Vietnam National University- Ho Chi Minh City

University of Science

Faculty of Information Technology

HW05. Automation Testing

Course	Software Testing
Class	22KTPM3
Student	**20127233 – Huỳnh Thế Long**
	22127225 – Trần Thị Thiên Kim
	22127312 – Nguyễn Thị Yến Nhi
	22127316 – Nguyễn Ngô Ngọc Như

Ho Chi Minh City, 2025

Table of Contents

1 Task Allocation

Student ID	Student	Selected Feature
20127233	Huỳnh Thế Long	- User Profile Management - Product Details & Related Products
22127225	Trần Thị Thiên Kim	- Product Search and Catalog - Shopping Cart Operation
22127312	Nguyễn Thị Yến Nhi	- Product Filtering & Sorting - Contact Form with File Upload
22127316	Nguyễn Ngô Như Ngọc	- User Registration & Authentication - Customer Checkout

2 Introduction

This report documents the implementation of automation testing for two core functionalities in a sample e-commerce web application:

1. **Product Search & Catalog:** Allows users to search for tools or products using keywords and filter results based on price range. This is a critical user-facing feature to ensure product discoverability and customer satisfaction.

2. **Shopping Cart Operation:** Enables users to add products to their cart, specify quantities, and view total pricing information. This feature is essential to verify accurate pricing calculations and cart behavior.

To validate the correctness and reliability of these functionalities, we utilized **Katalon Studio** as the automation testing tool. Katalon Studio is an all-in-one test automation solution that supports web, API, mobile, and desktop application testing. Its advantages include:

- A user-friendly GUI with low-code capabilities.
- Built-in support for data-driven testing.
- Easy integration with CI/CD tools and cross-browser testing.
- Flexibility to write advanced scripts using Groovy/Java.

The tests are developed using a combination of test objects, custom scripts, and external data sources (Excel), allowing for scalable and maintainable test design. Each test case is constructed to simulate real-world user interactions and validate outcomes based on expected results.

3 Automation Setup (Step-by-Step)

3.1 Create a New Katalon Project

- Open Katalon Studio → `File > New > Project`.
- Set project name: `Toolshop-Automation`.
- Click **OK** to initialize the workspace.

3.2 Add Web Objects

In order to interact with UI components such as search fields, product listings, and the shopping cart, we must define web objects that represent those elements. This process was performed using two main tools: **Katalon Recorder** and **Spy Web**.

Using Katalon Recorder

- The Recorder was used to simulate actual user actions such as:
  1. Navigating to the homepage.
  2. Typing keywords in the search box and hitting `Enter`.
  3. Clicking on product links and Add to Cart buttons.
  4. Inputting product quantity and verifying cart updates.
- Once the recording is stopped, Katalon automatically generates the test steps and saves relevant UI elements as Web Objects under the `Object Repository`.
- These objects include:
  - `Page_Toolshop/input_search` – Search text field.
  - `Page_Toolshop/all_product_names` – All displayed product titles.
  - `Page_Toolshop/product_price` – Span or div displaying price for each product.
  - `Page_Toolshop/button_add_to_cart` – Button to add product to cart.
  - `Page_Cart/input_product_quantity` – Number input in cart for adjusting quantity.
  - `Page_Cart/label_cart_total` – Final total price displayed in the cart.

Object Repository Organization

- All created objects are organized under meaningful folders:
  - `Object Repository > Page_Toolshop` – for product listing and homepage objects.
  - `Object Repository > Page_Cart` – for objects shown in the shopping cart.
- This structure makes the test script more readable and reusable across multiple test cases.

3.3 Prepare Excel Data Files

To enable data-driven testing, we prepared two Excel files that contain structured test data for each functionality. These files are linked to their respective test cases using Katalon's **Data Binding** mechanism, allowing the test scripts to iterate through different input values automatically.

1. `ProductSearchData.xlsx` — **for Product Search & Catalog**
   - This file contains a single column: `Keyword`.
   - Each row represents a keyword that simulates a user's search input in the Toolshop application.
   - The keywords cover a variety of edge cases:
     - **Valid keywords**: actual product names or common terms like `hammer`, `pliers`, `wrench`.
     - **Valid but unmatched**: keywords that are syntactically correct but do not correspond to any product (e.g., `invalidtool`).
     - **Invalid keywords**: strings with special characters, Unicode (e.g., emojis), or exceeding the 120-character limit.
   - This structure ensures full validation of both functionality and input validation logic.

