader = new FileReader(filePath);
        jsonObject = (JSONObject) parser.parse(reader);
    }

    public String getString(String key) {
        return (String) jsonObject.get(key);
    }

    public long getLong(String key) {
        return (Long) jsonObject.get(key);
    }

    public double getDouble(String key) {
        return (Double) jsonObject.get(key);
    }

    // New method: Get list of strings from JSON array
    public List<String> getStringList(String key) {
        List<String> list = new ArrayList<>();
        JSONArray array = (JSONArray) jsonObject.get(key);
        if (array != null) {
            for (Object obj : array) {
                list.add((String) obj);
            }
        }
        return list;
    }
}
```

```java
package ecommerceautomation.pages;

import ecommerceautomation.utils.WaitUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class HomePage {

    private WebDriver driver;
    private WaitUtils wait;
    private static Properties locators;

    static {
        locators = new Properties();
        try (FileInputStream fis = new FileInputStream("src/main/resources/locators.prope
rties")) {
            locators.load(fis);
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException("Failed to load locators.properties file.");
        }
    }

    public HomePage(WebDriver driver, WaitUtils wait) {
        this.driver = driver;
        this.wait = wait;
    }

    // ------------------- Locators -------------------
    private By loggedInUserLink = By.xpath(locators.getProperty("home.loggedInUserLink"))
;
    private By logoutButton = By.xpath(locators.getProperty("home.logoutButton"));
    private By productsLink = By.xpath(locators.getProperty("home.productsLink"));

    // ------------------- Methods -------------------
    public boolean isUserLoggedIn() {
        return wait.isElementVisible(loggedInUserLink, 5);
    }

    public LoginPage logout() {
        if (isUserLoggedIn()) {
            wait.waitForElementToBeClickable(logoutButton).click();
            return new LoginPage(driver, wait);
        }
        return null; // Or throw an exception
    }

    public ProductPage navigateToProductsPage() {
        wait.waitForElementToBeClickable(productsLink).click();
        return new ProductPage(driver, wait);
    }
}
```

```java
package ecommerceautomation.pages;

import ecommerceautomation.utils.WaitUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class PaymentPage {
	private WebDriver driver;
	private WaitUtils wait;
	private static Properties locators;

	// Static block to load properties file only once
	static {
		locators = new Properties();
		try (FileInputStream fis = new FileInputStream("src/main/resources/locato
rs.properties")) {
			locators.load(fis);
		} catch (IOException e) {
			e.printStackTrace();
			throw new RuntimeException("Failed to load locators.properties fi
le.");
		}
	}

	public PaymentPage(WebDriver driver, WaitUtils wait) {
		this.driver = driver;
		this.wait = wait;
	}

	// ------------------- Locators -------------------
	private By nameOnCard = By.xpath(locators.getProperty("payment.nameOnCard"));
	private By cardNumber = By.xpath(locators.getProperty("payment.cardNumber"));
	private By cardCVNumber = By.xpath(locators.getProperty("payment.cardCVNumber"));
	private By cardExpiryMonth = By.xpath(locators.getProperty("payment.cardExpiryMon
th"));
	private By cardExpiryYear = By.xpath(locators.getProperty("payment.cardExpiryYear
"));
	private By PayAndConfirmOrderButton = By.xpath(locators.getProperty("payment.PayA
ndConfirmOrderButton"));

	// ------------------- Actions -------------------
	public void enterCardDetails(String name, String cardNumber, String CVCNumber,
			String expiryMonth, String expiryYear) {
		wait.waitForElementToBeVisible(nameOnCard).sendKeys(name);
		wait.waitForElementToBeVisible(this.cardNumber).sendKeys(cardNumber);
		wait.waitForElementToBeVisible(cardCVNumber).sendKeys(CVCNumber);
		wait.waitForElementToBeVisible(cardExpiryMonth).sendKeys(expiryMonth);
		wait.waitForElementToBeVisible(cardExpiryYear).sendKeys(expiryYear);
	}

	public OrderConfirmationPage confirmOrder() {
		wait.waitForElementToBeClickable(PayAndConfirmOrderButton).click();
		return new OrderConfirmationPage(driver, wait);
	}
}
```

