University of Science – Vietnam National University
Faculty of Information Technology

CS423 – Software Testing

HW08 Report

API Testing

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

Ho Chi Minh City, December 16th, 2025

Contents

1. Overview 3
2. Test Environment & Testing Strategy 3
A. Test Environment 3
B. Testing Strategy 3
3. Detailed Analysis 4
A. API Module: User Registration (POST) 4
I. Configuration & Setup 4
II. Automation Logic 4
III. Test Results 4
B. API Module: Product Search (GET) 6
I. Configuration & Setup 6
II. Automation Logic 6
III. Test Results 6
C. API Module: Invoice Status Update (PUT) 8
I. Configuration & Setup 8
II. Automation Logic 8
III. Test Results 8
4. Appendix (Screenshots of Test Cases in Postman) 10
A. POST /users/register 10
B. GET /products 14
C. PUT /invoices/{invoiceId}/status 19
Self – Assessment 24

1. Overview

This report documents the results of the automated API testing performed on the Toolshop (with bugs version) backend. The scope covered three critical functional modules:

1. **User Registration:** Validating data integrity, security, and error handling.
2. **Product Search:** Verifying search algorithms and filtering logic.
3. **Invoice Status Update:** Testing business logic transitions, security permissions, and error handling.

Overall Results:

- **Total Test Cases:** 107
- **Passed:** 73 (68%)
- **Failed:** 34 (32%)

2. Test Environment & Testing Strategy

A. Test Environment

- **API Base URL:** http://localhost:8091
- **Testing Tool:** Postman (Collection Runner & scripting engine)
- **Authentication:** Admin Bearer Token, automatically generated via the login API and stored as a collection variable.

B. Testing Strategy

- **Data-Driven API Testing:**
  Postman's Collection Runner was used to execute multiple test cases for the same endpoint. Test inputs, expected HTTP status codes, and validation conditions were supplied through external CSV files, allowing each iteration to represent an independent test case.
- **State Transition & State Management:**
  For APIs involving state changes (such as invoice status updates), a pre-request scripting mechanism was implemented to control the initial state before execution. This approach ensured that each test case was executed from a known and consistent state, preventing dependency between test runs and enabling proper state transition testing.
- **Dynamic Response Validation:**
  Postman test scripts were used to validate both HTTP status codes and response body content. For list-based responses (such as product search and filtering), scripts dynamically parsed returned data to verify business logic, including sorting order, filtering conditions, and value ranges, rather than relying on static assertions.

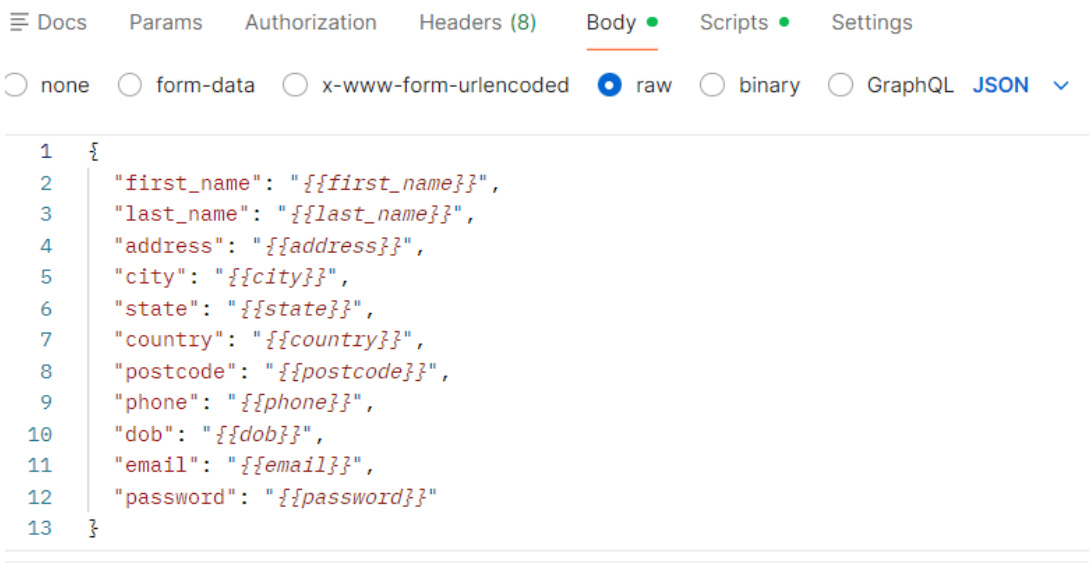
3. Detailed Analysis

A. API Module: User Registration (POST)

Objective: Verify the system's ability to create new users while enforcing strict input validation, data type constraints, and security policies.

I. Configuration & Setup

- **Endpoint:** http://localhost:8091/users/register
- **Method:** POST
- **Data Strategy:** Data-Driven Testing using register_data.csv.
- **Payload Configuration:** The request body uses dynamic variables mapped to CSV column headers:



II. Automation Logic

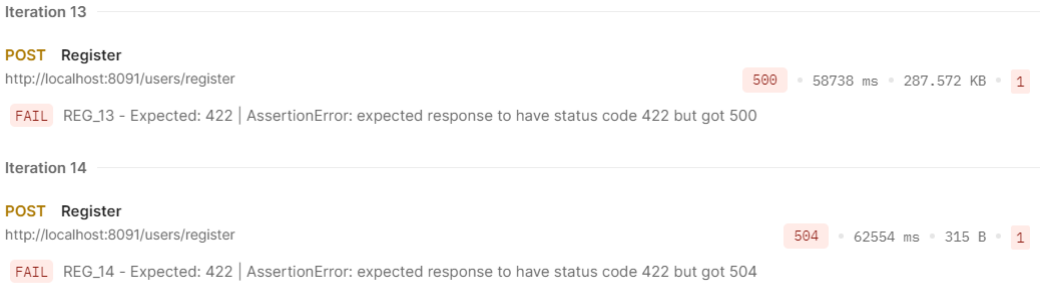
Since this is a creation endpoint, no pre-request setup was needed. Validation logic was handled in the *Post-response* script:

Dynamic Status Check: Instead of hardcoding 200 or 422, the script reads the expected_status column from the CSV. This allows one script to handle both Positive (201) and Negative (422) test cases.