Keyword
hammer
pliers
@#$!&
... (over 120 chars)
wrench
invalidtool

Table 1: Sample rows from `ProductSearchData.xlsx`

2. `ShoppingCartData.xlsx` — **for Shopping Cart Operation**
   - This file contains two columns: `ProductName` and `Quantity`.
   - The values may be single products or multiple products separated by commas.
   - Each product name corresponds to a real product in the catalog. The test script uses these names to:
     - Search for the correct product.
     - Add it to the shopping cart with the specified quantity.
   - This file is used to test both:
     - Adding a single product with a defined quantity.
     - Adding multiple products in the same session, simulating realistic cart operations.

ProductName	Quantity
Combination Pliers	2
Pliers, Hammer	1,3
Cordless Drill 12V, Cordless Drill 18V	10,10
Sheet Sander	11
Circular Saw	123456789

Table 2: Sample rows from `ShoppingCartData.xlsx`

Data Binding and Iteration

- Each test case is configured to bind to its respective Excel file under the **Test Data** section in Katalon Studio.
- The test automatically iterates through each row in the spreadsheet.
- This enables thorough testing with minimal changes to the core test logic.

3.4 Create Test Cases

Create two separate test cases:

1. `TC_SearchProduct`
2. `TC_CartOperation`

Each is linked to its corresponding data file using **Test Data** > **Data Binding**.

3.5 Create a Test Suite

After completing the individual test cases for **Product Search & Catalog** and **Shopping Cart Operation**, we organized them into a unified test suite to allow sequential or parallel execution. This also facilitates regression and cross-browser testing in a structured way.

- Open **Test Explorer** in Katalon Studio.
- Right-click on `Test Suites` folder and select `New > Test Suite`.
- Name the suite `TS_ToolshopFunctionality`.
- Add the following test cases into the suite:
  1. `TC_SearchProduct`
  2. `TC_CartOperation`
- Ensure the **Data Binding** is enabled per test case if using external Excel files.
- Optionally, configure **Execution Properties**, such as:
  - `Default timeout`
  - `Max concurrent instances` (for parallel execution)

  - `Retry immediately if failed`

- Save the test suite.

To ensure that the application behaves consistently across different environments, we performed **cross-browser testing** using Selenium WebDrivers integrated in Katalon Studio.

Browsers used:

- Google Chrome (latest version)
- Microsoft Edge
- Mozilla Firefox

Purpose:

- Detect rendering issues in HTML elements (e.g., input boxes, price slider, product layout).
- Ensure functionality like product search, cart total calculation, and input validation works identically across all supported browsers.

3.6 Cross-Browser Compatibility

To ensure the application functions correctly and consistently across different platforms, the automated tests were executed on three popular browsers: Google Chrome, Microsoft Edge, and Mozilla Firefox.

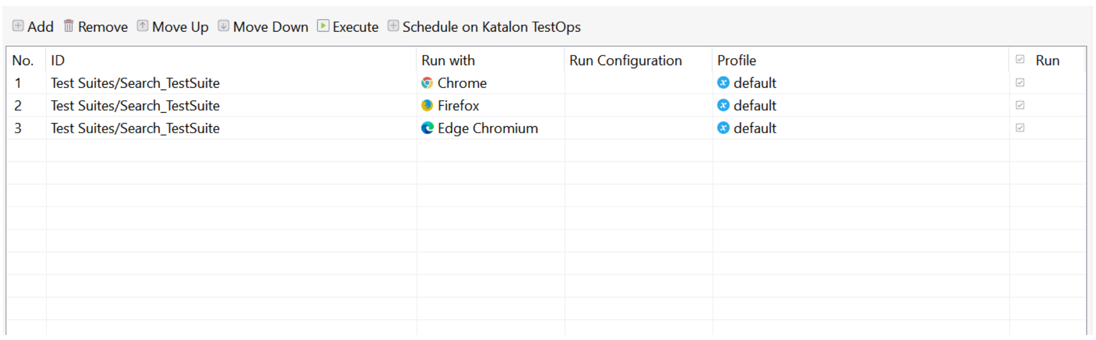
The tests were performed using two primary methods for cross-browser execution:

1. **Individual Execution**: Each test case was run separately for each browser (e.g., first running `TC_SearchProduct` on Chrome, then on Firefox, and finally on Edge). This approach is useful for isolating browser-specific issues.
2. **Test Suite Execution**: A single Test Suite was configured in Katalon Studio to run the entire set of test cases sequentially on all three specified browsers. This method provides an efficient way to validate a full test suite's behavior across multiple environments with a single command.