```
package ecommerceautomation.pages;

import ecommerceautomation.utils.WaitUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.ui.Select;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.List;
import java.util.Properties;

public class AccountInformationPage {

    private WebDriver driver;
    private WaitUtils wait;
    private static Properties locators;

    static {
        locators = new Properties();
        try (FileInputStream fis = new FileInputStream("src/main/resources/locators.prope
rties")) {
            locators.load(fis);
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException("Failed to load locators.properties file.");
        }
    }

    public AccountInformationPage(WebDriver driver, WaitUtils wait) {
        this.driver = driver;
        this.wait = wait;
    }

    // ------------------- Locators -------------------
    private By passwordInput = By.xpath(locators.getProperty("account.passwordInputAcntIn
fo"));
    private By dayOfBirth = By.xpath(locators.getProperty("account.dayOfBirth"));
    private By monthOfBirth = By.xpath(locators.getProperty("account.monthOfBirth"));
    private By yearOfBirth = By.xpath(locators.getProperty("account.yearOfBirth"));
    private By titleRadioButtons = By.xpath("//input[@name='title']");
    private By firstName = By.xpath(locators.getProperty("account.firstName"));
    private By lastName = By.xpath(locators.getProperty("account.lastName"));
    private By companyName = By.xpath(locators.getProperty("account.companyName"));
    private By firstAddress = By.xpath(locators.getProperty("account.firstAddress"));
    private By countryName = By.xpath(locators.getProperty("account.countryName"));
    private By stateName = By.xpath(locators.getProperty("account.stateName"));
    private By cityName = By.xpath(locators.getProperty("account.cityName"));
    private By zipcode = By.xpath(locators.getProperty("account.zipcode"));
    private By mobileNumber = By.xpath(locators.getProperty("account.mobileNumber"));
    private By createAccountButton = By.xpath(locators.getProperty("account.createAccount
Button"));

    // ------------------- Public Methods -------------------
    public void enterAccountInformation(String title, String password, String day, String
 month, String year) {
        selectTitle(title);
        wait.waitForElementToBeVisible(passwordInput).sendKeys(password);
        new Select(wait.waitForElementToBeVisible(dayOfBirth)).selectByValue(day);
        new Select(wait.waitForElementToBeVisible(monthOfBirth)).selectByVisibleText(mont
h);
        new Select(wait.waitForElementToBeVisible(yearOfBirth)).selectByValue(year);
    }

    public void enterAddressInformation(String firstname, String lastname, String company
, String address, String country,
                                        String state, String city, String zip, String mob
ileNo) {
        wait.waitForElementToBeVisible(firstName).sendKeys(firstname);
        wait.waitForElementToBeVisible(lastName).sendKeys(lastname);
```

```
        wait.waitForElementToBeVisible(companyName).sendKeys(company);
        wait.waitForElementToBeVisible(firstAddress).sendKeys(address);
        new Select(wait.waitForElementToBeVisible(countryName)).selectByValue(country);
        wait.waitForElementToBeVisible(stateName).sendKeys(state);
        wait.waitForElementToBeVisible(cityName).sendKeys(city);
        wait.waitForElementToBeVisible(zipcode).sendKeys(zip);
        wait.waitForElementToBeVisible(mobileNumber).sendKeys(mobileNo);
    }

    public HomePage clickOnCreateAccountButton() {
        wait.waitForElementToBeClickable(createAccountButton).click();
        return new HomePage(driver, wait);
    }

    private void selectTitle(String title) {
        List<WebElement> titles = wait.waitForAllElementsToBeVisible(titleRadioButtons);
        for (WebElement titleElement : titles) {
            if (titleElement.getAttribute("value").equalsIgnoreCase(title)) {
                titleElement.click();
                break;
            }
        }
    }
}
```