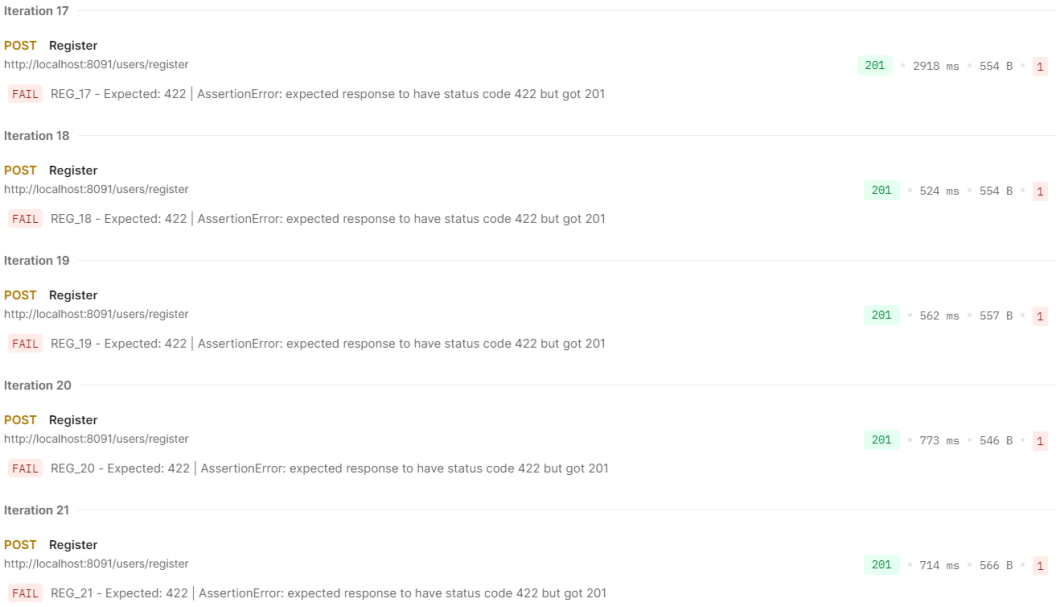
III. Test Results

- **Total Cases:** 35
  - Passed: 26
  - Failed: 9
- **Bugs Found:** *(Please view the Test Cases file for more detailed)*

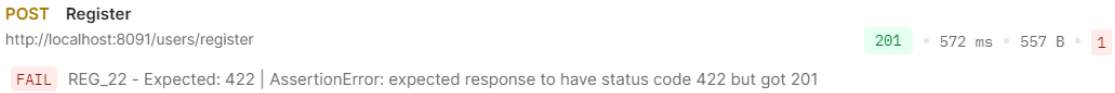
- **Bug-01 (Server Error on Invalid Email Format):**
  Registration requests with malformed email formats (missing @, missing domain) resulted in *HTTP 500 Internal Server Error* or *HTTP 504 Gateway Timeout* instead of returning a validation error (422). This indicates insufficient input validation and unhandled exceptions in the backend.



- **Bug-02 (Incomplete Input Validation for Password and Phone Fields):**
  The system accepted invalid password and phone number values, including passwords that violated complexity rules (missing uppercase letters or numeric characters) and phone numbers with non-numeric characters or invalid lengths. This shows that format and boundary validations are not consistently enforced.



- **Bug-03 (Weak Validation of Postcode Format):**
  Invalid postcode values containing alphabetical characters were accepted by the system, although length constraints were correctly applied. This reflects inconsistent validation rules across different user input fields.

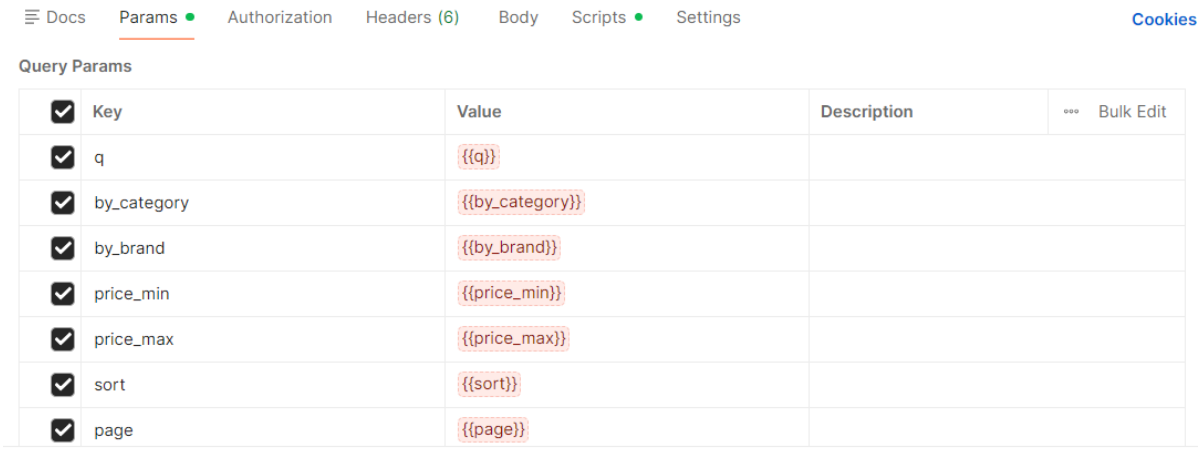


B. API Module: Product Search (GET)

Objective: Validate the search engine's filtering and sorting logic to ensure accurate data retrieval.

I. Configuration & Setup

- **Endpoint:** http://localhost:8091/products
- **Method:** GET
- **Data Strategy:** Data-Driven Testing using product_data.csv.
- **Query Parameters:**



II. Automation Logic

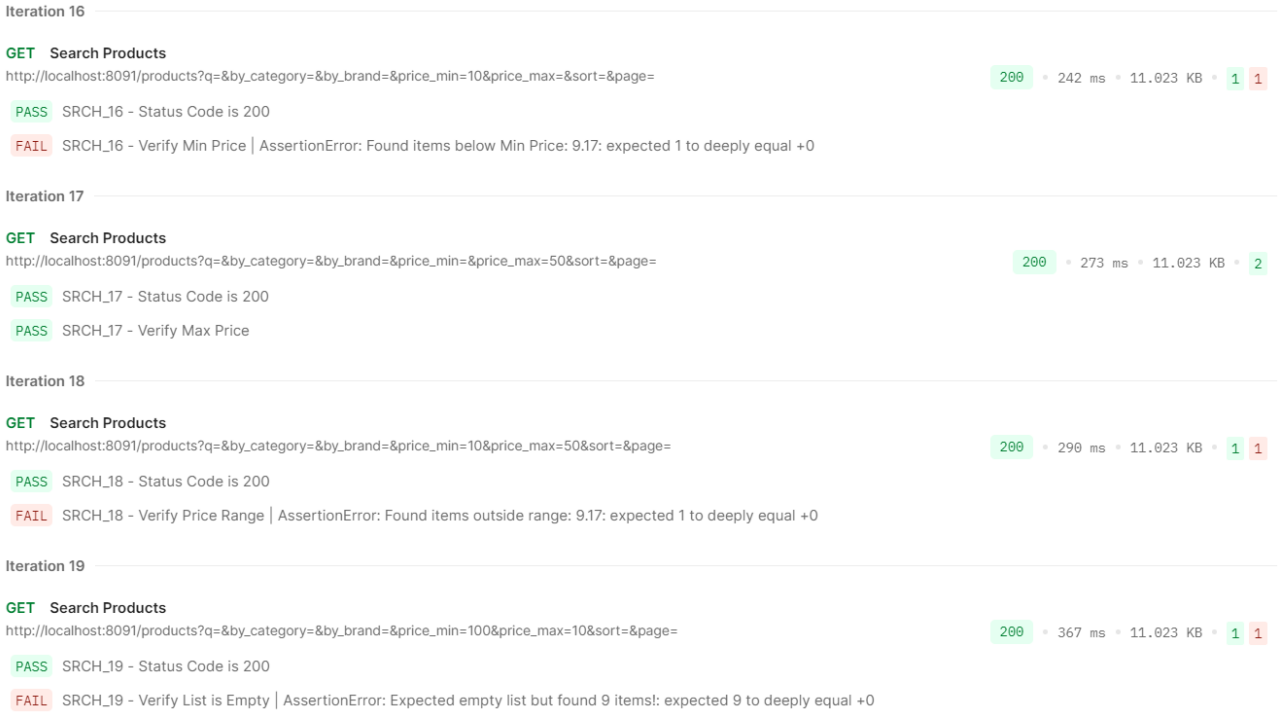
A standard "Status Code Check" is insufficient for search tests because a *200 OK* doesn't prove the *right* products were returned. I implemented a *Smart Logic Script* that:

1. **Parses the Response List:** Detects if the JSON returns data[] or items[].
2. **Logical Verification:**
    - **For Keywords:** It converts item names to lowercase to verify they contain the search query.
    - **For Sorting:** It extracts prices/names/co2_rating, sorts them in JavaScript, and compares them to the API response order.
    - **For Filtering:** It iterates through every returned item to verify price range, categories and brands.

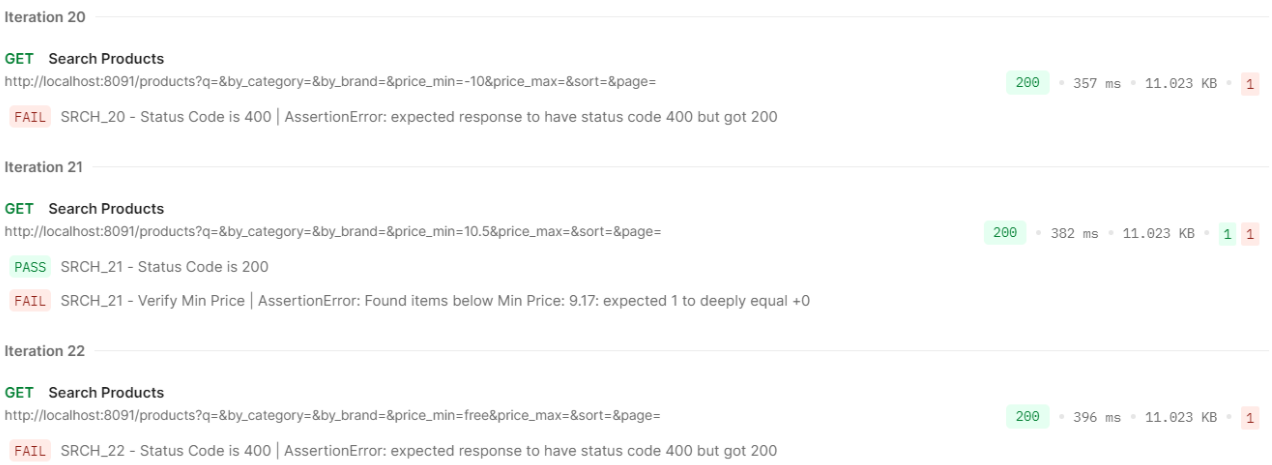
III. Test Results

- **Total Cases:** 38
    - Passed: 29
    - Failed: 9
- **Bugs Found:** *(Please view the Test Cases file for more detailed)*
    - **BUG-04 (Broken Price Filter Validation & Range Logic):**
      Price-related parameters (price_min, price_max) are not enforced correctly. The API returned products below price_min, above price_max, and outside the

expected range. In addition, invalid price inputs (negative values, non-numeric values, and cases where price_min > price_max) were not rejected and sometimes returned normal results. This indicates missing domain validation and incorrect filtering logic for price constraints.



- **BUG-05 (Weak Input Validation and Error Handling for Query Parameters):**
  Invalid or unsupported query parameters, including malformed filter values (non-numeric category or brand IDs), negative pagination values, and unsupported sort fields, were not handled consistently. In some cases, the API silently ignored invalid inputs, while in others it triggered server-side errors (HTTP 500) instead of returning appropriate client errors. This reflects inadequate input validation and unstable error-handling behavior.



C. API Module: Invoice Status Update (PUT)

Objective: Verify business logic for order state transitions, ensuring data integrity and correct error handling.

I. Configuration & Setup

- **Endpoint:** http://localhost:8091/invoices/{{invoice_id}}/status
- **Method:** PUT
- **Authorization:** Bearer Token (Admin).
- **Data Strategy:** CSV input (invoice_data.csv) controlling start_status (Pre-condition) and status (Target Status).

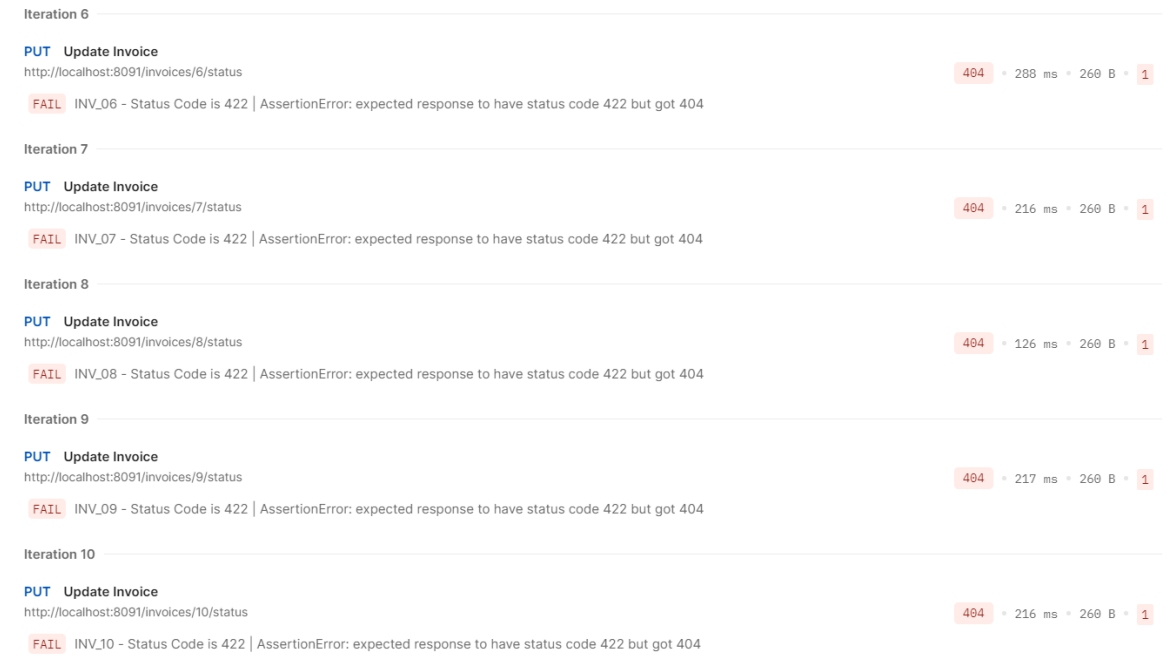
II. Automation Logic

To ensure tests were independent and repeatable (Idempotent), I developed a specific strategy:

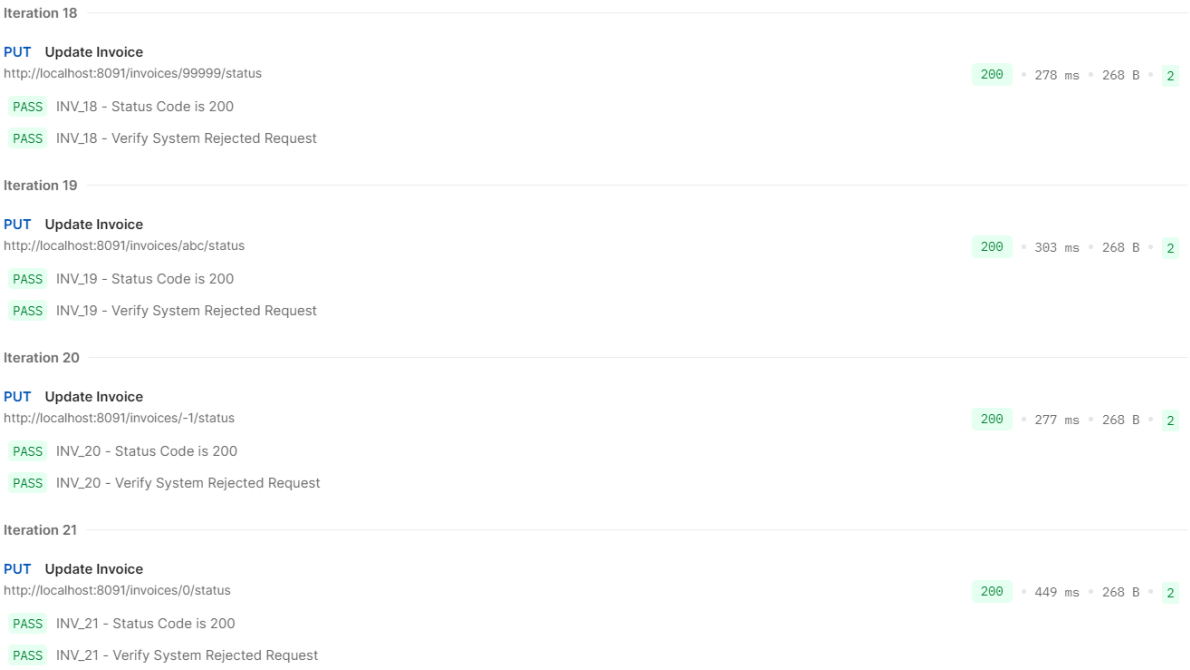
- **Pre-request Script:** Before the test runs, this script sends a "Secret" API request to force the invoice into the correct start_status.
  Why? You cannot test "Transition from ON_HOLD to SHIPPED" if the invoice is currently "COMPLETED". This script guarantees the valid starting state.
- **Post-response Script:** The API response body for updates is empty ({success: true}). This is insufficient for testing.
  Solution: The test script triggers a *second API call* (GET /invoices/:id) to fetch the database state and verify the status actually persisted.

III. Test Results

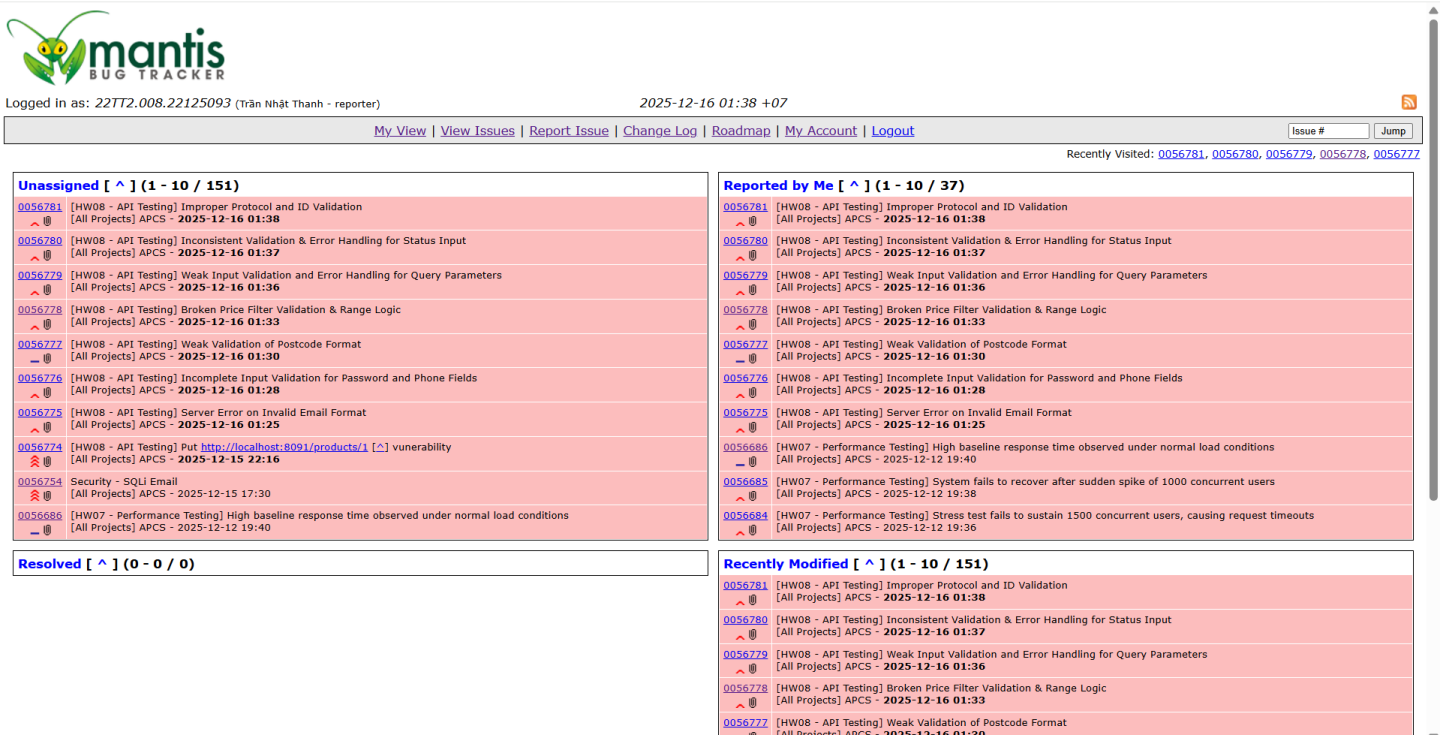
- **Total Cases:** 34
  - Passed: 18
  - Failed: 16
- **Bugs Found:** *(Please view the Test Cases file for more detailed)*
  - **BUG-06 (Inconsistent Validation & Error Handling for Status Input):**
    The API does not consistently validate the status field. Invalid status values (wrong case, random strings, empty/null values, numeric types, overly long strings, malformed JSON, and special characters) frequently resulted in *404 Not Found* responses or were silently accepted instead of returning appropriate validation errors (422 or 400). This indicates weak input validation and incorrect error mapping for status-related inputs.



