University of Science – Vietnam National University
Faculty of Information Technology

CS423 – Software Testing

HW06 Report
Automation Testing

Student's Name	**Student ID**	**Class**
Tran Nhat Thanh	22125093	22TT

Ho Chi Minh City, December 4th, 2025

Contents

1. Overall Analysis

This report details the automated testing process for three features of the *Toolshop v5.0-with-bugs* application: Add to Cart (Guest), Edit Profile (User), and *Add/Edit Product (Admin)*. The testing strategy utilized Data-Driven Testing and Domain Testing (Equivalence Partitioning & Boundary Value Analysis) to ensure robust coverage.

Overall Status: The application exhibits significant defects, particularly in User Experience (UX) and core business logic (Cart Calculation, Product Management, etc.).

- **Total Test Cases Executed:** ~100 (across 3 features)
- **Overall Test Status:**
  - Feature 1 (Cart): FAILED (Critical logic & UX bugs).
  - Feature 2 (Profile): FAILED (Blocker backend bugs).
  - Feature 3 (Product): PASS with Warnings (Functional success but poor UX).

2. Test Environment

- **Target URL:** http://localhost:4200/#/ (Local Deployment of v5.0 with bugs)
- **Automation Framework:**
  - **Language:** Python 3.x
  - **Library:** Selenium WebDriver 4.x
  - **Driver Management:** webdriver-manager (Auto-downloads Chrome/Firefox/Edge drivers)
  - **Data Source:** CSV files (pandas library used for parsing)
- **Browsers Tested:**
  - Google Chrome
  - Mozilla Firefox
  - Microsoft Edge

3. Test Methodology

A. Data-Driven Testing (DDT)

To ensure maximum coverage without redundancy, we separated *Test Logic* from *Test Data.*

- **Scripts:** Reusable Python scripts (test_cart.py, test_profile.py and test_product.py) containing the Selenium interaction logic.
- **Data:** CSV files containing inputs, expected values, and error messages for valid/invalid partitions and boundaries.

B. Verification Strategy

I implemented a "Strict Validation" protocol to catch both functional failures and UI inconsistencies:

1. **Functional Check:** Does the data change? (checking the Cart Table for the item).
2. **UX Check:** Does the system provide the correct feedback? (Green Toast for success, Red Toast for error).

3. **Data Check:** Are calculations correct? (Line Item Total vs. Grand Total).

4. Detailed Analysis

A. Feature 1: Guest - Add to Cart

I. Workflow & Script Logic

The script test_cart.py simulates a Guest user browsing for products and adding them to the cart, replicating realistic user behavior.

1. **Reset:** Before every test case, the script performs a "Nuclear Reset" by clearing LocalStorage, SessionStorage, and Cookies, ensuring a total clear state (0 items in cart).
2. **Product Discovery (Pagination Scanning):** The script scans the product grid. If the target product ("Hammer" or "Pliers") is not found on the current page, it automatically scrolls to the bottom, identifies the "Next (»)" button, clicks it, and repeats the scan on the subsequent page until the item is located.
3. **Interaction:**
   - Navigate to the product detail page.
   - Clear the quantity input field using a robust method (Ctrl+A → Delete) to prevent input concatenation errors.
   - Enter the test quantity (Valid or Invalid).
   - Click "Add to Cart" and immediately listen for any Toast Notification (Success or Error) to capture messages.
4. **Verification:**
   - **Force Navigation:** Use JavaScript to click the "Cart" icon, by passing any overlay blocking the UI.
   - **Table Scan:** Iterate through the Cart table rows to verify that the specific Product Name exists and the Quantity matches the expected value.
   - **Financial Check:** Verify the Line Item Total for that row (detecting the $0.00 bug) and the Grand Total at the bottom of the page.

II. Data Organization (cart_data.csv)

The automation framework uses a structured CSV file to drive the test execution. This allows a single Python script to execute 32 distinct test scenarios ranging from simple inputs to complex multi-step workflows.

1. Data Schema

The CSV file is organized into 7 control columns that guide the test behavior:

Column Name	Purpose	Example Data
TC_ID	Unique identifier for tracking results and mapping back to the Test Plan.	ATC-EP-04
Description	Short summary of what the test verifies.	Valid Cumulative Add

Flow_Type	*The Logic Controller*. Tells the script which execution path to take (see below).	Simple, Cumulative
Product_Name	Specifies the exact product to search for and interact with. Supports single items or comma-separated lists for multi-product tests.	Hammer, Hammer,Pliers
Input_Value	The data entered into the Quantity field. Supports integers, decimals, text, or comma-separated steps for sequential actions.	5, abc, 5,5
Expected_Qty	The final quantity expected in the Cart table. If 0, implies the action should be blocked.	10
Expected_Error	Dual-purpose assertion column: 1. **Text:** The specific error message expected in the toast ("Limit reached"). 2. **Price:** For price checks, this holds the expected total amount ("60.05").	Limit is 10 or 60.05

2. Execution Flow Logic (Flow_Type)

The script reads the Flow_Type column to determine the complexity of the test case:

- **Simple:** The standard flow. Resets state → Finds Product → Adds Item → Verifies Cart.
- **Cumulative:** Tests state persistence. The script parses Input_Value as a list ("5,5"), performing multiple "Add" actions sequentially to verify if the backend *sums* quantities correctly or *overwrites* them.
- **MultiProd:** Tests cart capacity. It iterates through a list of products (Hammer,Pliers) adding them one by one before performing a final cart verification.
- **CheckCart:** A specialized flow that adds an item and prioritizes *Price Verification* (Line Item Total & Grand Total) over standard toast error checking.
- **ComplexReset:** A stress test that Adds items → Deletes them via the Cart page → Returns Home → Adds items again. This verifies that the "Delete" function cleans up the application state correctly.
- **UIStepper:** Bypasses the text input field to interact directly with the + and – buttons, verifying UI responsiveness.

III. Automation Test Results

1. How Results Were Captured

All cart test cases were executed using the unified automation script, which outputs results into CSV files for Chrome, Firefox, and Edge.

Each record stores: TC_ID, description, browser, PASS/FAIL, and details.

Please check the *attached csv files* for more details.

2. Summary of Results by Browser

Chrome

- Total: 32
- Pass: 11, Fail: 21
- Main failure patterns:
  - Error toast ("Oeps…") appears even for valid inputs.
  - Cumulative quantity incorrect.
  - Line total stays $0.00 although grand total is correct.
  - Stepper buttons (+/–) not working.
  - Delete button fails.
  - Decimal inputs incorrectly added to cart.

Firefox

- Total: 32
- Pass: 11, Fail: 21
- Same issues as Chrome → confirms defects are system-wide, not browser-specific.