```java
package ecommerceautomation.pages;

import ecommerceautomation.utils.WaitUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.List;
import java.util.Properties;
import java.util.Arrays;

public class CheckoutPage {

    private WebDriver driver;
    private WaitUtils wait;
    private static Properties locators;

    // Static block to load properties file only once
    static {
        locators = new Properties();
        try (FileInputStream fis = new FileInputStream("src/main/resources/locators.prope
rties")) {
            locators.load(fis);
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException("Failed to load locators.properties file.");
        }
    }

    public CheckoutPage(WebDriver driver, WaitUtils wait) {
        this.driver = driver;
        this.wait = wait;
    }

    // ------------------- Locators -------------------
    private By deliveryName = By.xpath(locators.getProperty("order.deliveryName"));
    private By deliveryCompany = By.xpath(locators.getProperty("order.deliveryCompany"));
    private By deliveryAddress = By.xpath(locators.getProperty("order.deliveryAddress"));
    private By deliveryCityStateZip = By.xpath(locators.getProperty("order.deliveryCitySt
ateZip"));
    private By deliveryCountry = By.xpath(locators.getProperty("order.deliveryCountry"));
    private By deliveryPhone = By.xpath(locators.getProperty("order.deliveryPhone"));
    private By billingName = By.xpath(locators.getProperty("order.billingName"));
    private By billingCompany = By.xpath(locators.getProperty("order.billingCompany"));
    private By billingAddress = By.xpath(locators.getProperty("order.billingAddress"));
    private By billingCityStateZip = By.xpath(locators.getProperty("order.billingCityStat
eZip"));
    private By billingCountry = By.xpath(locators.getProperty("order.billingCountry"));
    private By billingPhone = By.xpath(locators.getProperty("order.billingPhone"));
    private By placeOrderButton = By.xpath(locators.getProperty("order.placeOrderButton")
);

    // ------------------- Public Methods -------------------
    public List<String> getDeliveryAddressDetails() {
        String dName = wait.waitForElementToBeVisible(deliveryName).getText();
        String dCompany = wait.waitForElementToBeVisible(deliveryCompany).getText();
        String dAddress = wait.waitForElementToBeVisible(deliveryAddress).getText();
        String dCityStateZip = wait.waitForElementToBeVisible(deliveryCityStateZip).getTe
xt();
        String dCountry =  wait.waitForElementToBeVisible(deliveryCountry).getText();
        String dPhone = wait.waitForElementToBeVisible(deliveryPhone).getText();

        return Arrays.asList(dName, dCompany, dAddress, dCityStateZip, dCountry, dPhone);
    }

    public List<String> getBillingAddressDetails() {
        String bName = wait.waitForElementToBeVisible(billingName).getText();
        String bCompany = wait.waitForElementToBeVisible(billingCompany).getText();
        String bAddress = wait.waitForElementToBeVisible(billingAddress).getText();
```

```
        String bCityStateZip = wait.waitForElementToBeVisible(billingCityStateZip).getTex
t();
        String bCountry = wait.waitForElementToBeVisible(billingCountry).getText();
        String bPhone = wait.waitForElementToBeVisible(billingPhone).getText();

        return Arrays.asList(bName, bCompany, bAddress, bCityStateZip, bCountry, bPhone);
    }

    public PaymentPage placeOrder() {
        wait.waitForElementToBeClickable(placeOrderButton).click();
        return new PaymentPage(driver, wait);
    }
}
```

```java
package ecommerceautomation.pages;

import ecommerceautomation.utils.WaitUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class OrderConfirmationPage {

	private WebDriver driver;
	private WaitUtils wait;
	private static Properties locators;