- **BUG-07 (Improper Protocol and ID Validation):**
  Invalid invoice identifiers (non-existent IDs, malformed IDs, negative or zero values) and incorrect request formats (invalid content type) were not handled according to RESTful standards. The API often returned *HTTP 200* or allowed requests to proceed instead of returning proper client errors (400, 404, or 415). Although the system generally prevented unauthorized updates, the protocol-level error handling is inconsistent and misleading for API consumers.

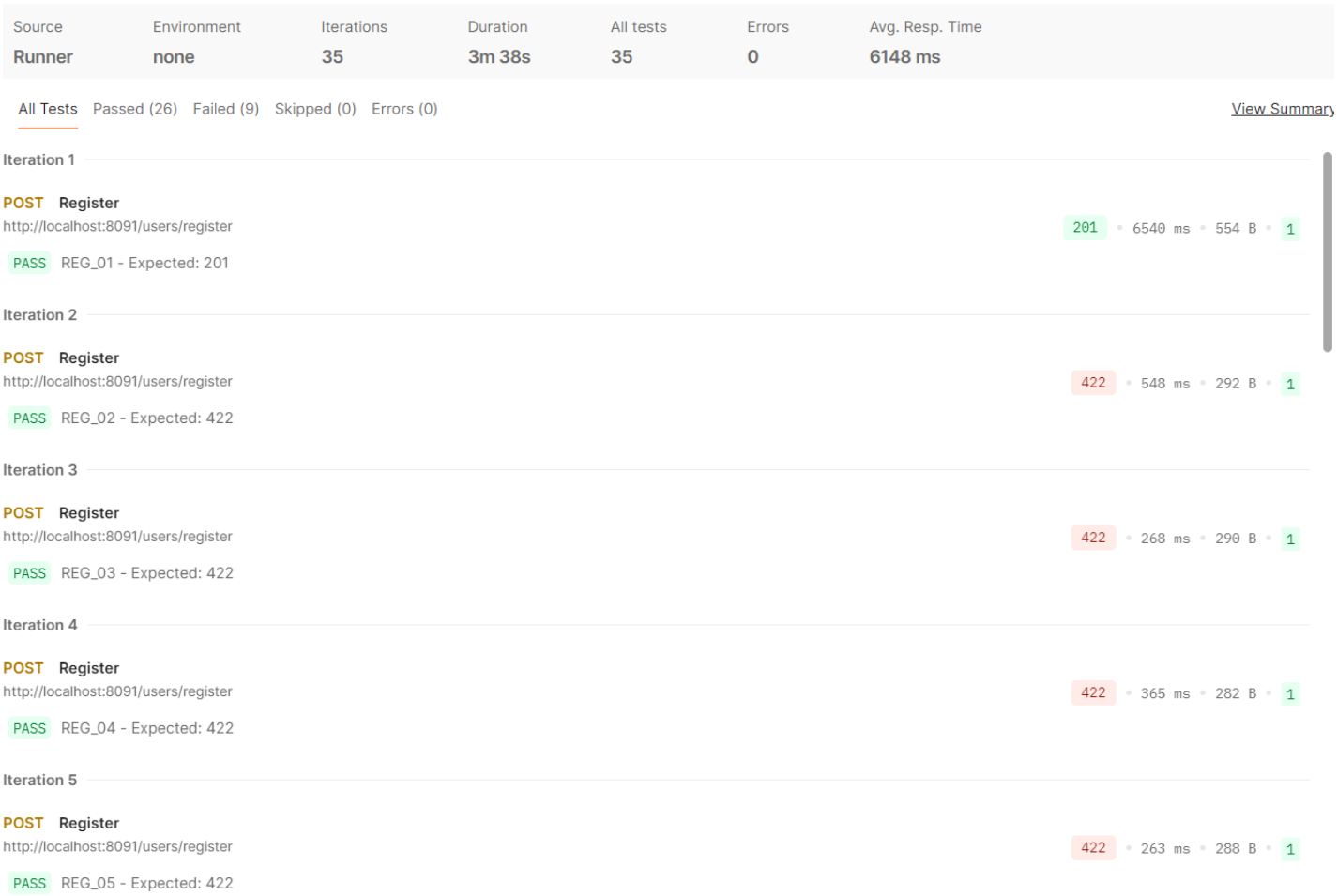


The updated bugs on Mantis



4. Appendix (Screenshots of Test Cases in Postman)

A. POST /users/register



Iteration 6

POST Register
http://localhost:8091/users/register
422 • 208 ms • 286 B • 1

PASS REG_06 - Expected: 422

Iteration 7

POST Register
http://localhost:8091/users/register
422 • 186 ms • 280 B • 1

PASS REG_07 - Expected: 422

Iteration 8

POST Register
http://localhost:8091/users/register
422 • 375 ms • 287 B • 1

PASS REG_08 - Expected: 422

Iteration 9

POST Register
http://localhost:8091/users/register
422 • 252 ms • 286 B • 1

PASS REG_09 - Expected: 422

Iteration 10

POST Register
http://localhost:8091/users/register
422 • 231 ms • 293 B • 1

PASS REG_10 - Expected: 422

Iteration 11

POST Register
http://localhost:8091/users/register
422 • 256 ms • 287 B • 1

PASS REG_11 - Expected: 422

Iteration 12

POST Register
http://localhost:8091/users/register
422 • 256 ms • 320 B • 1

PASS REG_12 - Expected: 422

Iteration 13

POST Register
http://localhost:8091/users/register
500 • 58738 ms • 287.572 KB • 1

FAIL REG_13 - Expected: 422 | AssertionError: expected response to have status code 422 but got 500

Iteration 14

POST Register
http://localhost:8091/users/register
504 • 62554 ms • 315 B • 1

FAIL REG_14 - Expected: 422 | AssertionError: expected response to have status code 422 but got 504

Iteration 15

POST Register
http://localhost:8091/users/register
422 • 6496 ms • 306 B • 1

PASS REG_15 - Expected: 422

Iteration 16

POST Register
http://localhost:8091/users/register
422 • 479 ms • 315 B • 1

PASS REG_16 - Expected: 422

Iteration 17

POST Register
http://localhost:8091/users/register
201 • 2918 ms • 554 B • 1

FAIL REG_17 - Expected: 422 | AssertionError: expected response to have status code 422 but got 201

Iteration 18

POST Register
http://localhost:8091/users/register
201 • 524 ms • 554 B • 1

FAIL REG_18 - Expected: 422 | AssertionError: expected response to have status code 422 but got 201

Iteration 19

POST Register
http://localhost:8091/users/register
201 • 562 ms • 557 B • 1

FAIL REG_19 - Expected: 422 | AssertionError: expected response to have status code 422 but got 201