Edge

- Total: 32
- Pass: 11, Fail: 21
- Same failure behavior as Chrome/Firefox.

3. Cross-Browser Conclusion

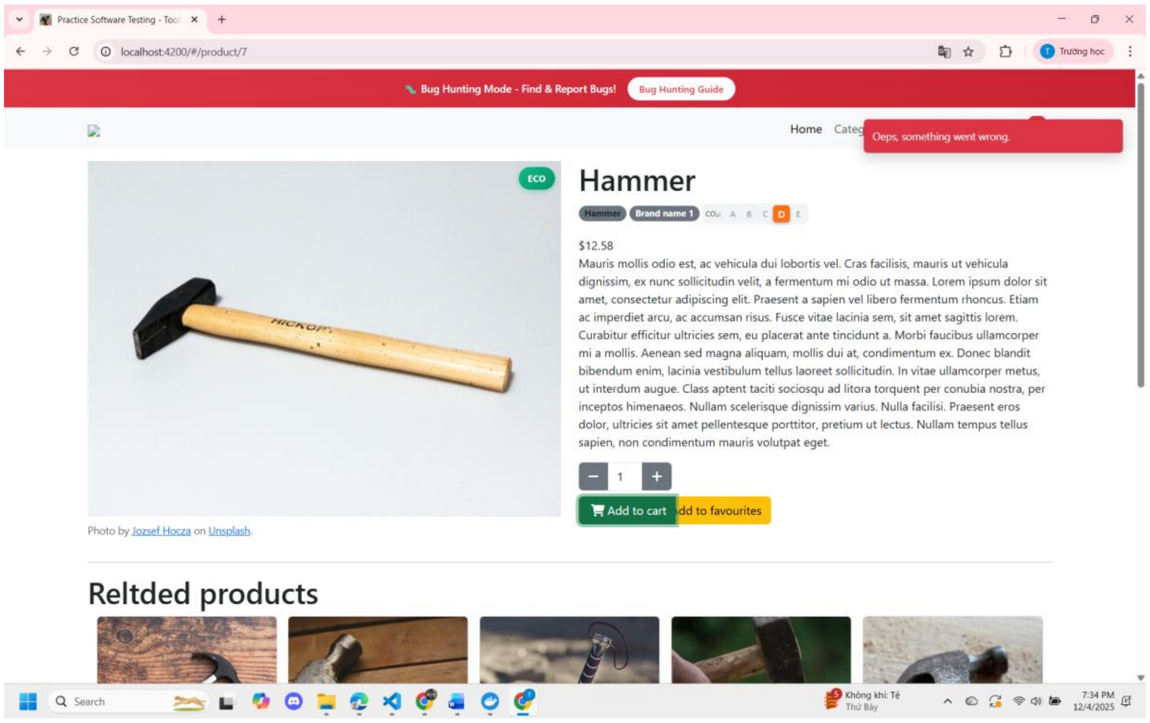
Across all three browsers, the results are consistent:

- Valid add-to-cart actions fail due to the incorrect error toast.
- Cumulative cart logic is broken.
- Line totals are wrong ($0.00).
- Stepper and delete actions malfunction.
- Input validation for decimals is incorrect.

Overall, this feature does not function properly and fails the majority of core functional scenarios.

IV. Discovered Bugs

- **Bug 1: False Error Toast ("Oeps").**
  - *Scenario:* Successfully add a valid item (1 Hammer).
  - *Actual:* A red error toast appears: *"Oeps, something went wrong."*



Oeps, something went wrong.

Hammer

$12.58

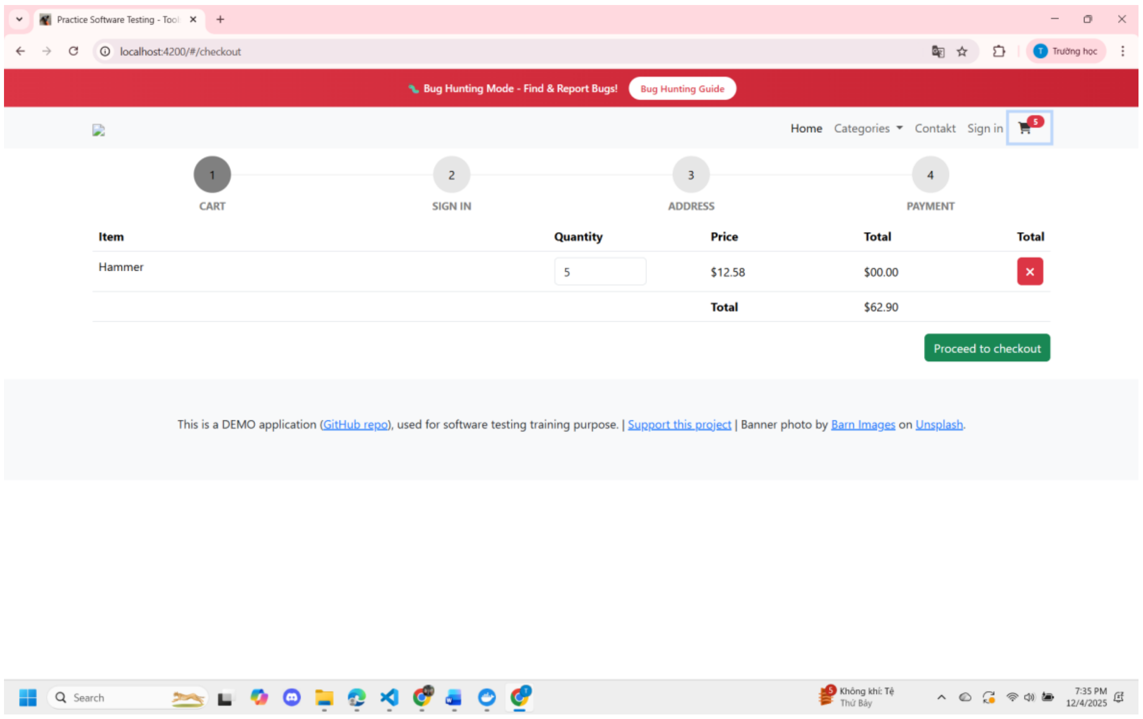
Mauris mollis odio est, ac vehicula dui lobortis vel. Cras facilisis, mauris ut vehicula dignissim, ex nunc sollicitudin velit, a fermentum mi odio ut massa. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent a sapien vel libero fermentum rhoncus. Etiam ac imperdiet arcu, ac accumsan risus. Fusce vitae lacinia sem, sit amet sagittis lorem. Curabitur efficitur ultricies sem, eu placerat ante tincidunt a. Morbi faucibus ullamcorper mi a mollis. Aenean sed magna aliquam, mollis dui at, condimentum ex. Donec blandit bibendum enim, lacinia vestibulum tellus laoreet sollicitudin. In vitae ullamcorper metus, ut interdum augue. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nullam scelerisque dignissim varius. Nulla facilisi. Praesent eros dolor, ultricies sit amet pellentesque porttitor, pretium ut lectus. Nullam tempus tellus sapien, non condimentum mauris volutpat eget.

