Name: Piyush Kasture

Module Name: Latest Industry Trends

PRN: 2509401147003

Assignment No. 6

1) Write 10 testcases by hand for online shopping cart module. (Requirements given below).
Create a prompt to generate these testcases.
Use AI model to generate these testcases.
Compare tests created by AI model and the ones created by you.
Also create selenium tests for the above testcases.

Requirement Document : Online Shopping Cart (E-commerce)
Module: Shopping Cart
Description:
Users can add items to the cart and place an order.
Functional Requirements
User should be able to add products to the cart.
User should be able to remove products from the cart.
Cart should automatically recalculate total price when quantity changes.
Discount coupons should be applied before checkout.
If the cart is empty, checkout button should be disabled.
User must be logged in to proceed to checkout.
Business Rules
Maximum 10 units allowed per product.
Only one coupon can be applied per order.

Prompt for generate test cases:

Role:
You are a QA Test Case Generator with expertise in writing clear, structured, beginner-friendly test cases.

Intent:
Generate a complete set of positive, negative, and boundary test cases for the Shopping Cart (E-commerce) module.

Context (source of truth):
Requirement Document 3: Online Shopping Cart (E-commerce) — Shopping Cart module.
Key points:
- Users can add/remove products to/from cart and place an order.

- FR-001: Add product to cart (product_id, quantity default 1). Validate product exists, stock > 0, total quantity per product ≤ 10.
- FR-002: Remove product (decrease quantity and remove full line; optional confirmation).
- FR-003: Quantity updates recalc (debounced 200–300ms). Currency formatting, tax/shipping placeholders.
- FR-004: Apply discount coupon (validate exists/active/eligible). Only one coupon per order. Coupon applied before checkout.
- FR-005: Checkout button disabled when cart empty.
- FR-006: Require login to checkout (guest redirected to login; guest cart preserved/merged on login).
- BR-001: Max 10 units per product (client & server).
- BR-002: Only one coupon per order.

Output Format:
- Table with columns: Test Case ID, Test Scenario, Test Steps (numbered), Test Data (sample), Expected Result (explicit message/state).
- Produce 12–15 distinct test cases covering: add, remove, update qty (valid/invalid/boundary), auto-recalc, coupon flows (valid/expired/second coupon), checkout behaviors (disabled when empty, guest->login merge), persistence, edge cases (out-of-stock after add, price change).
- Use explicit sample product_ids, coupon codes and messages from requirements (e.g., "Maximum 10 units allowed per product", "Only one coupon can be applied per order", "Add items to your cart to proceed").
- Keep language concise and beginner-friendly.

Output:

ID	Scenario (short)	Key Test Data / Expected messages
TC-CART-001	Add single product (happy)	P1001 qty=1 → subtotal $20; "Item added to cart"
TC-CART-002	Add same product increments	P1001 qty 1→2
TC-CART-003	Add product >10 (block)	attempt 11 → "Maximum 10 units allowed per product"
TC-CART-004	Add beyond stock	P3003 stock 3 → attempt 4 → "Only 3 units available"
TC-CART-005	Decrease quantity by 1	3→2; totals update
TC-CART-006	Remove line item (confirm)	removed; checkout disabled if empty

ID	Scenario (short)	Key Test Data / Expected messages
TC-CART-007	Quantity boundaries (1 and 10)	qty=1 OK; qty=10 OK
TC-CART-008	Invalid qty (0, -1, non-numeric)	show validation "Quantity must be between 1 and 10" / "Enter a valid number"
TC-CART-009	Auto-recalc (debounce)	set qty→3 → totals update after ~300ms
TC-CART-010	Apply valid coupon	SAVE10 → discount shown; "Coupon applied successfully."
TC-CART-011	Apply expired/invalid coupon	OLD50 → "Invalid or expired coupon"
TC-CART-012	Attempt second coupon	apply second → "Only one coupon can be applied per order"
TC-CART-013	Checkout disabled when empty	checkout disabled; tooltip "Add items to your cart to proceed"
TC-CART-014	Guest checkout → login → preserve cart (merge)	guest adds items → clicks checkout → redirected to login → after login cart preserved
TC-CART-015	Item goes out of stock after added	on checkout you see "Item no longer available; removed from cart" or item removed

Compare tests created by AI model and the ones created by me:

AI Gives Test Cases In: Detailed, structured format, Includes sample product IDs, sample data, exact validation messages, Contains step-by-step test steps, Written in an automation-ready style (clear expected results, measurable outcomes) and Uses clear naming conventions (TC-CART-001)

I write in manual style, Shorter and simpler test steps Less detailed, Uses general product names, not specific product IDs and Focuses mainly on core functionality.