	// Static block to load properties file only once
	static {
		locators = new Properties();
		try (FileInputStream fis = new FileInputStream("src/main/resources/locato
rs.properties")) {
			locators.load(fis);
		} catch (IOException e) {
			e.printStackTrace();
			throw new RuntimeException("Failed to load locators.properties fi
le.");
		}
	}

	public OrderConfirmationPage(WebDriver driver, WaitUtils wait) {
		this.driver = driver;
		this.wait = wait;
	}

	// ------------------- Locators -------------------
	private By orderConfirmationMessage = By.xpath(locators.getProperty("confirm.conf
irmMessage"));
	private By downloadInvoiceButton = By.xpath(locators.getProperty("order.downloadI
nvoiceButton"));
	private By continueButton = By.xpath(locators.getProperty("confirm.continueButton
"));

	// ------------------- Actions & Getters -------------------
	public String getOrderConfirmationMessage() {
		return wait.waitForElementToBeVisible(orderConfirmationMessage).getText()
;
	}

	public void downloadInvoice() {
		wait.waitForElementToBeClickable(downloadInvoiceButton).click();
	}

	public HomePage clickOnContinueButton() {
		wait.waitForElementToBeClickable(continueButton).click();
		// Assuming clicking continue redirects to the home page
		return new HomePage(driver, wait);
	}
}
```

```java
package ecommerceautomation.pages;

import ecommerceautomation.utils.WaitUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.Properties;
import java.util.stream.Collectors;

public class CartPage {

    private WebDriver driver;
    private WaitUtils wait;
    private static Properties locators;

    // Static block to load properties file only once
    static {
        locators = new Properties();
        try (FileInputStream fis = new FileInputStream("src/main/resources/locators.prope
rties")) {
            locators.load(fis);
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException("Failed to load locators.properties file.");
        }
    }

    public CartPage(WebDriver driver, WaitUtils wait) {
        this.driver = driver;
        this.wait = wait;
    }

    // ------------------- Locator Methods -------------------
    // Use By objects for reusability and explicit waits
    private By productQuantity = By.xpath(locators.getProperty("cart.productQuantity"));
    private By allDeleteButtons = By.xpath(locators.getProperty("cart.allDeleteButtons"))
;
    private By allQuantities = By.xpath(locators.getProperty("cart.allQuantities"));
    private By allProductNames = By.xpath(locators.getProperty("cart.allProductNames"));
    private By checkoutButton = By.xpath(locators.getProperty("cart.checkoutButton"));
    private By checkoutModal = By.xpath(locators.getProperty("cart.checkoutModal"));
    private By registerLoginButtonAtModal = By.xpath(locators.getProperty("cart.RegisterL
oginButtonAtModal"));

    // ------------------- Public Methods -------------------
    public boolean isProductInCart(String productName) {
        List<WebElement> products = wait.waitForAllElementsToBeVisible(allProductNames);
        return products.stream()
                .anyMatch(product -> product.getText().equalsIgnoreCase(productName));
    }

    public int getProductQuantity() {
        String quantity = wait.waitForElementToBeVisible(productQuantity).getText();
        return Integer.parseInt(quantity);
    }

    public void removeProduct(String productName) {
        // Wait for all product names to be visible.
        List<WebElement> names = wait.waitForAllElementsToBeVisible(allProductNames);

        // Find the index of the product you want to remove
        int productIndex = -1;
        for (int i = 0; i < names.size(); i++) {
            if (names.get(i).getText().equalsIgnoreCase(productName)) {
                productIndex = i;
                break;
```

```
            }
        }

        if (productIndex != -1) {
            // Construct the locator for the specific delete button using the index
            // This is a much more robust approach than trying to get the WebElement dire
ctly
            String deleteButtonXpath = locators.getProperty("cart.allDeleteButtons") + "[
" + (productIndex + 1) + "]";
            By deleteButtonLocator = By.xpath(deleteButtonXpath);