Photo by Jozsef Hocza on Unsplash.

Reltded products

Figure 1: Unexpected error toast appearing after a valid Add to Cart action.

- **Bug 2: Cumulative Logic Failure.**
  - *Scenario:* Add 5 items, then add 5 more of the same item.
  - *Expected:* 10 items in cart.
  - *Actual:* 5 items. The system overwrites previous quantity instead of adding to it.



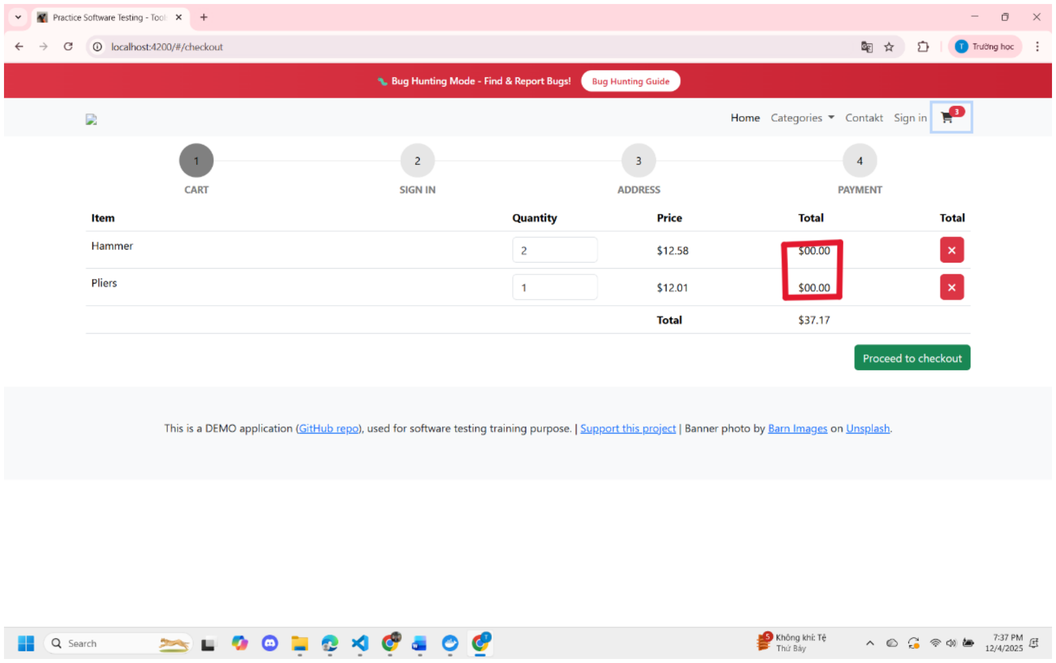
Item	Quantity	Price	Total	Total
Hammer	5	$12.58	$00.00	✕
		Total	$62.90	

This is a DEMO application (GitHub repo), used for software testing training purpose. | Support this project | Banner photo by Barn Images on Unsplash.

Figure 2: Cart quantity overwritten instead of accumulated after adding the same item twice.

You can watch the Add to Cart demo from 00:48 for a clearer view.

- **Bug 3: Line Item Price is always $0.00.**
  - *Scenario:* Add any item and view the Cart.
  - *Actual:* The "Total" column for the individual row displays $0.00, even though the Grand Total at the bottom is correct.



localhost:4200/#/checkout

Bug Hunting Mode - Find & Report Bugs! Bug Hunting Guide

Home Categories Contakt Sign in

1 CART 2 SIGN IN 3 ADDRESS 4 PAYMENT

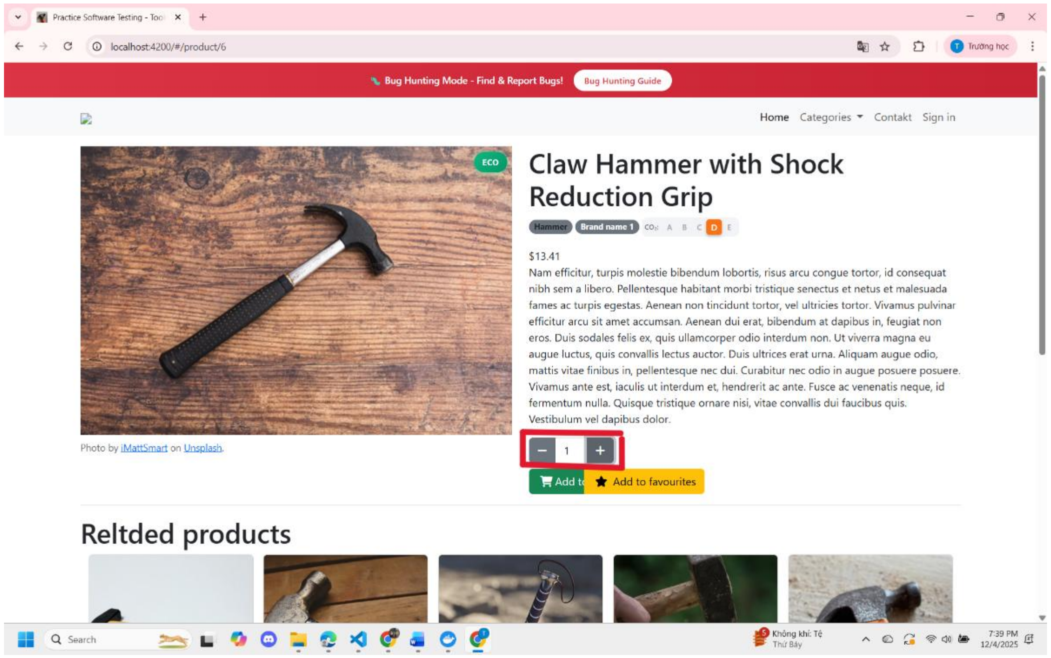
Item	Quantity	Price	Total	Total
Hammer	2	$12.58	$00.00	×
Pliers	1	$12.01	$00.00	×
		Total	$37.17	

Proceed to checkout

This is a DEMO application (GitHub repo), used for software testing training purpose. | Support this project | Banner photo by Barn Images on Unsplash.