Prompt for generate selenium test cases:

Role:
You are an Automation Engineer experienced in Selenium with Java and TestNG. Produce production-ready test code and project files.

Intent:
Generate a Selenium + Java (TestNG) test suite that automates TC-CART-001 through TC-CART-015 for the Shopping Cart module.

Context:

Use the test cases TC-CART-001..TC-CART-015 (as specified in Requirement Document 3). Assume the web app exposes testable attributes (recommended data-testid attributes).

Deliverables (exact):
1. pom.xml listing dependencies: selenium-java, testng, webdrivermanager (versions compatible with latest stable).
2. BaseTest.java: WebDriver init/teardown using WebDriverManager and headless Chrome option.
3. testng.xml: simple suite runner
4. testid selectors; if not available.
5. Recommended waits and handling for dynamic UI elements (debounce, toasts).

Output format:
Provide code in inline (complete Java classes and pom.xml). Keep them concise but complete so a developer can copy/paste into a Maven project and run after adjusting BASE_URL for local host

Output:

File name: BaseTest.java

```
package base;

import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;

public class BaseTest {
   protected WebDriver driver;
   protected String BASE_URL = "http://localhost:3000"; // update for your app

   @BeforeClass
   public void setUp() {
      WebDriverManager.chromedriver().setup();
      ChromeOptions options = new ChromeOptions();
      options.addArguments("--headless=new");
      options.addArguments("--no-sandbox");
      options.addArguments("--disable-dev-shm-usage");
      driver = new ChromeDriver(options);
      driver.manage().window().maximize();
```

```
    }

    @AfterClass
    public void tearDown() {
        if (driver != null) {
            driver.quit();
        }
    }
}
```

File name: CartPage.java

```
package pages;

import org.openqa.selenium.By;
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;

public class CartPage {
    private WebDriver d;
    private WebDriverWait wait;
    private final int WAIT = 10;
    private String baseUrl;

    public CartPage(WebDriver driver, String baseUrl) {
        this.d = driver;
        this.wait = new WebDriverWait(d, Duration.ofSeconds(WAIT));
        this.baseUrl = baseUrl;
    }

    // Navigate to product
    public void openProduct(String productId) {
        d.get(baseUrl + "/product/" + productId);
        wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector("[data-
testid='product-title']")));
    }

    public void clickAddToCart(int qty) {
        try {
```

```
            WebElement qtyInput = d.findElement(By.cssSelector("[data-testid='product-qty']"));
            qtyInput.clear();
            qtyInput.sendKeys(String.valueOf(qty));
        } catch (Exception e) {
            // qty input optional
        }
        d.findElement(By.cssSelector("[data-testid='add-to-cart']")).click();
        // wait for toast or cart update
        try {
            wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector("[data-
testid='toast']")));
        } catch (Exception ignored) {}
    }

    public void openCart() {
        d.get(baseUrl + "/cart");
        wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector("[data-
testid='cart-list']")));
    }

    public int getCartItemQuantity(String productId) {
        String selector = String.format("[data-testid='cart-item-%s'] [data-testid='cart-qty']",
productId);
        WebElement qEl =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(selector)));
        String val = qEl.getAttribute("value");
        if (val == null || val.isEmpty()) {
            val = qEl.getText();
        }
        return Integer.parseInt(val.trim());
    }

    public void setCartItemQuantity(String productId, String qty) {
        String selector = String.format("[data-testid='cart-item-%s'] [data-testid='cart-qty']",
productId);
        WebElement qEl =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(selector)));
        qEl.clear();
        qEl.sendKeys(qty);
        qEl.sendKeys("\t"); // blur to trigger update
        try { Thread.sleep(400); } catch (InterruptedException ignored) {}
    }

    public void removeCartItem(String productId, boolean confirm) {
```

```
        String selector = String.format("[data-testid='cart-item-%s'] [data-testid='cart-remove']",
productId);
        WebElement btn =
wait.until(ExpectedConditions.elementToBeClickable(By.cssSelector(selector)));
        btn.click();
        if (confirm) {
            WebElement yes =
wait.until(ExpectedConditions.elementToBeClickable(By.cssSelector("[data-testid='confirm-
yes']")));
            yes.click();
        }
        try { Thread.sleep(300); } catch (InterruptedException ignored) {}
    }

    public double getSubtotal() {
        WebElement el =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector("[data-
testid='subtotal']")));
        String text = el.getText().replace("$", "").replace(",", "").trim();
        if (text.isEmpty()) return 0.0;
        return Double.parseDouble(text);
    }

    public void applyCoupon(String code) {
        WebElement in =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector("[data-
testid='coupon-input']")));
        in.clear();
        in.sendKeys(code);
        d.findElement(By.cssSelector("[data-testid='coupon-apply']")).click();
        try { Thread.sleep(500); } catch (InterruptedException ignored) {}
    }

    public String getCouponMessage() {
        try {
            WebElement el =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector("[data-
testid='coupon-msg']")));
            return el.getText().trim();
        } catch (Exception e) {
            return "";
        }
    }
```

```
    public boolean isCheckoutEnabled() {
        WebElement btn =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector("[data-
testid='checkout-btn']")));
        return btn.isEnabled();
    }

    public void clickCheckout() {
        WebElement btn =
wait.until(ExpectedConditions.elementToBeClickable(By.cssSelector("[data-testid='checkout-
btn']")));
        btn.click();
    }

    public void login(String email, String password) {
        d.get(baseUrl + "/login");
        WebElement e =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector("[data-
testid='login-email']")));
        e.clear();
        e.sendKeys(email);
        WebElement p = d.findElement(By.cssSelector("[data-testid='login-password']"));
        p.clear();
        p.sendKeys(password);
        d.findElement(By.cssSelector("[data-testid='login-submit']")).click();
        try { Thread.sleep(1000); } catch (InterruptedException ignored) {}
    }

    public String getInlineProductError(String productId) {
        try {
            String sel = String.format("[data-testid='product-%s-error']", productId);
            return d.findElement(By.cssSelector(sel)).getText().trim();
        } catch (Exception e) {
            return "";
        }
    }

    public String getEmptyCartMessage() {
        try {
            return d.findElement(By.cssSelector("[data-testid='empty-cart-
msg']")).getText().trim();
        } catch (Exception e) {
            return "";
        }
```

```
    }
}
```