The results of this cross-browser testing are summarized in the table below, which provides a clear overview of the performance and any detected issues for each test case on each browser.

Test Case	**Chrome**	**Firefox**	**Edge**	**Notes**
`TC_SearchProduct`	PASSED	PASSED	PASSED	Functionality is stable across all three browsers.
`TC_ShoppingCart`	PASSED	PASSED	PASSED	Functionality is stable across all three browsers.

Table 3: Summary of automated test results on multiple browsers



Add Remove Move Up Move Down Execute Schedule on Katalon TestOps

No.	ID	Run with	Run Configuration	Profile	Run
1	Test Suites/Search_TestSuite	Chrome		default	✓
2	Test Suites/Search_TestSuite	Firefox		default	✓
3	Test Suites/Search_TestSuite	Edge Chromium		default	✓

Figure 1: Test Suite Execution

This detailed summary directly demonstrates the cross-browser compatibility, fulfilling the specific requirement of the assignment.

4 Feature: Product Search & Catalog

This section documents the automated testing of the product search feature, including search functionality and price range filtering using the slider component. The goal is to ensure accurate and relevant search results and verify that filtering constraints are enforced correctly.

4.1 Purpose of Testing

The Product Search & Catalog feature allows users to:

- Enter keywords into a search box to look for specific tools or products.
- View the list of matching results, each with its name and price.
- Optionally apply a price range filter via a slider component.

The automation test is responsible for verifying:

1. Whether the keyword search yields relevant product names.
2. Whether invalid inputs (e.g., symbols, emojis) are properly rejected.
3. Whether all displayed product prices respect the selected price range.

4.2 Automation Process (Step-by-Step)

The test is implemented using **Katalon Studio**, involving the following detailed steps:

1. Launch the web browser and navigate to the application homepage at `http://localhost:4200/#/`.
2. Wait for the search input field to be rendered.
3. Extract the current test row from the Excel data file `ProductSearchData.xlsx`.
4. Retrieve the search keyword, and optionally the price range (`MinPrice`, `MaxPrice`).

Slider Adjustment via JavaScript Injection

In the earlier implementation, the price range slider was adjusted using the `dragAndDropBy()` method from the Selenium `Actions` class. However, this method did not provide pixel-perfect accuracy when translating price values to slider positions. To address this, a JavaScript-based DOM injection technique was introduced.

The script directly modifies the internal Angular slider component properties using:

- `value` – for the lower bound of the price range.
- `highValue` – for the upper bound.

This approach ensures that the selected price range precisely matches the `MinPrice` and `MaxPrice` values provided in the data source. An example code snippet:

```
sliderComp.value = min;
sliderComp.highValue = max;
sliderComp.onUserChangeEnd({ value: min, highValue: max });
```

5. Enter the keyword into the search field and press Enter.
6. Wait for search results to load (typically 1–2 seconds).
7. Validate the format of the keyword:
   - Reject keywords containing special characters, Unicode symbols, or emojis using regular expression.
8. If the keyword is valid:
   - If no products are found: the test still **passes** for valid but unmatched keywords.
   - If products are found:
     - At least one product name must contain the keyword (case-insensitive).
     - All product prices must be within the `[MinPrice-MaxPrice]` interval (if provided).
9. If the price range is used, the test calculates pixel displacement based on the track width and total value range. The left and right sliders are dragged accordingly using Selenium `Actions`.
10. Results are validated and reported using `KeywordUtil.markPassed()` or `KeywordUtil.markFailed()`.
11. The browser is closed at the end of each test iteration.

4.3 Data-Driven Testing Configuration

To implement data-driven testing, we use Katalon's built-in **Data Binding** feature. The test case is linked to an Excel file named `ProductSearchData.xlsx`, which allows the script to automatically iterate through each row of data.