            // Pass the By locator to the WaitUtils method
            wait.waitForElementToBeClickable(deleteButtonLocator).click();
        } else {
            throw new NoSuchElementException("Product '" + productName + "' not found in
the cart.");
        }
    }

    public CheckoutPage proceedToCheckout() {
        wait.waitForElementToBeClickable(checkoutButton).click();

        // Handle modal if it appears (user not logged in)
        if (wait.isElementVisible(checkoutModal, 3)) {
            wait.waitForElementToBeClickable(registerLoginButtonAtModal).click();
        }
        // Note: The isElementVisible method in WaitUtils should return a boolean
        // to handle the condition gracefully without exceptions.
        return new CheckoutPage(driver, wait);
    }

    public List<Integer> getAllProductQuantities() {
        return wait.waitForAllElementsToBeVisible(allQuantities).stream()
                .map(qty -> Integer.parseInt(qty.getAttribute("value")))
                .collect(Collectors.toList());
    }
}
```

```
package ecommerceautomation.pages;

import ecommerceautomation.utils.WaitUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.List;
import java.util.Properties;

public class ProductPage {

    private WebDriver driver;
    private WaitUtils wait;
    private static Properties locators;

    static {
        locators = new Properties();
        try (FileInputStream fis = new FileInputStream("src/main/resources/locators.prope
rties")) {
            locators.load(fis);
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException("Failed to load locators.properties file.");
        }
    }

    public ProductPage(WebDriver driver, WaitUtils wait) {
        this.driver = driver;
        this.wait = wait;
    }

    // ------------------- By Locators -------------------
    private By productHeader = By.xpath(locators.getProperty("product.productsLink"));
    private By searchBox = By.xpath(locators.getProperty("product.searchBox"));
    private By searchButton = By.xpath(locators.getProperty("product.searchButton"));
    private By productListLocator = By.xpath("//div[@class='productinfo text-center']");
    private By modalContent = By.xpath("//div[@class='modal-content']");
    private By viewCartLink = By.xpath("//a//u[text()='View Cart']");

    // ------------------- Public Methods -------------------
    public CartPage searchProduct(String productName) {
        wait.waitForElementToBeClickable(productHeader).click();
        performSearch(productName);
        WebElement productElement = findProductInList(productName);
        if (productElement != null) {
            addProductToCart(productElement);
        } else {
            throw new RuntimeException("Product '" + productName + "' not found on the pa
ge.");
        }
        return handleAddToCartModal();
    }

    // ------------------- Private Helper Methods -------------------
    private void performSearch(String productName) {
        wait.waitForElementToBeVisible(searchBox).sendKeys(productName);
        wait.waitForElementToBeClickable(searchButton).click();
    }

    private WebElement findProductInList(String productName) {
        List<WebElement> products = wait.waitForAllElementsToBeVisible(productListLocator
);
        for (WebElement product : products) {
            WebElement searchedProductName = product.findElement(By.tagName("p"));
            if (searchedProductName.getText().equalsIgnoreCase(productName)) {
                return product;
            }
```

```
        }
        return null;
    }

    private void addProductToCart(WebElement productElement) {
        // Hover over the product container
        Actions actions = new Actions(driver);
        actions.moveToElement(productElement).perform();

        // Now, find the 'Add to cart' button.
        // The wait is now a local wait, and it's not the WaitUtils wait.
        WebElement addToCartButton = productElement.findElement(By.xpath(".//a[contains(t
ext(),'Add to cart')]"));
        addToCartButton.click();
    }

    private CartPage handleAddToCartModal() {
        wait.waitForElementToBeVisible(modalContent);
        wait.waitForElementToBeClickable(viewCartLink).click();
        return new CartPage(driver, wait);
    }
}
```

