AUGUST 5

QA Portfolio – Manual, API & Test Cycle Reports

Deepika S

Contents

Table of Contents

Section 1: API Testing with Postman

Project Summary:

This section demonstrates real-time API testing done using Postman.
Includes GET, POST methods, header/body validation, and JSON schema checks.
Positive and negative test cases are covered with screenshots and explanations.

Tools Used:

Postman **API**

Used:

https://jsonplaceholder.typicode.com

Skills Demonstrated:

- API Parameter Usage
- Response Header Validation
- Handling Empty Results

✅ Part 1: Response with Headers

Request URL:

`GET https://jsonplaceholder.typicode.com/posts/1`

Expected:

- Status Code: 200 OK
- Response Headers include:
  - Content-Type: application/json
  - Cache-Control: max-age=43200

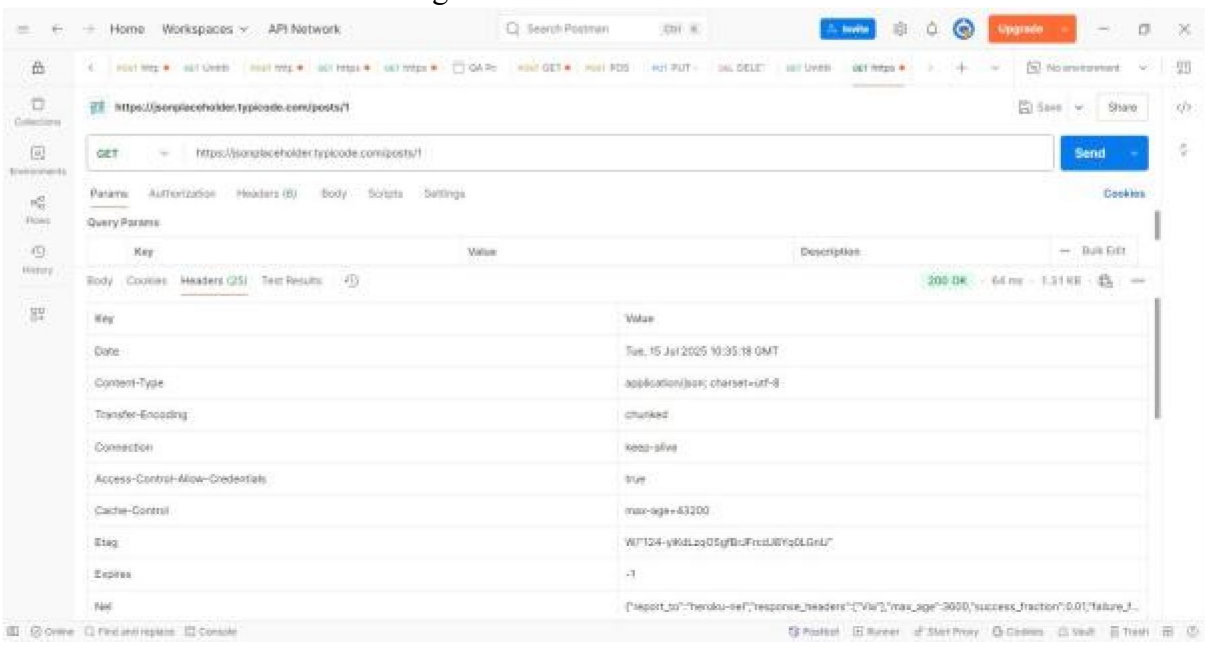


Figure 1: GET Request in Postman – API Response Headers Displayed

This screenshot shows a successful `GET` request made to the endpoint `https://jsonplaceholder.typicode.com/posts/1` using Postman.
The **Headers tab** displays important response metadata like `Content-Type`, `Status: 200 OK`, `Cache-Control`, and `Connection`.

✓ Part 2: Response with Params

Request URL:

`GET https://jsonplaceholder.typicode.com/posts?userId=2`

Expected:

- Status Code: 200 OK
- Response Body returns only posts created by userId 2

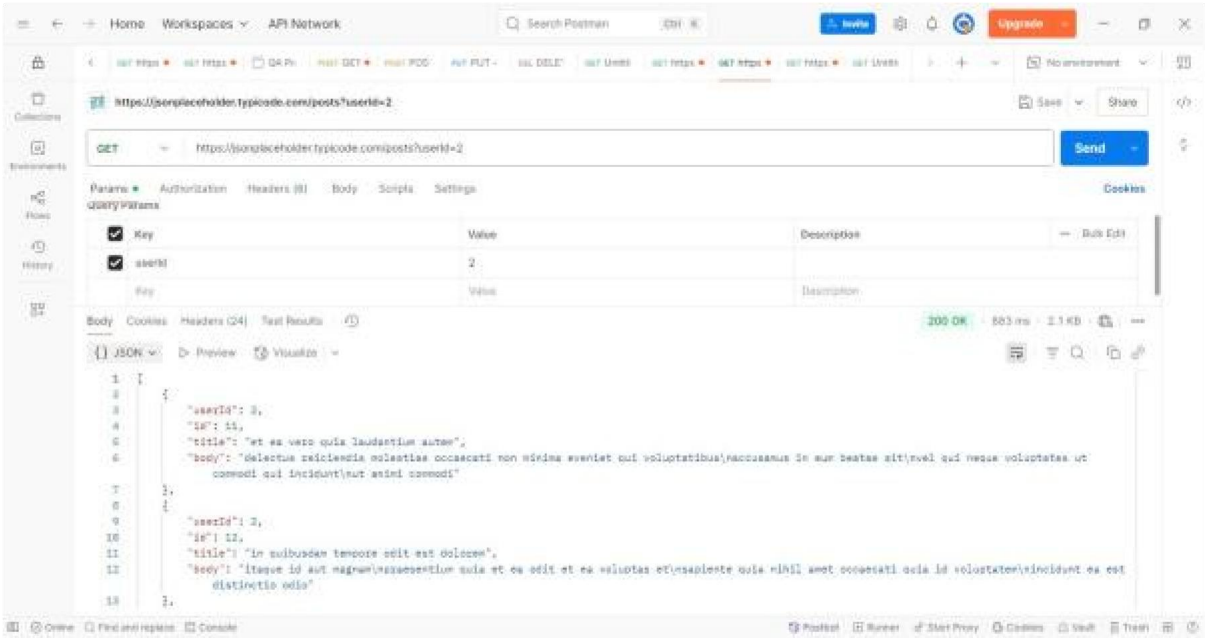


Figure 2: GET Request with Query Parameter – JSON Response in Postman

This screenshot demonstrates a successful `GET` request to the endpoint `https://jsonplaceholder.typicode.com/posts?userId=2` using Postman.
The **Query Parameter** `userId=2` filters results, and the **Body tab** shows the returned JSON array of posts created by the user with ID 2.
The response status code is `200 OK`, indicating the request was successful.

✓ Part 3: Failed or Empty Response

Request URL:

`GET https://jsonplaceholder.typicode.com/posts?userId=999`

Expected:

- Status Code: 200 OK

- Response: Empty array `[]`
- UI (if built on top) should show: "No posts found"