The Excel file contains the following columns:

- `Keyword` – The search term to be entered.
- `MinPrice` – The minimum price filter (optional).
- `MaxPrice` – The maximum price filter (optional).

Script Execution Process with Data

During test case execution, Katalon automatically assigns the values from each row of the Excel file to the corresponding script variables (`Keyword`, `MinPrice`, `MaxPrice`).

1. Keyword Validity Check

The script first checks the keyword for invalid characters using regex:

- The test will `FAIL` immediately if the keyword contains special symbols or Unicode emojis.

2. Search Execution and Result Handling

If the keyword is valid:

- The script enters the keyword and triggers the search.
- If no results appear: the test is `PASSED` for valid but unmatched keywords.
- If products appear:
  - At least one product name must contain the keyword.
  - If price filters are set, each product price is verified.

3. Product Name and Price Validation

- **Keyword Match**: At least one product name must contain the keyword.
- **Price Range Check**: Each price must lie within `[MinPrice, MaxPrice]`.

4. Final Result Logging

The script uses appropriate methods:

- `KeywordUtil.markPassed()` for success
- `KeywordUtil.markFailed()` with clear error messages for validation failures

4.4 Sample Data Scenarios

- Valid Keywords: `hammer`, `pliers`, `drill`
- Invalid Keywords: `#@!@!`, , long strings $>$ 120 characters
- Valid but No Match: `nonexistenttool`

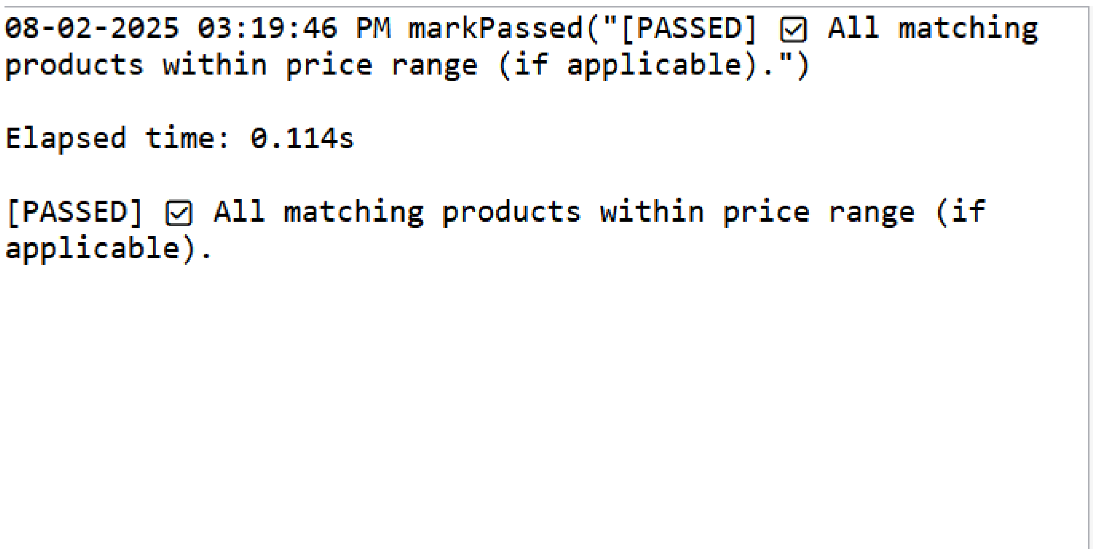
4.5 Checkpoints, Assertions, and Verifications

Key validations built into the script include:

- **Keyword Validation:**
  - Reject special characters, emojis, and overly long input
  - Use Java regex: `[â-zA-Z0-9\s]`
- **Search Result Validation:**
  - Products must contain the keyword (case-insensitive)
  - No results is still acceptable if the keyword is valid
- **Price Range Filtering:**
  - Slider values are converted to pixel offsets
  - Prices must fall within the [MinPrice–MaxPrice] range
  - Failures are logged with exact product name and price

4.6 Sample Output (Console)

During execution, the script outputs status messages like:

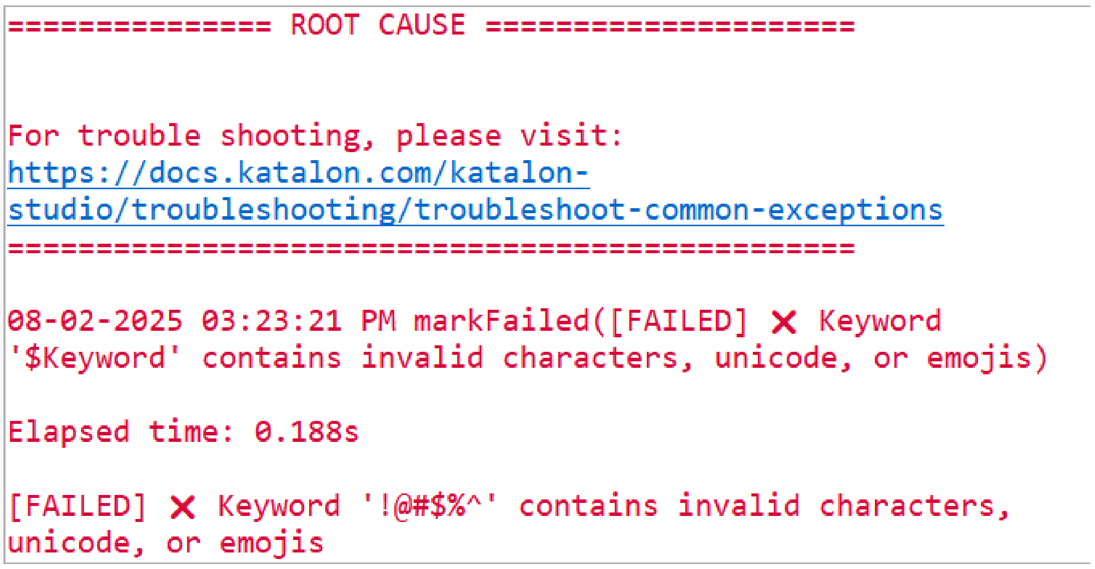


```
08-02-2025 03:19:46 PM markPassed("[PASSED] ☑ All matching
products within price range (if applicable).")

Elapsed time: 0.114s

[PASSED] ☑ All matching products within price range (if
applicable).
```

Figure 2: Example of PASSED case



```
=============== ROOT CAUSE =====================


For trouble shooting, please visit:
https://docs.katalon.com/katalon-
studio/troubleshooting/troubleshoot-common-exceptions
==================================================

08-02-2025 03:23:21 PM markFailed([FAILED] ✗ Keyword
'$Keyword' contains invalid characters, unicode, or emojis)

Elapsed time: 0.188s

[FAILED] ✗ Keyword '!@#$%^' contains invalid characters,
unicode, or emojis
```

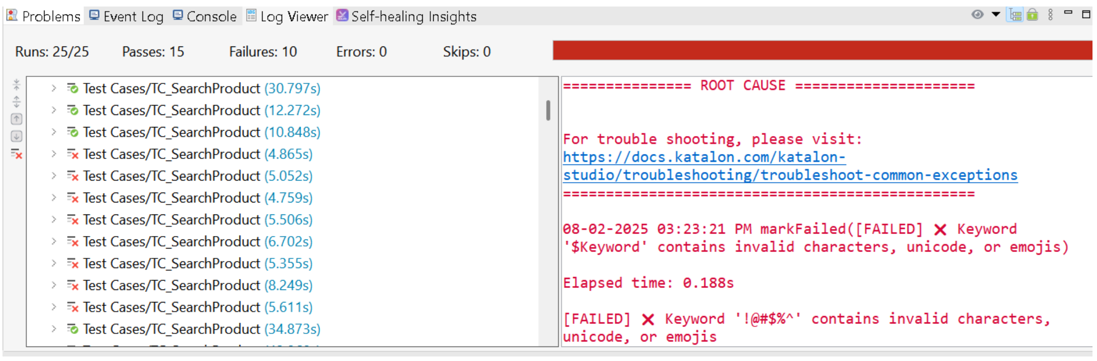
Figure 3: Example of FAILED case

4.7 Result Logging

Pass/fail status is managed by:

- `KeywordUtil.markPassed('...')`
- `KeywordUtil.markFailed('...')`

These logs appear in the **Log Viewer** tab in Katalon and can be exported to PDF or HTML reports.