File name: CartTests.java

```
package tests;

import base.BaseTest;
import org.testng.Assert;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;
import pages.CartPage;

public class CartTests extends BaseTest {

    private CartPage cart;

    @BeforeMethod
    public void beforeEach() {
        cart = new CartPage(driver, BASE_URL);
    }

    @Test
    public void TC_CART_001_add_single_product() {
        cart.openProduct("P1001");
        cart.clickAddToCart(1);
        cart.openCart();
        Assert.assertEquals(cart.getCartItemQuantity("P1001"), 1);
        Assert.assertEquals(cart.getSubtotal(), 20.0, 0.1);
    }

    @Test
    public void TC_CART_002_add_same_product_increments() {
        cart.openProduct("P1001");
        cart.clickAddToCart(1);
        cart.openCart();
        Assert.assertEquals(cart.getCartItemQuantity("P1001"), 2);
    }

    @Test
    public void TC_CART_003_add_product_exceeding_maximum() {
        cart.openProduct("P2002");
```

```
        cart.clickAddToCart(11);
        String err = cart.getInlineProductError("P2002");
        Assert.assertTrue(err.toLowerCase().contains("maximum 10") ||
err.toLowerCase().contains("maximum"));
    }

    @Test
    public void TC_CART_004_add_product_beyond_stock() {
        cart.openProduct("P3003");
        cart.clickAddToCart(4);
        String err = cart.getInlineProductError("P3003");
        Assert.assertTrue(err.toLowerCase().contains("available") ||
err.toLowerCase().contains("only"));
    }

    @Test
    public void TC_CART_005_remove_single_unit_decrease_quantity() {
        cart.openProduct("P1001"); cart.clickAddToCart(1);
        cart.openCart();
        int before = cart.getCartItemQuantity("P1001");
        cart.setCartItemQuantity("P1001", String.valueOf(before - 1));
        Assert.assertEquals(cart.getCartItemQuantity("P1001"), before - 1);
    }

    @Test
    public void TC_CART_006_remove_entire_line_item() {
        cart.openCart();
        cart.removeCartItem("P1001", true);
        // Depending on UI, either not present or qty = 0 -> try both
        try {
            Assert.assertEquals(cart.getCartItemQuantity("P1001"), 0);
        } catch (Exception e) {
            Assert.assertTrue(true); // element not present acceptable
        }
    }

    @Test
    public void TC_CART_007_update_quantity_boundaries() {
        cart.openProduct("P4004"); cart.clickAddToCart(1);
        cart.openCart();
        cart.setCartItemQuantity("P4004", "1");
        Assert.assertEquals(cart.getCartItemQuantity("P4004"), 1);
        cart.setCartItemQuantity("P4004", "10");
        Assert.assertEquals(cart.getCartItemQuantity("P4004"), 10);
```

```
        Assert.assertEquals(cart.getSubtotal(), 50.0, 0.1);
    }

    @Test
    public void TC_CART_008_update_quantity_invalid_values() {
        cart.openCart();
        cart.setCartItemQuantity("P4004", "0");
        String err0 = cart.getInlineProductError("P4004");
        Assert.assertTrue(err0.toLowerCase().contains("quantity") ||
err0.toLowerCase().contains("valid"));
        cart.setCartItemQuantity("P4004", "-1");
        String errNeg = cart.getInlineProductError("P4004");
        Assert.assertTrue(errNeg.toLowerCase().contains("quantity") ||
errNeg.toLowerCase().contains("valid"));
        cart.setCartItemQuantity("P4004", "abc");
        String errAlpha = cart.getInlineProductError("P4004");