Iteration 20

POST Register
http://localhost:8091/users/register
201 • 773 ms • 546 B • 1

FAIL REG_20 - Expected: 422 | AssertionError: expected response to have status code 422 but got 201

Iteration 21

POST Register
http://localhost:8091/users/register
201 • 714 ms • 566 B • 1

FAIL REG_21 - Expected: 422 | AssertionError: expected response to have status code 422 but got 201

Iteration 22

POST Register
http://localhost:8091/users/register
201 • 572 ms • 557 B • 1

FAIL REG_22 - Expected: 422 | AssertionError: expected response to have status code 422 but got 201

Iteration 23

POST Register
http://localhost:8091/users/register
422 • 183 ms • 315 B • 1

PASS REG_23 - Expected: 422

Iteration 24

POST Register
http://localhost:8091/users/register
422 • 271 ms • 282 B • 1

PASS REG_24 - Expected: 422

Iteration 25

POST Register
http://localhost:8091/users/register
422 • 291 ms • 294 B • 1

PASS REG_25 - Expected: 422

Iteration 26

POST Register
http://localhost:8091/users/register
201 • 438 ms • 591 B • 1

PASS REG_26 - Expected: 201

Iteration 27

POST Register
http://localhost:8091/users/register
422 • 305 ms • 319 B • 1

PASS REG_27 - Expected: 422

Iteration 28

POST Register
http://localhost:8091/users/register
201 • 537 ms • 572 B • 1

PASS REG_28 - Expected: 201

Iteration 29

POST Register
http://localhost:8091/users/register
422 • 185 ms • 317 B • 1

PASS REG_29 - Expected: 422

Iteration 30

POST Register
http://localhost:8091/users/register
201 • 569 ms • 564 B • 1

PASS REG_30 - Expected: 201

Iteration 31

POST Register
http://localhost:8091/users/register
422 • 308 ms • 313 B • 1

PASS REG_31 - Expected: 422

Iteration 32

POST Register
http://localhost:8091/users/register
201 • 566 ms • 563 B • 1

PASS REG_32 - Expected: 201

Iteration 33

POST Register
http://localhost:8091/users/register
422 • 261 ms • 784 B • 1

PASS REG_33 - Expected: 422

Iteration 34

POST Register
http://localhost:8091/users/register
500 • 58181 ms • 287.567 KB • 1

FAIL REG_34 - Expected: 422 | AssertionError: expected response to have status code 422 but got 500

Iteration 35

POST Register
http://localhost:8091/users/register
422 • 8998 ms • 298 B • 1

PASS REG_35 - Expected: 422

B. GET /products

Source	Environment	Iterations	Duration	All tests	Errors	Avg. Resp. Time
Runner	none	38	25s 815ms	72	0	587 ms

All Tests Passed (63) Failed (9) Skipped (0) Errors (0) View Summary

Iteration 1

GET Search Products
http://localhost:8091/products?q=Hammer&by_category=&by_brand=&price_min=&price_max=&sort=&page= 200 • 10990 ms • 9.213 KB • 2

PASS SRCH_01 - Status Code is 200

PASS SRCH_01 - Verify Results Match Keyword

Iteration 2

GET Search Products
http://localhost:8091/products?q=Ham&by_category=&by_brand=&price_min=&price_max=&sort=&page= 200 • 625 ms • 9.213 KB • 2

PASS SRCH_02 - Status Code is 200

PASS SRCH_02 - Verify Results Match Keyword

Iteration 3

GET Search Products
http://localhost:8091/products?q=Supercalifragilistic&by_category=&by_brand=&price_min=&price_max=&sort=&page= 200 • 453 ms • 338 B • 2

PASS SRCH_03 - Status Code is 200

PASS SRCH_03 - Verify List is Empty

Iteration 4

GET Search Products
http://localhost:8091/products?q=HAMMER&by_category=&by_brand=&price_min=&price_max=&sort=&page= 200 • 349 ms • 9.213 KB • 2

PASS SRCH_04 - Status Code is 200

PASS SRCH_04 - Verify Results Match Keyword

Iteration 5

GET Search Products
http://localhost:8091/products?q=@%23%&by_category=&by_brand=&price_min=&price_max=&sort=&page= 200 • 180 ms • 338 B • 2

PASS SRCH_05 - Status Code is 200

PASS SRCH_05 - Verify List is Empty

Iteration 6

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=&price_max=&sort=&page= 200 • 241 ms • 11.023 KB • 2

PASS SRCH_06 - Status Code is 200

PASS SRCH_06 - Verify Returns All/Multiple Products

Iteration 7

GET Search Products
http://localhost:8091/products?q=' OR 1=1 --&by_category=&by_brand=&price_min=&price_max=&sort=&page= 200 • 275 ms • 338 B • 2

PASS SRCH_07 - Status Code is 200

PASS SRCH_07 - Verify List is Empty

Iteration 8

GET Search Products
http://localhost:8091/products?q=<script>alert(1)</script>&by_category=&by_brand=&price_min=&price_max=&sort=&page= 200 • 245 ms • 338 B • 2

PASS SRCH_08 - Status Code is 200

PASS SRCH_08 - Verify List is Empty

Iteration 9

GET **Search Products**
http://localhost:8091/products?q=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa... 200 • 204 ms • 338 B • 2

PASS SRCH_09 - Status Code is 200

PASS SRCH_09 - Verify List is Empty

Iteration 10

GET **Search Products**
http://localhost:8091/products?q=&by_category=3&by_brand=&price_min=&price_max=&sort=&page= 200 • 231 ms • 9.213 KB • 2

PASS SRCH_10 - Status Code is 200

PASS SRCH_10 - Verify Category ID

Iteration 11

GET **Search Products**
http://localhost:8091/products?q=&by_category=99999&by_brand=&price_min=&price_max=&sort=&page= 200 • 247 ms • 338 B • 2

PASS SRCH_11 - Status Code is 200

PASS SRCH_11 - Verify List is Empty

Iteration 12

GET **Search Products**
http://localhost:8091/products?q=&by_category=abc&by_brand=&price_min=&price_max=&sort=&page= 200 • 301 ms • 338 B • 2

PASS SRCH_12 - Status Code is 200

PASS SRCH_12 - Verify List is Empty

Iteration 13

GET **Search Products**
http://localhost:8091/products?q=&by_category=&by_brand=1&price_min=&price_max=&sort=&page= 200 • 174 ms • 11.704 KB • 2

PASS SRCH_13 - Status Code is 200

PASS SRCH_13 - Verify Brand ID

Iteration 14

GET **Search Products**
http://localhost:8091/products?q=&by_category=&by_brand=99999&price_min=&price_max=&sort=&page= 200 • 173 ms • 338 B • 2

PASS SRCH_14 - Status Code is 200

PASS SRCH_14 - Verify List is Empty

Iteration 15

GET **Search Products**
http://localhost:8091/products?q=&by_category=&by_brand=abc&price_min=&price_max=&sort=&page= 200 • 226 ms • 338 B • 2

PASS SRCH_15 - Status Code is 200

PASS SRCH_15 - Verify List is Empty

Iteration 16

GET **Search Products**
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=10&price_max=&sort=&page= 200 • 242 ms • 11.023 KB • 1 1