```java
package ecommerceautomation.pages;

import ecommerceautomation.utils.WaitUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class LoginPage {

    private WebDriver driver;
    private WaitUtils wait;
    private static Properties locators;

    static {
        locators = new Properties();
        try (FileInputStream fis = new FileInputStream("src/main/resources/locators.prope
rties")) {
            locators.load(fis);
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException("Failed to load locators.properties file.");
        }
    }

    public LoginPage(WebDriver driver, WaitUtils wait) {
        this.driver = driver;
        this.wait = wait;
    }

    // ------------------- By Locators -------------------
    private By signupLoginLink = By.xpath(locators.getProperty("login.signupLoginLink"));
    private By emailInput = By.xpath(locators.getProperty("login.emailInput"));
    private By passwordInput = By.xpath(locators.getProperty("login.passwordInput"));
    private By loginButton = By.xpath(locators.getProperty("login.loginButton"));
    private By loginErrorMessage = By.xpath(locators.getProperty("login.loginErrorMessage
"));
    private By logoutButton = By.xpath(locators.getProperty("login.logoutButton"));

    // ------------------- Public Methods -------------------
    public void clickOnSignupLoginLink() {
        wait.waitForElementToBeClickable(signupLoginLink).click();
    }

    public void enterEmailForLogin(String email) {
        WebElement emailElement = wait.waitForElementToBeVisible(emailInput);
        emailElement.clear();
        emailElement.sendKeys(email);
    }

    public void enterPasswordForLogin(String password) {
        WebElement passwordElement = wait.waitForElementToBeVisible(passwordInput);
        passwordElement.clear();
        passwordElement.sendKeys(password);
    }

    public void clickOnLoginButton() {
        wait.waitForElementToBeClickable(loginButton).click();
    }

    public boolean isLoginButtonVisible() {
        try {
            wait.waitForElementToBeVisible(loginButton);
            return true;
        } catch (Exception e) {
            return false;
        }
    }
```

```
    public boolean isSignupLoginLinkVisible() {
        return wait.isElementVisible(signupLoginLink, 5);
    }

    public String getLoginErrorMessage() {
        if (wait.isElementVisible(loginErrorMessage, 3)) {
            return wait.waitForElementToBeVisible(loginErrorMessage).getText();
        }
        return null;
    }


    public Object login(String email, String password) {
        clickOnSignupLoginLink();
        enterEmailForLogin(email);
        enterPasswordForLogin(password);
        clickOnLoginButton();

        // Check for a unique element on the Home Page to determine success
        // This is a more reliable way to check for successful navigation
        if (wait.isElementVisible(logoutButton, 5)) { // Check if logout button is visibl
e on the next page
            return new HomePage(driver, wait);
        } else {
            return new LoginPage(driver, wait); // Stay on the login page on failure
        }
    }

    public Object loginFromCurrentPage(String email, String password) {
        enterEmailForLogin(email);
        enterPasswordForLogin(password);
        clickOnLoginButton();

        // Check for a unique element on the Home Page to determine success
        if (wait.isElementVisible(logoutButton, 5)) {
            return new HomePage(driver, wait);
        } else {
            return new LoginPage(driver, wait);
        }
    }

    // You may also want a simplified logout method
    public void logout() {
        if (wait.isElementVisible(logoutButton, 5)) {
            wait.waitForElementToBeClickable(logoutButton).click();
        }
    }
}
```

```java
package ecommerceautomation.pages;

import ecommerceautomation.utils.WaitUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class SignupPage {

    private WebDriver driver;
    private WaitUtils wait;
    private static Properties locators;

    static {
        locators = new Properties();
        try (FileInputStream fis = new FileInputStream("src/main/resources/locators.prope
rties")) {
            locators.load(fis);
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException("Failed to load locators.properties file.");
        }
    }

    public SignupPage(WebDriver driver, WaitUtils wait) {
        this.driver = driver;
        this.wait = wait;
    }