Figure 3: Cart displays a $0.00 line-item total despite correct grand total.

- **Bug 4: Broken Stepper Buttons.**
  - *Scenario:* Click + or – on the product detail page.
  - *Actual:* Buttons are unresponsive.



localhost:4200/#/product/6

Bug Hunting Mode - Find & Report Bugs! Bug Hunting Guide

Home Categories Contakt Sign in

Claw Hammer with Shock Reduction Grip

Hammer Brand name 1 CO₂ A B C D E

$13.41

Nam efficitur, turpis molestie bibendum lobortis, risus arcu congue tortor, id consequat nibh sem a libero. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean non tincidunt tortor, vel ultricies tortor. Vivamus pulvinar efficitur arcu sit amet accumsan. Aenean dui erat, bibendum at dapibus in, feugiat non eros. Duis sodales felis ex, quis ullamcorper odio interdum non. Ut viverra magna eu augue luctus, quis convallis lectus auctor. Duis ultrices erat urna. Aliquam augue odio, mattis vitae finibus in, pellentesque nec dui. Curabitur nec odio in augue posuere posuere. Vivamus ante est, iaculis ut interdum et, hendrerit ac ante. Fusce ac venenatis neque, id fermentum nulla. Quisque tristique ornare nisi, vitae convallis dui faucibus quis. Vestibulum vel dapibus dolor.

– 1 +

Add to cart Add to favourites

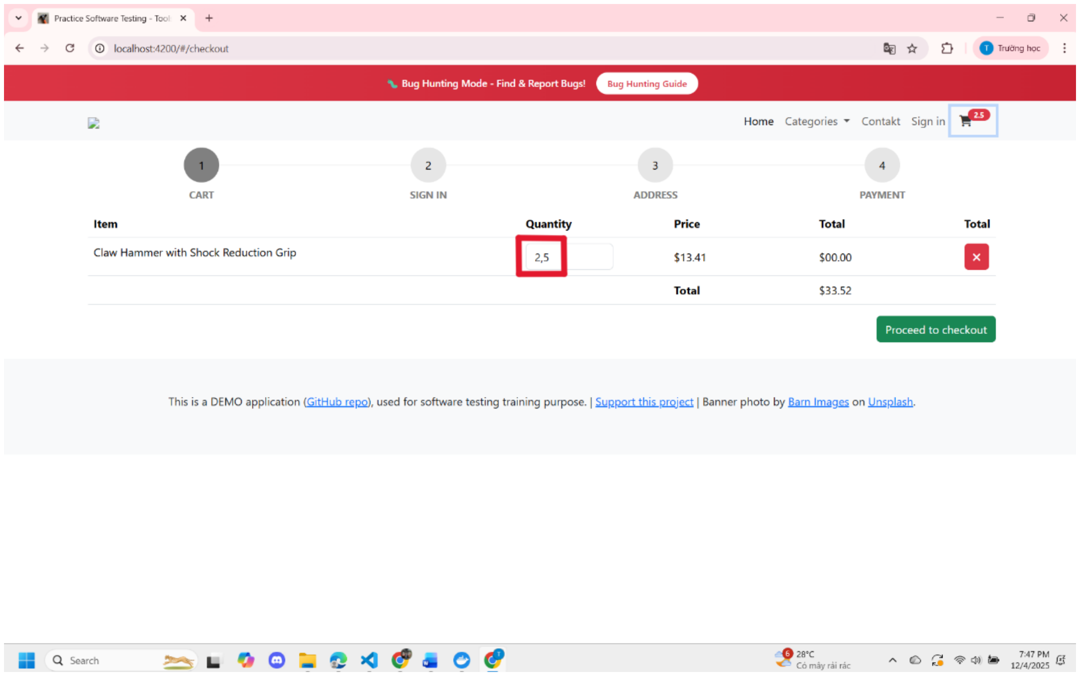
Photo by iMattSmart on Unsplash.

Reltded products

Figure 4: Quantity stepper (+/–) buttons unresponsive on the product detail page.

You can watch the Add to Cart demo from 04:32 to see clearer.

- **Bug 5: Input validation for decimals is incorrect.**
  - *Scenario:* On the product detail page, the Guest user enters 2.5 in the Quantity field and clicks "Add to Cart"
  - *Expected:* The system should reject the decimal input and show error message.
  - *Actual:* The cart accepts the value 2.5 and updates with a quantity of 2.5.



Item	Quantity	Price	Total	Total
Claw Hammer with Shock Reduction Grip	2,5	$13.41	$00.00	×
		Total	$33.52	

This is a DEMO application (GitHub repo), used for software testing training purpose. | Support this project | Banner photo by Barn Images on Unsplash.

Figure 5: System accepts decimal quantity input (2.5) instead of rejecting it.

V. Video Demo

Please check this link for the Add to Cart demo, the timestamps for each browser are listed in the description box.

B. Feature 2: User - Edit Profile

I. Workflow & Script Logic

The script test_profile.py simulates a logged-in customer updating their profile fields, focusing on both valid updates and server-side validation.

1. **Login & Session Setup**
   - The script launches the browser and navigates to the login page.
   - It logs in using a known customer account (customer@practicesoftwaretesting.com / welcome01).
   - Successful login is confirmed by waiting for the URL to contain account.
2. **Navigation to Edit Profile**

- For every test case, the script refreshes the page to clear any transient state and then forces navigation to #/account/profile.
- It waits for the profile form to be present before proceeding.

3. **Interaction**

   a) **Standard Update Cases**

   - Locate the input using [data-test='<field_name>'] (first-name, last-name, email, phone, etc.).
   - Use clear_and_type() to robustly clear the old value (click → Ctrl+A → Delete) before typing the new Input_Value from the CSV.
   - Locate "Update Profile" button and click it.
   - Immediately listen for a toast/alert using capture_toast(), which detects both success and error messages.

   b) **Data Persistence / Refresh Case**

   For test cases whose Description contains "Refresh" or "Persistence" (PROF-EP-19), the script:

   - Enters the new value into the selected field without clicking "Update Profile".
   - Refreshes the page.
   - Re-reads the field value to verify whether the unsaved change is discarded or persisted.

4. **Verification**

   - For **valid tests** (empty Expected_Error):
     - The script expects a non-error result.
     - If capture_toast() flags a red/error toast (contains "error", "wrong", "not found", or danger/error classes), the case is marked FAIL.
   - For **invalid tests** (Expected_Error is filled):
     - The script checks whether the actual toast text contains the expected message ("Numbers only", "Field required", "Too short").
     - If the expected string is not present, the case is marked FAIL with the actual toast content included in the log.
   - For **Refresh/Persistence**:
     - If the field reverts after refresh (value is not equal to the temporary input), the test is marked PASS.
     - If the temporary input survives a page refresh without saving, it is marked as an unexpected persistence and flagged as FAIL.