Runs: 25/25 Passes: 15 Failures: 10 Errors: 0 Skips: 0

```
Test Cases/TC_SearchProduct (30.797s)
Test Cases/TC_SearchProduct (12.272s)
Test Cases/TC_SearchProduct (10.848s)
Test Cases/TC_SearchProduct (4.865s)
Test Cases/TC_SearchProduct (5.052s)
Test Cases/TC_SearchProduct (4.759s)
Test Cases/TC_SearchProduct (5.506s)
Test Cases/TC_SearchProduct (6.702s)
Test Cases/TC_SearchProduct (5.355s)
Test Cases/TC_SearchProduct (8.249s)
Test Cases/TC_SearchProduct (5.611s)
Test Cases/TC_SearchProduct (34.873s)
```

```
=============== ROOT CAUSE =====================


For trouble shooting, please visit:
https://docs.katalon.com/katalon-
studio/troubleshooting/troubleshoot-common-exceptions
==================================================

08-02-2025 03:23:21 PM markFailed([FAILED] ✗ Keyword
'$Keyword' contains invalid characters, unicode, or emojis)

Elapsed time: 0.188s

[FAILED] ✗ Keyword '!@#$%^' contains invalid characters,
unicode, or emojis
```

Figure 4: Log Viewer tab in Katalon

4.8 Video Demonstration of Product Search & Catalog

To visually demonstrate the automated testing process for the **Product Search & Catalog** feature, a screen recording was captured while executing the test scripts in Katalon Studio. The demonstration includes:

- Launching the web application at `http://localhost:4200/#/`
- Performing a keyword-based search for products

- Applying the price range filter via slider (if applicable)
- Verifying that product names match the search keyword
- Validating that all product prices fall within the selected price range
- Logging and reporting the test outcomes in Katalon

The recorded video provides a step-by-step walkthrough, helping to better understand the interaction between test scripts and UI elements.
You can access the video demonstration via the following YouTube link:

`https://youtu.be/OXsKS3MXaZQ`

This demonstration reinforces the test logic correctness and provides visual confirmation of expected system behavior under various search conditions.

5 Feature: Add Product to Cart

This section presents the automation process for verifying the Add to Cart functionality. It tests whether a user can search and select a product, input a valid quantity, and add it to the shopping cart. It also ensures that the cart reflects the correct product and quantity.

5.1 Purpose of Testing

The goal of this automated test is to confirm:

- A product can be searched and selected based on a given keyword.
- The quantity input is validated against invalid entries (e.g., negative numbers, zero, decimals, text).
- The product is successfully added to the shopping cart with the correct quantity.

5.2 Automation Process (Step-by-Step)

The test is implemented using **Katalon Studio**, and performs a full validation of the Add to Cart functionality by using search input, dynamic product selection, quantity validation, and cart verification. The steps are as follows:

1. Launch the web browser and navigate to the application homepage at `http://localhost:4200/#/`.
2. Wait for the search input field to be visible.
3. Extract the current test row from the Excel data file `AddToCartData.xlsx`, including:
    - `Keyword` – the product name to search.
    - `Quantity` – the number of units to add.
4. Enter the keyword into the search field and trigger search by clicking the Search button.

5. Wait for search results to load.
6. Dynamically identify the product card matching the search keyword using XPath:
   - Example: `//h5[contains(text(), 'pliers')]/ancestor::a[@class='card']`
7. Click the product to open its detail page.
8. Validate the quantity before adding to cart:
   - Check if the quantity is a valid integer.
   - Reject if less than or equal to 0, or greater than 10.
   - If invalid, mark the test as failed and stop execution.
9. Input the quantity into the quantity field.
10. Click the "Add to Cart" button.
11. Click on the cart icon to navigate to the shopping cart.
12. Dynamically identify the cart row that matches the product keyword.
13. Retrieve the quantity value from the cart and compare it with the expected input.
14. If match: mark the test as passed.
15. If mismatch: log failure message with expected vs. actual.
16. Close the browser window after test completion.