    // ------------------- Locators -------------------
    private By signupLoginLink = By.xpath(locators.getProperty("login.signupLoginLink"));
    private By nameInputForSignup = By.xpath(locators.getProperty("signup.signupName"));
    private By emailInputForSignup = By.xpath(locators.getProperty("signup.signupEmail"))
;
    private By signupButton = By.xpath(locators.getProperty("signup.signupButton"));
    private By emailExistsMessage = By.xpath(locators.getProperty("signup.signupEmailExis
tMessage"));

    // ------------------- Public Methods -------------------
    public void clickOnSignupLoginLink() {
        wait.waitForElementToBeClickable(signupLoginLink).click();
    }

    public void enterNameAtSignUp(String name) {
        wait.waitForElementToBeVisible(nameInputForSignup).sendKeys(name);
    }

    public void enterEmailAtSignUp(String email) {
        wait.waitForElementToBeVisible(emailInputForSignup).sendKeys(email);
    }

    public boolean isEmailAlreadyRegistered() {
        return wait.isElementVisible(emailExistsMessage, 10);
    }

    public AccountInformationPage clickOnSignupButton() {
        wait.waitForElementToBeClickable(signupButton).click();
        return new AccountInformationPage(driver, wait);
    }
}
```

```java
package ecommerceautomation.listeners;

import ecommerceautomation.base.BaseTest;
import ecommerceautomation.utils.ActionsUtils;
import ecommerceautomation.utils.WaitUtils;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.testng.ITestContext;
import org.testng.ITestListener;
import org.testng.ITestResult;
import com.aventstack.extentreports.Status;
import com.aventstack.extentreports.ExtentReports;
import com.aventstack.extentreports.ExtentTest;
import com.aventstack.extentreports.reporter.ExtentSparkReporter;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;

public class ExtentListener implements ITestListener {

        private WebDriver driver;

    public void setDriver(WebDriver driver) {
        this.driver = driver;
    }

    private static ExtentReports extent;
    private static ExtentTest test;


    @Override
    public void onStart(ITestContext context) {
        ExtentSparkReporter spark = new ExtentSparkReporter("test-output/ExtentReport.htm
l");
        extent = new ExtentReports();
        extent.attachReporter(spark);
    }

    @Override
    public void onTestStart(ITestResult result) {
        test = extent.createTest(result.getMethod().getMethodName());
    }

    @Override
    public void onTestSuccess(ITestResult result) {
        test.log(Status.PASS, "Test Passed");
    }

    @Override
    public void onTestFailure(ITestResult result) {
        test.log(Status.FAIL, result.getThrowable());

        // Take screenshot
        File srcFile = ((TakesScreenshot) BaseTest.driver).getScreenshotAs(OutputType.FIL
E);
        String destPath = "test-output/screenshots/" + result.getMethod().getMethodName()
 + ".png";
        try {
            Files.createDirectories(new File("test-output/screenshots/").toPath());
            Files.copy(srcFile.toPath(), new File(destPath).toPath());
            test.addScreenCaptureFromPath(destPath);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onTestSkipped(ITestResult result) {
        test.log(Status.SKIP, "Test Skipped");
```

```
    }

    @Override
    public void onFinish(ITestContext context) {
        extent.flush();
    }
}
```

```java
package ecommerceautomation.base;

import java.time.Duration;
import java.util.Properties;

import org.openqa.selenium.WebDriver;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;

import ecommerceautomation.utils.WaitUtils;

public class BaseTest {

    // Make the config variable static so it’s shared across all tests
    protected static Properties config;
    protected WebDriver driver;
    protected WaitUtils wait;


    @BeforeSuite(alwaysRun = true)
    public void setupConfig() {
        if (config == null) {
            config = ConfigReader.getProperties();
        }
    }

    @BeforeMethod(alwaysRun = true)
    public void setUp() {
        // Now you can safely use the static config variable
        String browser = config.getProperty("browser", "chrome");
        boolean headless = Boolean.parseBoolean(config.getProperty("headless", "false"));