II. Data Organization (profile_data.csv)

The automation uses a compact CSV to drive all Edit Profile scenarios; one script can run all tests without code changes.

1. Data Schema

The CSV file is organized into 5 control columns that guide the test behavior:

Column Name	Purpose	Example Data
TC_ID	Unique identifier for tracking results and mapping back to the Test Plan.	PROF-EP-10
Description	Short summary of what the test verifies.	Invalid Email (No @)
Field_Name	Logical field identifier used to build the CSS selector ([data-test='first-name']).	email, phone
Input_Value	Value to type into the selected field (may be empty, text, numbers, or boundary strings).	test.com, 0987…
Expected_Error	Expected validation message for negative tests. Empty means the test expects success.	Invalid email format, Too short

2. Execution Flow Logic

The script uses the Description and Expected_Error values to decide how to treat each case:

- **Standard Field Update**
  - Default for most rows.
  - The script refreshes the page, navigates to the profile form, fills the specified field, clicks "Update Profile", and asserts the toast against Expected_Error (or absence of it).
- **Validation Tests**
  - All rows with a non-empty Expected_Error are treated as negative tests.
  - The script expects a specific human-friendly error message like "Field required", "Numbers only", "Too short", "Too long", etc.
  - If the app returns a generic backend error or an unrelated message, the test is considered failed.
- **Persistence / Refresh Flow**
  - For PROF-EP-19 ("Data Persistence (Refresh)"), the script enters data without saving and then refreshes.
  - This explicitly checks that unsaved profile changes are not silently persisted by the browser or frontend state.

III. Automation Test Results

1. How Results Were Captured

All profile test cases were executed using the unified automation script, which outputs results into CSV files for Chrome, Firefox, and Edge.

Each record stores: TC_ID, description, browser, PASS/FAIL, expected error, actual system toast, and details.

Please check the *attached csv files* for more details.

2. Summary of Results by Browser

Chrome

- Total: 34
- Pass: 1, Fail: 33
- Key issues observed:
    - Every valid update triggers “Resource not found” toast.
    - No client-side validation works.
    - All invalid inputs show backend error instead of specific validation errors.
    - Boundary tests fail consistently.
    - Only persistence (refresh) test passes reliably.

Firefox

- Total: 34
- Pass: 4, Fail: 31
- Behavior is *slightly different* from Chrome:
    - One valid update (Last Name) surprisingly passes.
    - Special characters in Name also pass.
    - Some postcode boundary tests behave differently compared to Chrome.
    - But overall: validation errors still do not display, and most updates fail with “Resource not found”.

Edge

- Total: 34
- Pass: 1, Fail: 33
- Mirrors Chrome more than Firefox.

3. Cross-Browser Conclusion

Unlike the *Add to Cart* feature (which failed in *identical* ways across all three browsers), the Edit Profile feature behaves *inconsistently*:

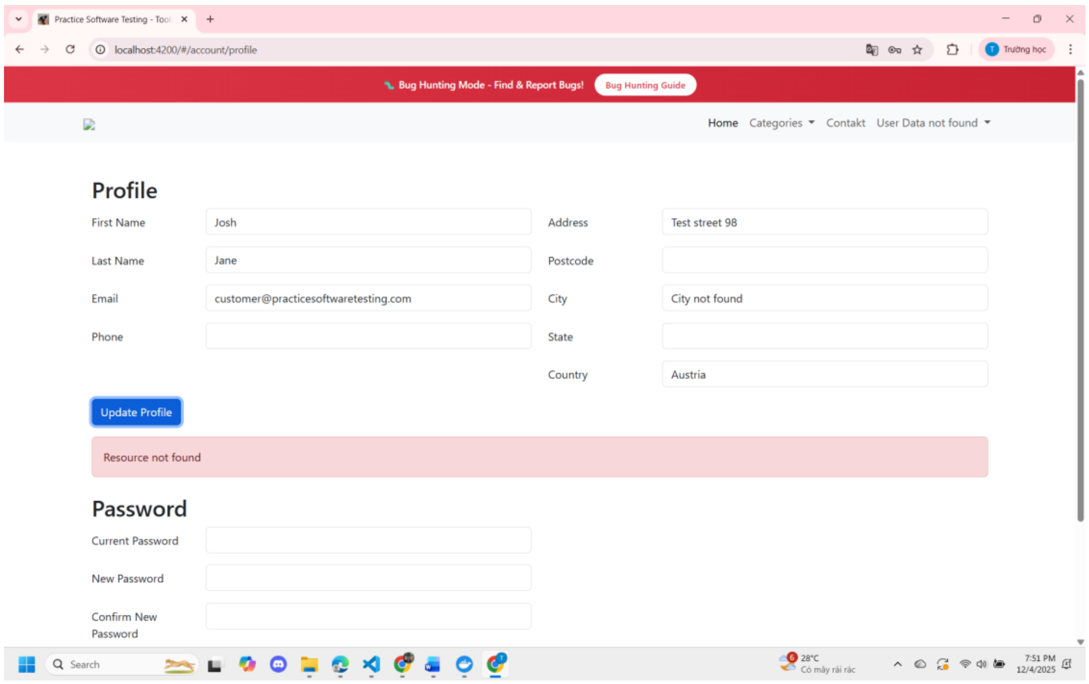
- Some tests pass only in Firefox but fail in Chrome and Edge.
- Boundary behavior varies unpredictably.
- Re-running the exact same automation script produces *different results across runs.*
- The backend appears unstable, returning “Resource not found” for almost any update.
- Front-end validation does not trigger at all, so invalid data is not blocked.

Overall, the Edit Profile feature is highly unstable, with inconsistent outcomes across browsers and across repeated executions.

IV. Discovered Bugs

- **Bug 6: "Resource Not Found".**
    - *Scenario:* Attempt to save *any* change to the profile (Valid or Invalid).

- *Actual:* A red toast appears: *"Resource not found"*.

