Airbase Assignment

Task 1: Write a test document to list the use cases you would test via automation to verify search, filter and add-to-cart functionalities on amazon.in.

Solution:

Use case 1. Search Functionality:

- *Verify that searching for a valid product (e.g. "smart phone") displays relevant results.*
- *Verify the page title after searching.*
- *Test searching for a product using partial keywords.*
- *Check that searching for a non-existent product show "no results found" or related product.*
- *Test searching with special characters or numbers to ensure proper handling.*
- *Verify search suggestions appear and are accurate as the user types.*
- *Check that recently viewed search terms are remembered.*
- *Test searching in different categories to ensure results are displayed correctly.*

Use case 2. Filter Functionality:

- ***Brand Filter:***
    - *Apply a brand filter (e.g. "Samsung") and confirm that only products from that brand are displayed.*
    - *Test by applying multiple brand filters.*
    - *Verify that removing a brand filter correctly updates the results.*
- ***Price Filter:***
    - *Add a price filter with a minimum and maximum value (e.g. Min: 500, Max: 50000).*
    - *Check that all displayed products fall within the specified price range.*
    - *Test with invalid price ranges (e.g. min greater than max).*
- ***Sort Option:***
    - *Sort results by price from high to low using the sort option available in the upper right corner.*
    - *Verify the order of products after sorting.*

    - *Test other sorting options (e.g**. low to high, Customer review, newest arrival, best seller**).*

- ***Availability Filter:** Test filtering by "In Stock" and "Out of Stock" to ensure accurate product display.*
- ***Customer Reviews Filter:** Apply a filter for a specific star rating (e.g., 4 Stars & Up) and verify results meet the criteria.*
- ***Delivery Speed/Prime Filter:** Check if filtering by Prime eligibility or faster delivery options works correctly.*
- ***Discount/Offer Filter:** test filtering by products with ongoing discounts or offers.*
- ***Category/Department Filter:** Verify that applying a category filter (e.g. electronic, fashion, shoes etc) narrows down the results appropriately.*
- ***Combined Filters:** Test applying a combination of multiple filters (e.g. Brand + Price + Customer Reviews) and ensure all conditions are met by the displayed products.*
- ***Filter Reset:** Verify that clearing all filters or individual filters correctly reverts the search results.*

Use case 3. Add-to-Cart Functionality:

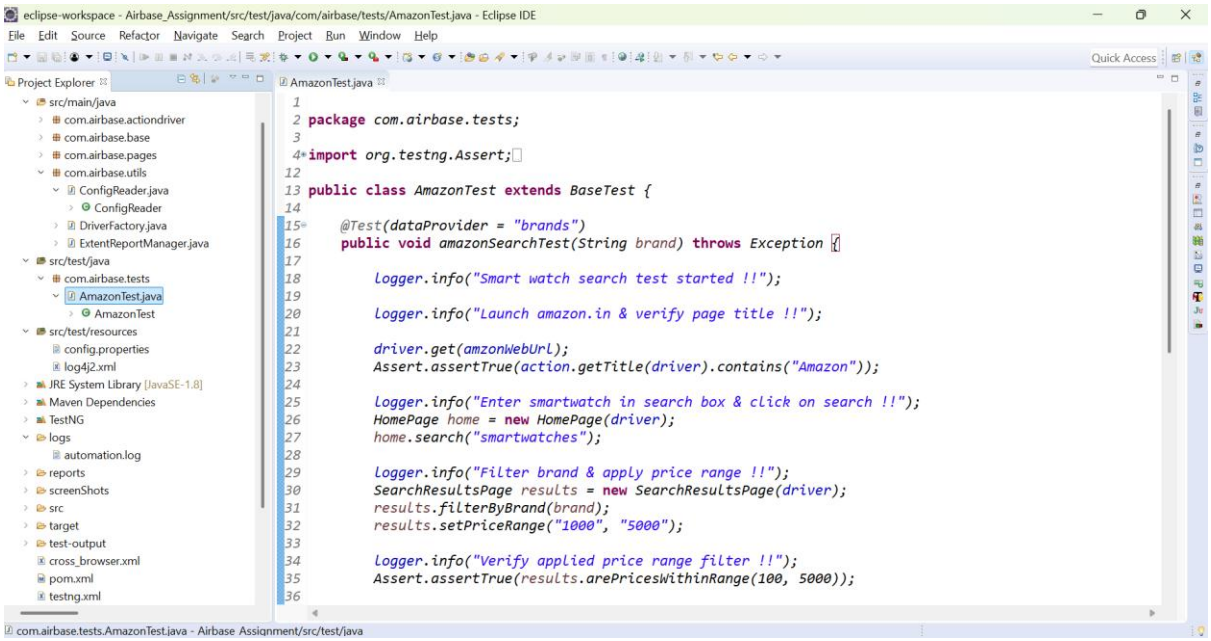
- *Select a product from the search results.*
- *Add the product to the cart.*
- *Verify that the product has been added to the cart.*
- *Close the new window and return to the main window.*
- *Test adding multiple different products to the cart.*
- *Verify that the cart count updates correctly after adding an item.*
- *Test adding an item to the cart when the user is not logged in.*
- *Try to add an item to the cart that is out of stock and verify the appropriate error message or behaviour.*
- *Verify the product details (name, price, image) in the cart match those of the product added.*
- *Test adding a product to the cart from different entry points (e.g. search results page, product details page, by clicking on buy now button).*
- *Verify that products remain in the cart across sessions*
- *Test modifying the quantity of an item directly within the cart.*
- *Test removing an item from the cart.*

Task 2: Automate the use case listed below using a testing framework of your choice. It'd be great if you make your framework modular, scalable and take care of reports and multi-browser scenarios.

Solutions:

This automation framework is built using Java, Selenium, and TestNG following a modular Page Object Model structure. It supports cross-browser execution, reusable action classes, Extent Reports for test reporting, and is designed for easy scalability and maintainability.

Framework architecture overview:



Package-Level Details:

- ***com.airbase.base -*** *Contains BaseTest which handles WebDriver initialization, setup, and teardown across different browsers via TestNG XML parameters.*
- ***com.airbase.pages*** *- Encapsulates all page-specific elements and actions, promoting reuse and maintainability using Page Factory.*
- ***com.airbase.actiondriver -*** *Abstracts common Selenium actions like click, type, getText into a reusable class ActionDriverImpl, improving readability and reducing boilerplate.*
- ***com.airbase.utils -*** *Includes core utilities like:*
    - ***DriverFactory –*** *launches browsers*
    - ***ConfigReader –*** *reads from config.properties*
    - ***ExtentReportManager –*** *sets up and manages Extent HTML reports*

- ***com.airbase.tests –*** *Actual test implemented with TestNG annotations and @DataProvider, e2e test flows using the page classes and action driver.*

Resource Files:

- ***config.properties -*** *Stores runtime config like browser type, URL, and environment-specific properties.*
- ***log4j2.xml -*** *Controls logging levels and outputs detailed test logs into the /logs folder.*

TestNG XML Configuration:

- ***testng.xml -*** *Main test suite file that controls which tests to run, allows parameterization (like browser type), and sets parallel execution/thread count.*
- ***cross_browser.xml*** *- Enables running the same test across multiple browsers in parallel using <parameter name="browser" value="chrome"/> (Firefox or Edge).*

Output Directories:

- ***logs/ -*** *Captures logs for debugging and traceability (automation.log).*
- ***reports/ -*** *Stores Extent Reports for test result in HTML format.*
- ***screenShots/ -*** *Saves screenshots when tests fail (helpful for debugging).*
- ***test-output/*** *- Default TestNG report directory with execution summaries.*

Please find the complete framework implementation in the attached ZIP file.