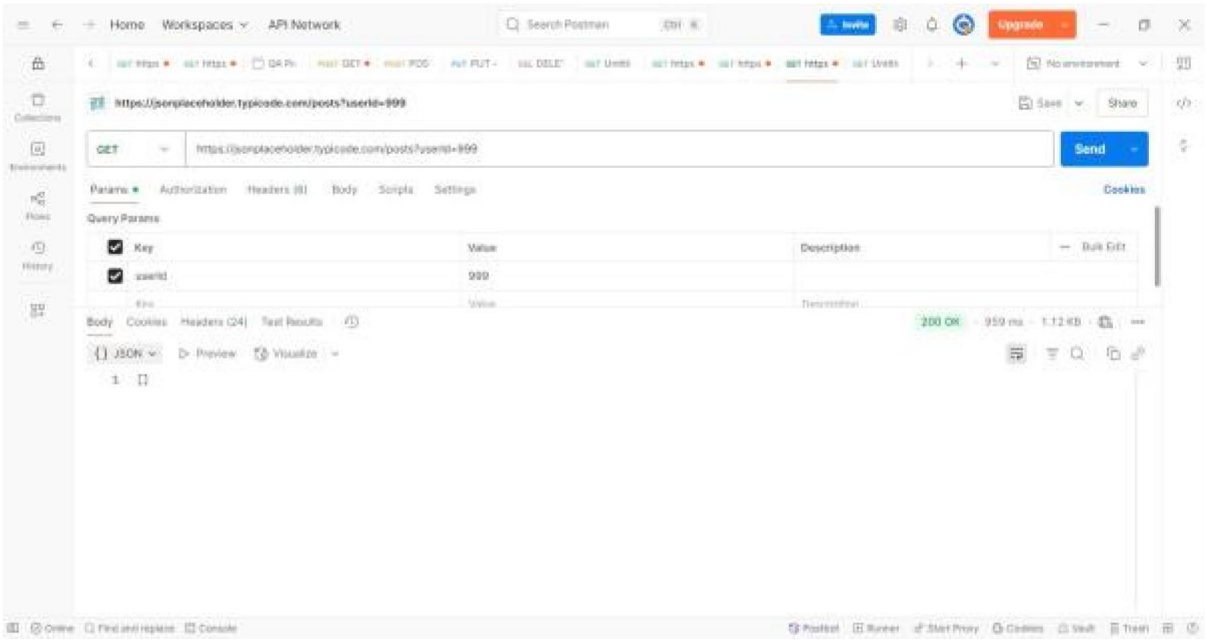


Figure 3: GET Request with Invalid User ID – Empty JSON Response

This screenshot shows a `GET` request made to the endpoint `https://jsonplaceholder.typicode.com/posts?userId=999` using Postman.
The **Query Parameter** `userId=999` does not match any data in the system, so the API returns an **empty JSON array** `[]`.
The response still shows a **200 OK** status, meaning the request was valid but returned no results.

QA Analysis Summary

Test Part	Validation Done
Headers Check	Verified Content-Type & cache-control
Param Testing	Confirmed filtering via userId
Empty Data Handling	Observed blank result (no crash/bug)

Conclusion:

Deepika S. has demonstrated the ability to:

- Perform GET requests with and without parameters
- Understand and validate HTTP status codes and headers
- Handle empty JSON responses gracefully in test cases

This portfolio adds value as proof of practical API testing knowledge using Postman

Section 2: Parabank Manual Test Case

Project Summary:

Profile update functionality was manually tested on the Parabank demo app.
UI changes were confirmed through Chrome DevTools, and bugs were reported when API failed to update fields properly.

Demo Site: https://parabank.parasoft.com

Objective:

Verify that updated profile info reflects immediately on screen without needing a refresh.

Test Case Details

Field	Description
Test Case ID	TC-ProfileUpdate-Parabank
Test Title	Verify Profile Info Updates Immediately on Parabank
Precondition	User must be logged into the site
Test Steps	1. Click "Update Contact Info"2. Change name/address3. Click "Update Profile"
Expected	Profile page should show updated data immediately (without refresh)
Actual	Profile was updated and reflected instantly
Status	✓ Pass

Figure 4: Update Profile Form – ParaBank Application

This screenshot shows the **Update Profile** page from the ParaBank demo application.
The user fills out fields such as `First Name`, `Last Name`, `Address`, `City`, `State`, `Zip Code`, and `Phone Number`.Clicking the **"UPDATE PROFILE"** button submits the changes, triggering a `POST` API request behind the scenes.

Section 3: Chrome DevTools API Validation

Project Summary:

This section explains how Chrome DevTools' Network tab was used to verify background API calls during profile update without page refresh.

Screenshot: DevTools Validation – Parabank Profile Update (Network Tab)