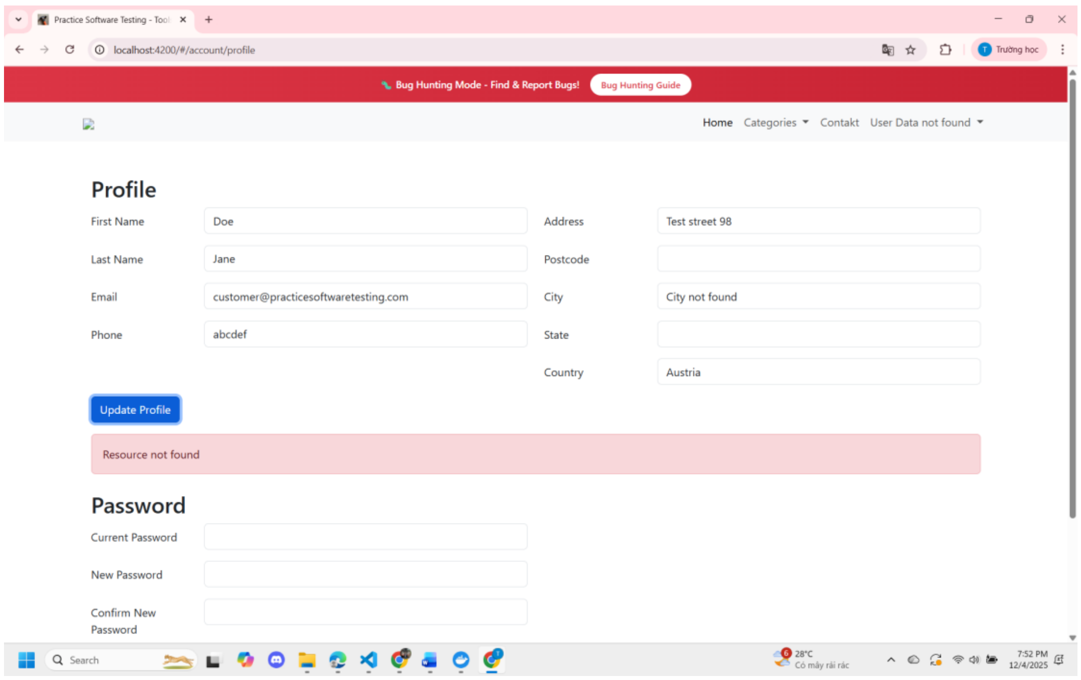

Profile

First Name: Josh | Address: Test street 98
Last Name: Jane | Postcode:
Email: customer@practicesoftwaretesting.com | City: City not found
Phone: | State:
Country: Austria

Resource not found

Figure 6: Profile update blocked by recurring “Resource not found” backend error.

- **Bug 7: Missing Validation.**
  - *Scenario:* Enter text into the "Phone" field.
  - *Expected:* Frontend error "Numbers only".
  - *Actual:* The backend 404 error ("Resource not found") overrides validation, or validation is missing entirely.


Profile

First Name: Doe | Address: Test street 98
Last Name: Jane | Postcode:
Email: customer@practicesoftwaretesting.com | City: City not found
Phone: abcdef | State:
Country: Austria

Resource not found

Figure 7: Save action blocked by confusing message.

V. Video Demo

Please check this link for the Edit Profile demo, the timestamps for each browser are listed in the description box.

C. Feature 3: Admin - Add/Edit Product

I. Workflow & Script Logic

The script test_product.py automates the Admin product management flow, covering adding new products, editing existing ones, and validating complex business rules (stock, price, name/description length, brand/category requirements, and rental/location flags).

1. **Admin Login & Access**
   - Launches browser and navigates to the login page.
   - Logs in as Admin using (admin@practicesoftwaretesting.com / welcome01).
   - Waits until the URL no longer contains /auth, confirming a logged-in session.
   - Navigates to #/admin/products and waits for the "Add Product" button ([data-test='product-add']).
2. **Interaction – Add Flow**
   - Clicks the Add Product button to open the product form.
   - Uses smart_input() for text/number fields:
     - Aggressively clears existing values using multiple strategies (click,. clear(), Ctrl+A → Delete, backspace loop, and JavaScript value reset) to avoid leftover input or browser autofill noise.
     - Types new values from the CSV (Name, Price, Stock, Description_Input).
   - Uses smart_select() to set dropdowns (Brand, Category, Image, CO_2 rating) via value and dispatches change events so the Angular form reacts correctly.
   - Uses smart_check() to tick checkboxes for is_location_offer and is_rental when requested by the CSV.
   - Scrolls to the bottom and clicks "Save" ([data-test='product-submit']).
3. **Interaction – Edit Flow**
   - Returns to the product list page (go_to_products_page()).
   - Locates the row whose product name matches Target_Product.
   - Within that row, finds the Edit link (either via data-test='product-edit' or the "Edit" text).
   - Clicks Edit, waits for the form, then selectively updates fields specified in the CSV row (Name, Price, Stock, Description_Input).
   - Clicks "Save" and proceeds to verification.
4. **Toast & Inline Error Capture**
   - After every Add/Edit action, the script calls capture_toasts() to collect any visible:
     - Success toasts (Product saved!, success classes).

- Error/validation toasts (required, must be, wrong, "Something went wrong", etc.).
- It also normalizes text and scans the page source to pick up inline validation messages (under form fields).
- A small retry loop (up to 3 attempts) gives late-rendering messages a chance to appear.

5. **Verification**
   - If Expected_Error is non-empty, the test is treated as a negative case:
     - The script normalizes Expected_Error and checks whether it appears in any toast or inline error block.
     - If found → PASS with "Expected error present".
     - If not found → FAIL with a summary of whatever toast text(s) were captured.
   - If Expected_Error is empty, the test is treated as a positive case:
     - If a success toast is seen and no pure error state remains → PASS.
     - If only error toasts are seen, or the product appears to be saved while still showing validation errors → FAIL or flagged behavior.
   - After each case, the script attempts to return to the product list, using a "Back" link or direct navigation, so each test starts from a clean page state.

II. Data Organization (product_data.csv)

The product admin tests use a large CSV that covers both equivalence partitioning and boundary value analysis across multiple fields.