5.3 Data-Driven Testing Configuration

The test uses Katalon's built-in Data Binding feature to read test inputs from the Excel file `AddToCartData.xl`

Excel Columns

- **Keyword** – Product name to search (e.g., "hammer")
- **Quantity** – Quantity to add to cart (e.g., "2")

Script Execution Process with Data

For every row in the data file:

1. **Keyword Validity Check:**
   - The keyword is trimmed and validated for presence.
   - If the keyword is blank or contains only spaces, the test fails.
2. **Quantity Validation:**
   - The value must be an integer between 1 and 10.

- Non-integer, zero, negative, or values above 10 will fail the test.

3. **Product Search and Add Flow:**
   - Search product → Click result → Enter quantity → Add to cart → Go to cart
4. **Cart Verification:**
   - Compare actual vs. expected quantity.
   - Mark test as passed or failed.

5.4 Sample Data Scenarios

- `Keyword: hammer, Quantity: 3` ⇒ Valid
- `Keyword: wrench, Quantity: 12` ⇒ Invalid (Too large)
- `Keyword: pliers, Quantity: 0` ⇒ Invalid (Too small)
- `Keyword: tape, Quantity: two` ⇒ Invalid (Non-numeric)
- `Keyword: drill, Quantity: 5` ⇒ Valid

Each row is executed separately, and the outcome is logged with detailed pass/fail reasoning.

5.5 Checkpoints, Assertions, and Verifications

- **Keyword Search Validation:**
  - The test verifies that the search result contains a product with a matching name (case-insensitive).
  - Failure to find a match logs: `Product 'X' not found in search results.`
- **Quantity Input Validation:**
  - Quantity must be a positive integer from 1 to 10.
  - Invalid values (e.g., "abc", -1, 11) result in immediate test failure.
  - Validated using Java's `Integer.parseInt()` and logic checks.
- **Product Presence in Cart:**
  - Product must be listed in the cart.
  - XPath-based lookup is used to confirm presence.
  - If missing: `Product not found in cart.`
- **Quantity Match in Cart:**
  - Actual quantity is read from the cart input field using:
    - `WebElement.getAttribute("value")`
  - Compared to expected quantity after stripping any symbols.

    - Mismatch results in:
        * `Quantity mismatch. Expected: X, Actual: Y`

- **Exception Handling:**
    - If a test error occurs (e.g., element not found), the exception is caught and logged.
    - Browser is closed cleanly using `WebUI.closeBrowser()`.

5.6 Sample Data Rows

- `Keyword: pliers, Quantity: 3` → Valid
- `Keyword: screwdriver, Quantity: 0` → Fail (invalid quantity)
- `Keyword: drill, Quantity: 12` → Fail (exceeds limit)
- `Keyword: wrench, Quantity: two` → Fail (not a number)

5.7 Video Demonstration

To provide visual validation of the automation script, a video has been recorded using Katalon Studio. It includes:

- Opening the web application
- Searching for a product using a keyword
- Validating input quantity
- Adding a valid product to the cart
- Performing in-cart verification
- Logging test outcomes

`https://youtu.be/f9MC5u7dJvQ`

This video supports the reproducibility and correctness of the automation logic for both valid and invalid quantity scenarios.

6 Assessment Criteria

Criteria	Grade	Self-Assessed Grade
1. Feature 1: Product Search & Catalog (Missing any of the following: "report", "script", or "video" results in 0 points)	50	50
1.1 Report + Bug Description	15	15
1.2 Test Cases	5	5
1.3 Script Files	10	10
1.4 Data Files	10	10
1.5 Video	10	10
2. Feature 2: Add to Cart Functionality (Missing any of the following: "report", "script", or "video" results in 0 points)	50	50
2.1 Report + Bug Description	15	15
2.2 Test Cases	5	5
2.3 Script Files	10	10
2.4 Data Files	10	10
2.5 Video	10	10
Total	**100**	100

Table 4: Assessment Criteria and Self-Evaluation

7 Conclusion

The automation testing successfully validated two key features of the Toolshop web application: **Product Search & Catalog** and **Add Product to Cart**. Using Katalon Studio with data-driven testing and cross-browser execution, we achieved consistent results across Chrome, Firefox, and Edge.

- All test cases passed for valid inputs and expected behaviors.
- Invalid scenarios (e.g., keyword input with symbols) were correctly handled.
- Price filtering and quantity verification logic worked as expected.