        DriverManager.initDriver(browser, headless);
        driver = DriverManager.getDriver();
        wait = new WaitUtils(driver);

        driver.manage().timeouts().pageLoadTimeout(Duration.ofSeconds(180));
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5));

        getDriver().get(config.getProperty("baseURL"));
    }

    @AfterMethod(alwaysRun = true)
    public void tearDown() {
        DriverManager.quitDriver();
    }

    // This method is already static, which is good
    public static WebDriver getDriver() {
        return DriverManager.getDriver();
    }

    public WaitUtils getWait() {
        return wait;
    }
}
```

```java
package ecommerceautomation.base;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.edge.EdgeDriver;
import org.openqa.selenium.edge.EdgeOptions;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxOptions;

import io.github.bonigarcia.wdm.WebDriverManager;

public class DriverManager {

    private static ThreadLocal<WebDriver> driver = new ThreadLocal<>();

    /**
     * Initialize WebDriver based on browser and headless flag
     */
    public static WebDriver initDriver(String browser, boolean headless) {
        if (browser.equalsIgnoreCase("chrome")) {
            WebDriverManager.chromedriver().setup();
            ChromeOptions options = new ChromeOptions();
            if (headless) options.addArguments("--headless=new");
            driver.set(new ChromeDriver(options));

        } else if (browser.equalsIgnoreCase("firefox")) {
            WebDriverManager.firefoxdriver().setup();
            FirefoxOptions options = new FirefoxOptions();
            if (headless) options.addArguments("-headless");
            driver.set(new FirefoxDriver(options));

        } else if (browser.equalsIgnoreCase("edge")) {
            WebDriverManager.edgedriver().setup();
            EdgeOptions options = new EdgeOptions();
            if (headless) options.addArguments("headless");
            driver.set(new EdgeDriver(options));

        } else {
            throw new IllegalArgumentException("Invalid browser: " + browser);
        }

        driver.get().manage().window().maximize();
        return driver.get();
    }

    public static WebDriver getDriver() {
        return driver.get();
    }

    public static void quitDriver() {
        if (driver.get() != null) {
            driver.get().quit();
            driver.remove();
        }
    }
}
```

```java
package ecommerceautomation.base;

import java.io.InputStream;
import java.util.Properties;

/**
 * ConfigReader:
 * - Loads values from src/main/resources/config.properties
 * - Allows overriding via system property (-Dkey=value)
 */
public final class ConfigReader {

    private static final String CONFIG_FILE = "config.properties";
    private static final Properties props = new Properties();

    static {
        try (InputStream is = ConfigReader.class.getClassLoader().getResourceAsStream(CON
FIG_FILE)) {
            if (is == null) {
                throw new RuntimeException("Configuration file '" + CONFIG_FILE + "' not
found in classpath");
            }
            props.load(is);
        } catch (Exception e) {
            throw new ExceptionInInitializerError("Failed to load " + CONFIG_FILE + ": "
+ e.getMessage());
        }
    }

    private ConfigReader() { /* prevent instantiation */ }

    public static String get(String key) {
        // System property override > config.properties
        String sys = System.getProperty(key);
        if (sys != null && !sys.isEmpty()) {
            return sys.trim();
        }
        String val = props.getProperty(key);
        return val == null ? null : val.trim();
    }

    public static String get(String key, String defaultValue) {
        String v = get(key);
        return v == null ? defaultValue : v;
    }

    public static int getInt(String key, int defaultValue) {
        String v = get(key);
        if (v == null) return defaultValue;
        try {
            return Integer.parseInt(v);
        } catch (NumberFormatException e) {
            return defaultValue;
        }
    }

    public static boolean getBoolean(String key, boolean defaultValue) {
        String v = get(key);
        if (v == null) return defaultValue;
        return Boolean.parseBoolean(v);
    }

    public static Properties getProperties() {
        return props;
    }
}
```