1. Data Schema

The CSV file is organized into 15 control columns that guide the test behavior:

Column Name	Purpose	Example Data
TC_ID	Unique identifier for tracking results and mapping back to the Test Plan.	PROD-EP-01
Description	Short summary of what the test verifies.	Invalid (Stock Decimal)
Flow_Type	*The Logic Controller*. Tells the script which execution path to take (see below).	Add, Edit
Target_Product	When Flow_Type = Edit, specifies which existing product to open in the list.	Pliers, Hammer, Combination Pliers
Name	Product name to type into the Name field during Add/Edit.	New Ivory Bulldozer
Price	Product price input (including negative, zero, and boundary values for BVA).	56.14, 0.00, -0.01
Stock	Stock quantity input (including negative, zero, decimals, and large values).	39, 0, -1, 10.5

Description_Input	Product description content, used heavily in max length BVA cases.	This reliable Ivory Bulldozer… or long a...
Brand	Brand ID used to select from the Brand dropdown.	6
Category	Category ID used to select from the Category dropdown.	5
Image	Product image ID used by the Image dropdown, when applicable.	1
CO2	CO_2 rating ID selected via dropdown.	1, 2
Check_Location	Flag to indicate whether the "location offer" checkbox should be enabled.	True / empty
Check_Rental	Flag to indicate whether the "rental" checkbox should be enabled.	True / empty
Expected_Error	Expected validation message for negative tests; empty means the form is expected to save.	Name is required, Price must be 0.00 or greater

2. Execution Flow Logic (Flow_Type)

The script reads the Flow_Type column to determine the complexity of the test case:

- **Add**
  - Opens the Add Product form, populates all relevant fields, and attempts to save.
  - Used for EP tests (valid happy path, missing required fields) and for BVA on name length, description length, stock, and price boundaries.
- **Edit**
  - Locates an existing product row by Target_Product, opens the Edit form, and changes only the fields specified in the CSV row.
  - Used to confirm that the same validation rules apply on edit (clearing name, description, or setting invalid stock/price).

III. Automation Test Results

1. How Results Were Captured

All product test cases were executed using the unified automation script, which outputs results into CSV files for Chrome, Firefox, and Edge.

Each record stores: TC_ID, description, browser, flow type, PASS/FAIL, expected error, actual error seen, toast, and details.

Please check the *attached csv files* for more details.

2. Summary of Results by Browser

Chrome

- Total: 35
- Pass: 26, Fail: 9

- Most valid Add/Edit flows executed, but dual-toast behaviour was common (Stock is required for non-rental products + Product saved!).
- Failures mainly came from:
  - Missing validation for Brand, Category.
  - Incorrect handling of invalid Stock/Price.
  - Editing a product with invalid values still returning "Product saved!".

Firefox

- Total: 35
- Pass: 25, Fail: 10
- Behaves quite similar to Chrome, only Edit Name is different, showing inconsistent client-side validation.

Edge

- Total: 35
- Pass: 11, Fail: 21
- Edge's behaviour aligned with Chrome and Firefox on all major failure points.
- Additional failures in Edit form validation (Name/Description) due to backend accepting invalid data.

3. Cross-Browser Conclusion

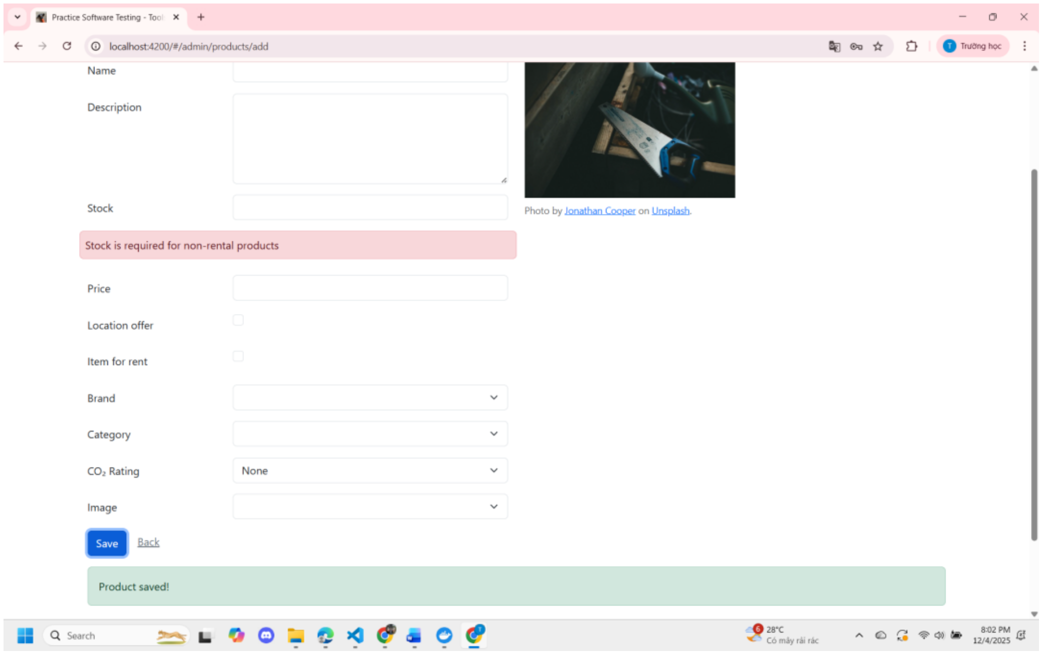
Worse than Edit Profile, this function is even more highly unstable and non-deterministic. The results vary not only between browsers but also between repeated runs of the same script:

- Missing validation messages.
- Conflicting UI messages.
- Backend accepting invalid data.
- Inconsistent Edit validation behavior.
- Cross-browser results that match in pattern but still show run-to-run flakiness.

Overall, the Add/Edit Product feature is unstable, lacks proper validation, and fails to provide reliable behavior across browsers or test executions.

IV. Discovered Bugs (New)

- **Bug 8: Confusing "Stock Required" Toast.**
  - *Scenario:* Create a valid product (with Stock).
  - *Actual:* Two toasts appear:
    1. (Green) "Product saved!" (Success).
    2. (Red) "Stock is required for non-rental products" (Error).


Practice Software Testing - Tool
localhost:4200/#/admin/products/add

Name	
Description	
Stock	

Stock is required for non-rental products

Price	
Location offer	
Item for rent	
Brand	
Category	
CO₂ Rating	None
Image	

Photo by Jonathan Cooper on Unsplash.

Save Back

Product saved!