PASS SRCH_16 - Status Code is 200

FAIL SRCH_16 - Verify Min Price | AssertionError: Found items below Min Price: 9.17: expected 1 to deeply equal +0

Iteration 17

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=&price_max=50&sort=&page= 200 • 273 ms • 11.023 KB • 2

PASS SRCH_17 - Status Code is 200

PASS SRCH_17 - Verify Max Price

Iteration 18

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=10&price_max=50&sort=&page= 200 • 290 ms • 11.023 KB • 1 1

PASS SRCH_18 - Status Code is 200

FAIL SRCH_18 - Verify Price Range | AssertionError: Found items outside range: 9.17: expected 1 to deeply equal +0

Iteration 19

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=100&price_max=10&sort=&page= 200 • 367 ms • 11.023 KB • 1 1

PASS SRCH_19 - Status Code is 200

FAIL SRCH_19 - Verify List is Empty | AssertionError: Expected empty list but found 9 items!: expected 9 to deeply equal +0

Iteration 20

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=-10&price_max=&sort=&page= 200 • 357 ms • 11.023 KB • 1

FAIL SRCH_20 - Status Code is 400 | AssertionError: expected response to have status code 400 but got 200

Iteration 21

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=10.5&price_max=&sort=&page= 200 • 382 ms • 11.023 KB • 1 1

PASS SRCH_21 - Status Code is 200

FAIL SRCH_21 - Verify Min Price | AssertionError: Found items below Min Price: 9.17: expected 1 to deeply equal +0

Iteration 22

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=free&price_max=&sort=&page= 200 • 396 ms • 11.023 KB • 1

FAIL SRCH_22 - Status Code is 400 | AssertionError: expected response to have status code 400 but got 200

Iteration 23

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=&price_max=&sort=name,asc&page= 200 • 365 ms • 11.035 KB • 2

PASS SRCH_23 - Status Code is 200

PASS SRCH_23 - Verify name is asc

Iteration 24

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=&price_max=&sort=name,desc&page= 200 • 290 ms • 11.127 KB • 2

PASS SRCH_24 - Status Code is 200

PASS SRCH_24 - Verify name is desc

Iteration 25

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=&price_max=&sort=price,desc&page= 200 • 381 ms • 9.844 KB • 2

PASS SRCH_25 - Status Code is 200

PASS SRCH_25 - Verify price is desc

Iteration 26

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=&price_max=&sort=price,asc&page= 200 • 275 ms • 11.614 KB • 2

PASS SRCH_26 - Status Code is 200

PASS SRCH_26 - Verify price is asc

Iteration 27

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=&price_max=&sort=co2_rating,asc&page= 200 • 237 ms • 11.274 KB • 2

PASS SRCH_27 - Status Code is 200

PASS SRCH_27 - Verify co2_rating is asc

Iteration 28

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=&price_max=&sort=co2_rating,desc&page= 200 • 179 ms • 10.593 KB • 2

PASS SRCH_28 - Status Code is 200

PASS SRCH_28 - Verify co2_rating is desc

Iteration 29

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=&price_max=&sort=xyz,asc&page= 500 • 908 ms • 274 B • 1

FAIL SRCH_29 - Status Code is 400 | AssertionError: expected response to have status code 400 but got 500

Iteration 30

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=&price_max=&sort=&page=1 200 • 251 ms • 11.023 KB • 2

PASS SRCH_30 - Status Code is 200

PASS SRCH_30 - Verify Current Page is 1

Iteration 31

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=&price_max=&sort=&page=1000 200 • 189 ms • 342 B • 2

PASS SRCH_31 - Status Code is 200

PASS SRCH_31 - Verify List is Empty

Iteration 32

GET Search Products
http://localhost:8091/products?q=&by_category=&by_brand=&price_min=&price_max=&sort=&page=-1 200 • 292 ms • 11.023 KB • 1

FAIL SRCH_32 - Status Code is 400 | AssertionError: expected response to have status code 400 but got 200

Iteration 33

GET Search Products
http://localhost:8091/products?q=&by_category=3&by_brand=1&price_min=&price_max=&sort=&page=
200 • 161 ms • 9.213 KB • 2

PASS SRCH_33 - Status Code is 200

PASS SRCH_33 - Verify Brand ID

Iteration 34

GET Search Products
http://localhost:8091/products?q=Hammer&by_category=&by_brand=&price_min=&price_max=20&sort=&page=
200 • 292 ms • 9.213 KB • 1 1

PASS SRCH_34 - Status Code is 200

FAIL SRCH_34 - Verify Max Price | AssertionError: Found items above Max Price: 20.14: expected 1 to deeply equal +0

Iteration 35

GET Search Products
http://localhost:8091/products?q=Hammer&by_category=&by_brand=&price_min=&price_max=&sort=price,desc&page=
200 • 280 ms • 9.213 KB • 2

PASS SRCH_35 - Status Code is 200

PASS SRCH_35 - Verify price is desc

Iteration 36

GET Search Products
http://localhost:8091/products?q=a&by_category=3&by_brand=&price_min=10&price_max=&sort=name,asc&page=
200 • 323 ms • 9.213 KB • 2

PASS SRCH_36 - Status Code is 200

PASS SRCH_36 - Verify name is asc

Iteration 37

GET Search Products
http://localhost:8091/products?q=&by_category=3&by_brand=99&price_min=&price_max=&sort=&page=
200 • 280 ms • 338 B • 2

PASS SRCH_37 - Status Code is 200

PASS SRCH_37 - Verify List is Empty

Iteration 38

GET Search Products
http://localhost:8091/products?q=Claw Hammer&by_category=&by_brand=&price_min=&price_max=&sort=&page=
200 • 370 ms • 4.142 KB • 2

PASS SRCH_38 - Status Code is 200

PASS SRCH_38 - Verify Results Match Keyword

C. PUT /invoices/{invoiceId}/status

Source	Environment	Iterations	Duration	All tests	Errors	Avg. Resp. Time
Runner	none	34	1m 13s	63	0	615 ms

All Tests Passed (47) Failed (16) Skipped (0) Errors (0) View Summary

Iteration 1

PUT Update Invoice
http://localhost:8091/invoices/1/status
200 • 364 ms • 267 B • 3

PASS INV_01 - Status Code is 200
PASS INV_01 - API Response says Success
PASS INV_01 - Verify Persisted Status Change

Iteration 2

PUT Update Invoice
http://localhost:8091/invoices/2/status
200 • 208 ms • 267 B • 3

PASS INV_02 - Status Code is 200
PASS INV_02 - API Response says Success
PASS INV_02 - Verify Persisted Status Change

Iteration 3

PUT Update Invoice
http://localhost:8091/invoices/3/status
200 • 326 ms • 267 B • 3

PASS INV_03 - Status Code is 200
PASS INV_03 - API Response says Success
PASS INV_03 - Verify Persisted Status Change

Iteration 4

PUT Update Invoice
http://localhost:8091/invoices/4/status
200 • 257 ms • 267 B • 3