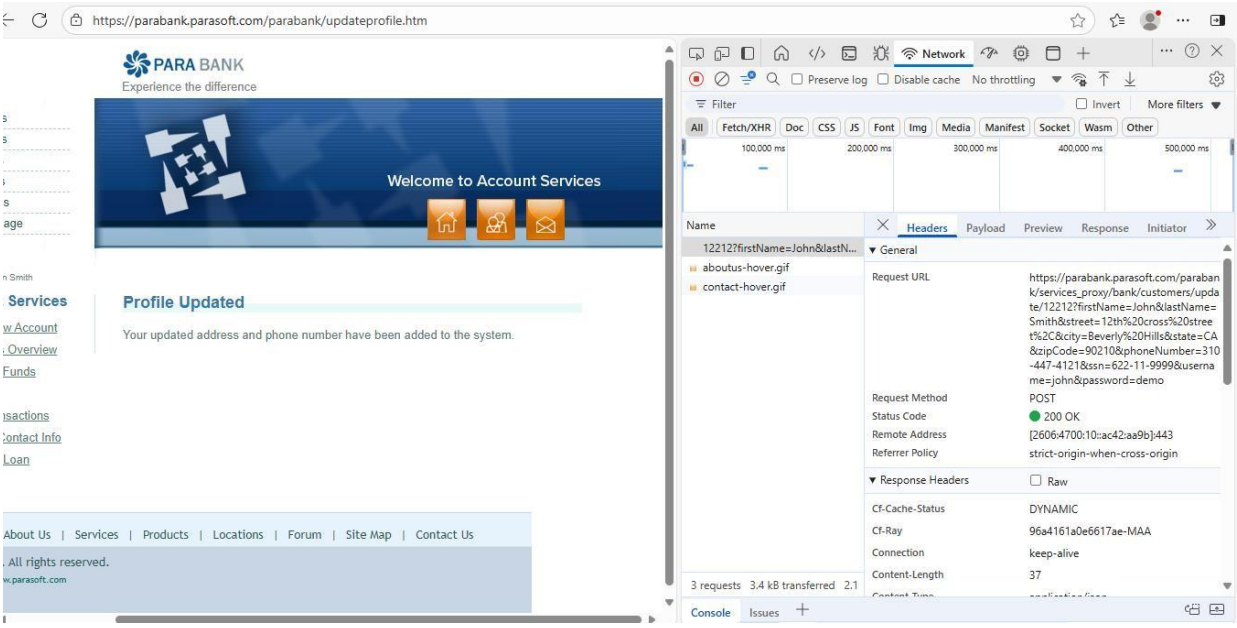


Figure 5: ParaBank Profile Update – API Request in Chrome DevTools

This screenshot captures the **successful profile update action** on the ParaBank web app and its corresponding **API call** shown in Chrome DevTools under the Network tab.

- **Method:** `POST`
- **Status Code:** `200 OK`
- **Request URL:** Contains user details like name, address, city, phone, etc.
- The server responds without reloading the page, confirming a real-time update using **REST API**.

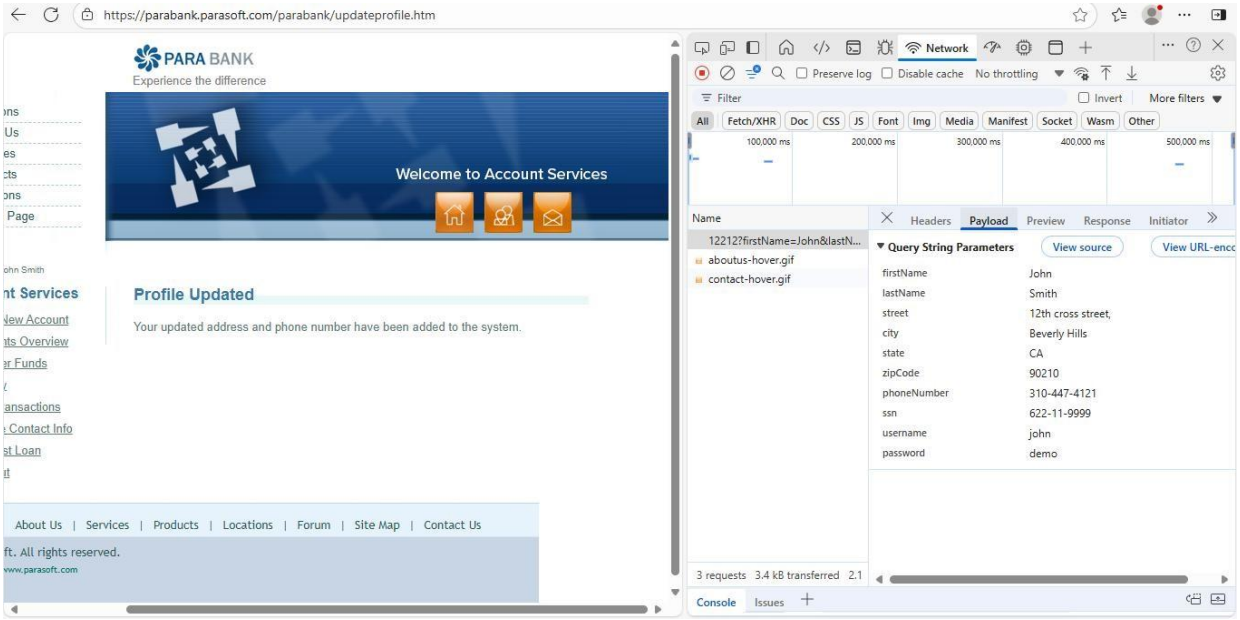


Figure 6: ParaBank Profile Update – API Payload Details via DevTools

This screenshot shows the **Payload tab** of the **POST API request** made during the profile update process on ParaBank. It displays all **user input data** sent to the server, including:

- `firstName`, `lastName`, `street`, `city`, `state`, `zipCode`, `phoneNumber`
- Authentication details like `username` and `password`

This confirms how real-time form submissions send user data as **query string parameters** via REST API, without a page refresh.