        Assert.assertTrue(errAlpha.toLowerCase().contains("valid") ||
errAlpha.toLowerCase().contains("number"));
    }

    @Test
    public void TC_CART_009_automatic_recalculation_debounce() throws
InterruptedException {
        cart.openCart();
        cart.setCartItemQuantity("P5005", "3");
        Thread.sleep(400); // wait for debounce
        Assert.assertEquals(cart.getCartItemQuantity("P5005"), 3);
        Assert.assertEquals(cart.getSubtotal(), 30.0, 0.5);
    }

    @Test
    public void TC_CART_010_apply_valid_coupon() {
        cart.openProduct("P2002"); cart.clickAddToCart(10); // subtotal 100
        cart.openCart();
        cart.applyCoupon("SAVE10");
        String msg = cart.getCouponMessage();
        Assert.assertTrue(msg.toLowerCase().contains("applied") ||
msg.toLowerCase().contains("success"));
        Assert.assertTrue(cart.getSubtotal() <= 100.0);
    }

    @Test
    public void TC_CART_011_apply_invalid_expired_coupon() {
        cart.openCart();
```

```
        cart.applyCoupon("OLD50");
        String msg = cart.getCouponMessage();
        Assert.assertTrue(msg.toLowerCase().contains("invalid") ||
msg.toLowerCase().contains("expired"));
    }

    @Test
    public void TC_CART_012_attempt_second_coupon_blocked() {
        cart.openCart();
        cart.applyCoupon("SAVE10");
        cart.applyCoupon("WELCOME5");
        String msg = cart.getCouponMessage();
        Assert.assertTrue(msg.toLowerCase().contains("only one coupon") ||
msg.toLowerCase().contains("one coupon"));
    }

    @Test
    public void TC_CART_013_checkout_disabled_when_cart_empty() {
        cart.openCart();
        // Attempt to clear cart (depends on UI)
        // Try clear button if exists
        // For stability, just assert checkout disabled or empty message present
        boolean enabled = cart.isCheckoutEnabled();
        if (enabled) {
            // If enabled, try to remove items to empty
            // This is environment dependent; assume test environment cleaned
            Assert.fail("Checkout unexpectedly enabled for empty cart");
        } else {
            String msg = cart.getEmptyCartMessage();
            Assert.assertTrue(msg.toLowerCase().contains("add items") || !msg.isEmpty());
        }
    }

    @Test
    public void TC_CART_014_guest_checkout_redirects_to_login_and_preserves_cart() {
        cart.openProduct("P1001"); cart.clickAddToCart(2);
        cart.openCart();
        cart.clickCheckout();
        Assert.assertTrue(driver.getCurrentUrl().contains("/login"));
        cart.login("userA@example.com", "Password123!");
        cart.openCart();
        Assert.assertTrue(cart.getCartItemQuantity("P1001") >= 2);
    }
```

```
    @Test
    public void TC_CART_015_item_goes_out_of_stock_after_add() {
        cart.openProduct("P6006"); cart.clickAddToCart(1);
        cart.openCart();
        cart.clickCheckout(); // simulate check that backend could reject
        String emptyOrAlert = cart.getInlineProductError("P6006");
        boolean removed = false;
        try { removed = cart.getCartItemQuantity("P6006") == 0; } catch (Exception ignored) {}
        Assert.assertTrue(emptyOrAlert.toLowerCase().contains("no longer") || removed);
    }
}
```

File name: testing.xml

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >
<suite name="CartSuite">
  <test name="CartTests">
    <classes>
      <class name="tests.CartTests"/>
    </classes>
  </test>
</suite>
```