PASS INV_04 - Status Code is 200
PASS INV_04 - API Response says Success
PASS INV_04 - Verify Persisted Status Change

Iteration 5

PUT Update Invoice
http://localhost:8091/invoices/5/status
200 • 304 ms • 267 B • 3

PASS INV_05 - Status Code is 200
PASS INV_05 - API Response says Success
PASS INV_05 - Verify Persisted Status Change

Iteration 6

PUT Update Invoice
http://localhost:8091/invoices/6/status
404 • 288 ms • 260 B • 1

FAIL INV_06 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 404

Iteration 7

PUT Update Invoice
http://localhost:8091/invoices/7/status
404 • 216 ms • 260 B • 1

FAIL INV_07 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 404

Iteration 8

PUT Update Invoice
http://localhost:8091/invoices/8/status
404 • 126 ms • 260 B • 1

FAIL INV_08 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 404

Iteration 9

PUT Update Invoice
http://localhost:8091/invoices/9/status
404 • 217 ms • 260 B • 1

FAIL INV_09 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 404

Iteration 10

PUT Update Invoice
http://localhost:8091/invoices/10/status
404 • 216 ms • 260 B • 1

FAIL INV_10 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 404

Iteration 11

PUT Update Invoice
http://localhost:8091/invoices/1/status
404 • 189 ms • 260 B • 1

FAIL INV_11 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 404

Iteration 12

PUT Update Invoice
http://localhost:8091/invoices/2/status
404 • 240 ms • 260 B • 1

FAIL INV_12 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 404

Iteration 13

PUT Update Invoice
http://localhost:8091/invoices/3/status
200 • 182 ms • 267 B • 1

FAIL INV_13 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 200

Iteration 14

PUT Update Invoice
http://localhost:8091/invoices/4/status
200 • 415 ms • 267 B • 1

FAIL INV_14 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 200

Iteration 15

PUT Update Invoice
http://localhost:8091/invoices/5/status
404 • 263 ms • 260 B • 1

FAIL INV_15 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 404

Iteration 16

PUT Update Invoice
http://localhost:8091/invoices/6/status
404 • 131 ms • 260 B • 1

FAIL INV_16 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 404

Iteration 17

PUT Update Invoice
http://localhost:8091/invoices/7/status
200 • 404 ms • 267 B • 3

PASS INV_17 - Status Code is 200
PASS INV_17 - API Response says Success
PASS INV_17 - Verify Persisted Status Change

Iteration 18

PUT Update Invoice
http://localhost:8091/invoices/99999/status
200 • 278 ms • 268 B • 2

PASS INV_18 - Status Code is 200
PASS INV_18 - Verify System Rejected Request

Iteration 19

PUT Update Invoice
http://localhost:8091/invoices/abc/status
200 • 303 ms • 268 B • 2

PASS INV_19 - Status Code is 200
PASS INV_19 - Verify System Rejected Request

Iteration 20

PUT Update Invoice
http://localhost:8091/invoices/-1/status
200 • 277 ms • 268 B • 2

PASS INV_20 - Status Code is 200
PASS INV_20 - Verify System Rejected Request

Iteration 21

PUT Update Invoice
http://localhost:8091/invoices/0/status
200 • 449 ms • 268 B • 2

PASS INV_21 - Status Code is 200
PASS INV_21 - Verify System Rejected Request

Iteration 22

PUT Update Invoice
http://localhost:8091/invoices/' OR 1=1 --/status
200 • 229 ms • 268 B • 2

PASS INV_22 - Status Code is 200
PASS INV_22 - Verify System Rejected Request

Iteration 23

PUT Update Invoice
http://localhost:8091/invoices/@#$/status
422 • 315 ms • 504 B • 1

PASS INV_23 - Status Code is 422

Iteration 24

PUT Update Invoice
http://localhost:8091/invoices/8/status
200 • 278 ms • 267 B • 3

PASS INV_24 - Status Code is 200
PASS INV_24 - API Response says Success
PASS INV_24 - Verify Persisted Status Change

Iteration 25

PUT Update Invoice
http://localhost:8091/invoices/9/status
200 • 277 ms • 267 B • 3

PASS INV_25 - Status Code is 200
PASS INV_25 - API Response says Success
PASS INV_25 - Verify Persisted Status Change

Iteration 26

PUT Update Invoice
http://localhost:8091/invoices/10/status
200 • 299 ms • 267 B • 3

PASS INV_26 - Status Code is 200
PASS INV_26 - API Response says Success
PASS INV_26 - Verify Persisted Status Change

Iteration 27

PUT Update Invoice
http://localhost:8091/invoices/1/status
200 • 292 ms • 267 B • 3

PASS INV_27 - Status Code is 200
PASS INV_27 - API Response says Success
PASS INV_27 - Verify Persisted Status Change

Iteration 28

PUT Update Invoice
http://localhost:8091/invoices/2/status
200 • 146 ms • 267 B • 3

PASS INV_28 - Status Code is 200
PASS INV_28 - API Response says Success
PASS INV_28 - Verify Persisted Status Change

Iteration 29

PUT Update Invoice
http://localhost:8091/invoices/3/status
404 • 215 ms • 260 B • 1

FAIL INV_29 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 404

Iteration 30

PUT Update Invoice
http://localhost:8091/invoices/4/status
200 • 298 ms • 267 B • 1

FAIL INV_30 - Status Code is 400 | AssertionError: expected response to have status code 400 but got 200

Iteration 31

PUT Update Invoice
http://localhost:8091/invoices/5/status
404 • 124 ms • 260 B • 1

FAIL INV_31 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 404

Iteration 32

PUT Update Invoice
http://localhost:8091/invoices/7/status
404 • 330 ms • 260 B • 1

FAIL INV_32 - Status Code is 422 | AssertionError: expected response to have status code 422 but got 404

Iteration 33

PUT Update Invoice
http://localhost:8091/invoices/8/status
200 • 346 ms • 267 B • 3

PASS INV_33 - Status Code is 200

PASS INV_33 - API Response says Success

PASS INV_33 - Verify Persisted Status Change

Iteration 34

PUT Update Invoice
http://localhost:8091/invoices/9/status
200 • 12113 ms • 267 B • 1

FAIL INV_34 - Status Code is 415 | AssertionError: expected response to have status code 415 but got 200

Self – Assessment

Criteria	Outcomes	Grade	Self-Assessed Grade
1	**API 1: POST /users/register**	**40**	**40**
	1.1 Report	10	10
	1.2 Test cases (35)	10	10
	1.3 Screenshots on Testing Tools	10	10
	1.4 Bugs	10	10
2	**API 2: GET /products**	**30**	**30**
	2.1 Report	10	10
	2.2 Test cases (38)	5	5
	2.3 Screenshots on Testing Tools	10	10
	2.4 Bugs	5	5
3	**API 3: PUT /invoices/{invoiceId}/status**	**30**	**30**
	3.1 Report	10	10
	3.2 Test cases (34)	5	5
	3.3 Screenshots on Testing Tools	10	10
	3.4 Bugs	5	5
	Total	**100**	**100**