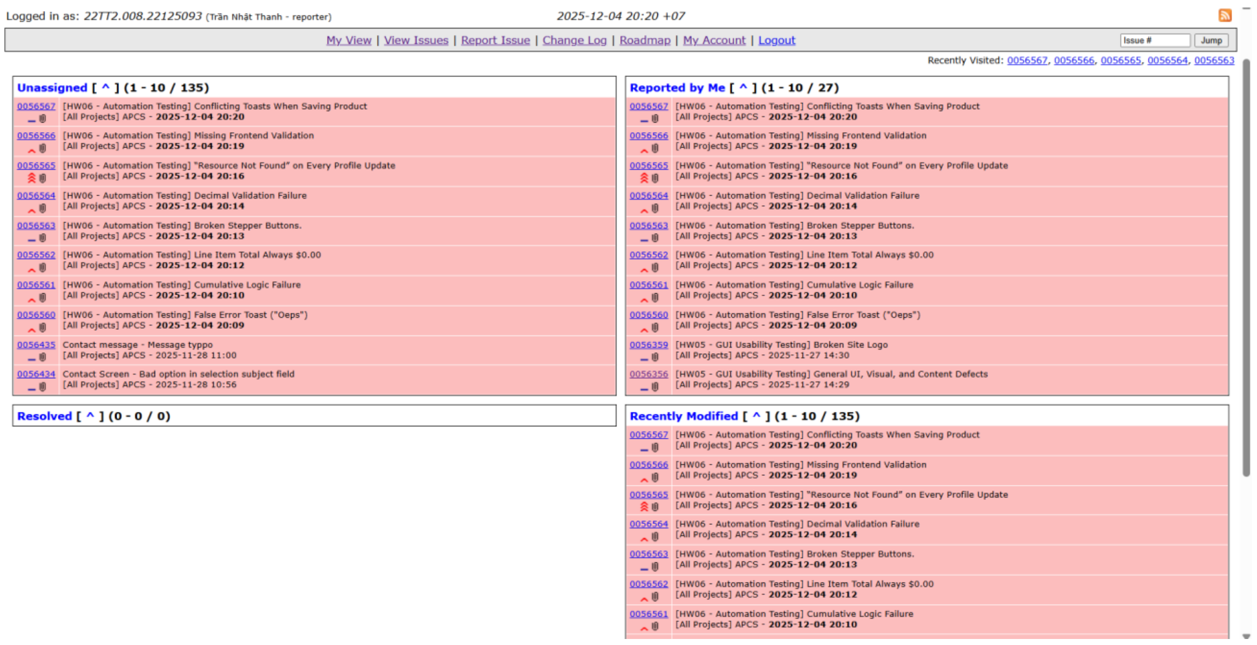
Figure 8: Product save operation triggers both success and error toasts simultaneously.

V. Video Demo

Please check this link for the Add/Edit Product demo, the timestamps for each browser are listed in the description box.

5. Bug Summary

The updated bugs on Mantis


Logged in as: *22TT2.008.22125093* (Trần Nhật Thanh - reporter) *2025-12-04 20:20 +07*

My View | View Issues | Report Issue | Change Log | Roadmap | My Account | Logout

Recently Visited: 0056567, 0056566, 0056565, 0056564, 0056563

Unassigned [^] (1 - 10 / 135)

0056567	[HW06 - Automation Testing] Conflicting Toasts When Saving Product [All Projects] APCS - **2025-12-04 20:20**
0056566	[HW06 - Automation Testing] Missing Frontend Validation [All Projects] APCS - **2025-12-04 20:19**
0056565	[HW06 - Automation Testing] "Resource Not Found" on Every Profile Update [All Projects] APCS - **2025-12-04 20:16**
0056564	[HW06 - Automation Testing] Decimal Validation Failure [All Projects] APCS - **2025-12-04 20:14**
0056563	[HW06 - Automation Testing] Broken Stepper Buttons. [All Projects] APCS - **2025-12-04 20:13**
0056562	[HW06 - Automation Testing] Line Item Total Always $0.00 [All Projects] APCS - **2025-12-04 20:12**
0056561	[HW06 - Automation Testing] Cumulative Logic Failure [All Projects] APCS - **2025-12-04 20:10**
0056560	[HW06 - Automation Testing] False Error Toast ("Oeps") [All Projects] APCS - **2025-12-04 20:09**
0056435	Contact message - Message typpo [All Projects] APCS - 2025-11-28 11:00
0056434	Contact Screen - Bad option in selection subject field [All Projects] APCS - 2025-11-28 10:56

Resolved [^] (0 - 0 / 0)

Reported by Me [^] (1 - 10 / 27)

0056567	[HW06 - Automation Testing] Conflicting Toasts When Saving Product [All Projects] APCS - **2025-12-04 20:20**
0056566	[HW06 - Automation Testing] Missing Frontend Validation [All Projects] APCS - **2025-12-04 20:19**
0056565	[HW06 - Automation Testing] "Resource Not Found" on Every Profile Update [All Projects] APCS - **2025-12-04 20:16**
0056564	[HW06 - Automation Testing] Decimal Validation Failure [All Projects] APCS - **2025-12-04 20:14**
0056563	[HW06 - Automation Testing] Broken Stepper Buttons. [All Projects] APCS - **2025-12-04 20:13**
0056562	[HW06 - Automation Testing] Line Item Total Always $0.00 [All Projects] APCS - **2025-12-04 20:12**
0056561	[HW06 - Automation Testing] Cumulative Logic Failure [All Projects] APCS - **2025-12-04 20:10**
0056560	[HW06 - Automation Testing] False Error Toast ("Oeps") [All Projects] APCS - **2025-12-04 20:09**
0056359	[HW05 - GUI Usability Testing] Broken Site Logo [All Projects] APCS - 2025-11-27 14:30
0056356	[HW05 - GUI Usability Testing] General UI, Visual, and Content Defects [All Projects] APCS - 2025-11-27 14:29

Recently Modified [^] (1 - 10 / 135)

0056567	[HW06 - Automation Testing] Conflicting Toasts When Saving Product [All Projects] APCS - **2025-12-04 20:20**
0056566	[HW06 - Automation Testing] Missing Frontend Validation [All Projects] APCS - **2025-12-04 20:19**
0056565	[HW06 - Automation Testing] "Resource Not Found" on Every Profile Update [All Projects] APCS - **2025-12-04 20:16**
0056564	[HW06 - Automation Testing] Decimal Validation Failure [All Projects] APCS - **2025-12-04 20:14**
0056563	[HW06 - Automation Testing] Broken Stepper Buttons. [All Projects] APCS - **2025-12-04 20:13**
0056562	[HW06 - Automation Testing] Line Item Total Always $0.00 [All Projects] APCS - **2025-12-04 20:12**
0056561	[HW06 - Automation Testing] Cumulative Logic Failure [All Projects] APCS - **2025-12-04 20:10**

Self – Assessment

Criteria	Outcomes	Grade	Self-Assessed Grade
1	**Guest: Add to Cart**	**40**	**40**
	1.1 Report + Bug	15	15
	1.2 Test cases	5	5
	1.3 Script files	5	5
	1.4 Data files	5	5
	1.5 Videos	10	10
2	**User: Edit Profile**	**30**	**30**
	2.1 Report + Bug	10	10
	2.2 Test cases	5	5
	2.3 Script files	5	5
	2.4 Data files	5	5
	2.5 Videos	5	5
3	**Admin: Add/Edit Product**	**30**	**30**
	3.1 Report + Bug	10	10
	3.2 Test cases	5	5
	3.3 Script files	5	5
	3.4 Data files	5	5
	3.5 Videos	5	5
	Total	**100**	**100**