Test Scenario:
Verify that user profile updates are reflected immediately and a real-time API call is triggered.

Tool Used: Chrome DevTools → Network Tab

Steps:

1. Logged in as user `john`
2. Navigated to **Update Contact Info**
3. Submitted new address and phone number
4. Opened **DevTools → Network tab**
5. Captured a **POST request** to the update profile API

Observations:

- **API URL:** `/services_proxy/bank/customers/update`
- **Method:** `POST`
- **Status Code:** `200 OK` (Successful update)
- **Parameters:** Sent updated fields like `firstName`, `lastName`, `phoneNumber`
- No page refresh needed → **UI instantly updated**

Section 4: Zephyr + Jira Test Cycle Summary

Project Summary:

Manual test cases were written and executed in Zephyr.
Test results (Pass/Fail) were tracked, and bugs were reported in Jira with proper linking to test cases.

1. Tester Name:

Deepika S

2. Project: QA Bug Tracker

3. Test Cycle:

Login + Profile Cycle - Sprint 1

4. Execution Date:

08 July 2025

5. Total Test Cases: 4 Total - 3 Passed, 1 Failed

6. Websites Tested:

- https://the-internet.herokuapp.com/login - Login Valid & Invalid -
https://opensource-demo.orangehrmlive.com - Login & Profile Scenarios

Test Case Execution Summary:

QBT-T1: Login with valid credentials - Passed (Herokuapp, OrangeHRM)

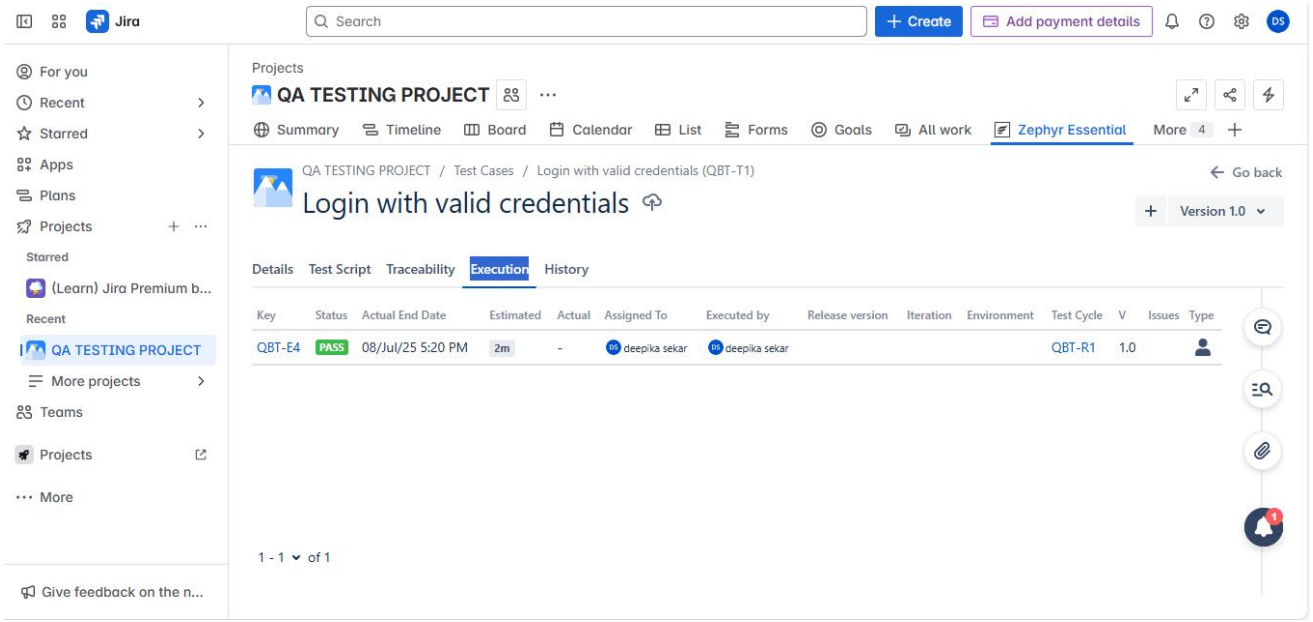


Figure 7: Zephyr test case execution – QBT-T1

QBT-T2: Login fails with incorrect username - Passed (Herokuapp, OrangeHRM)

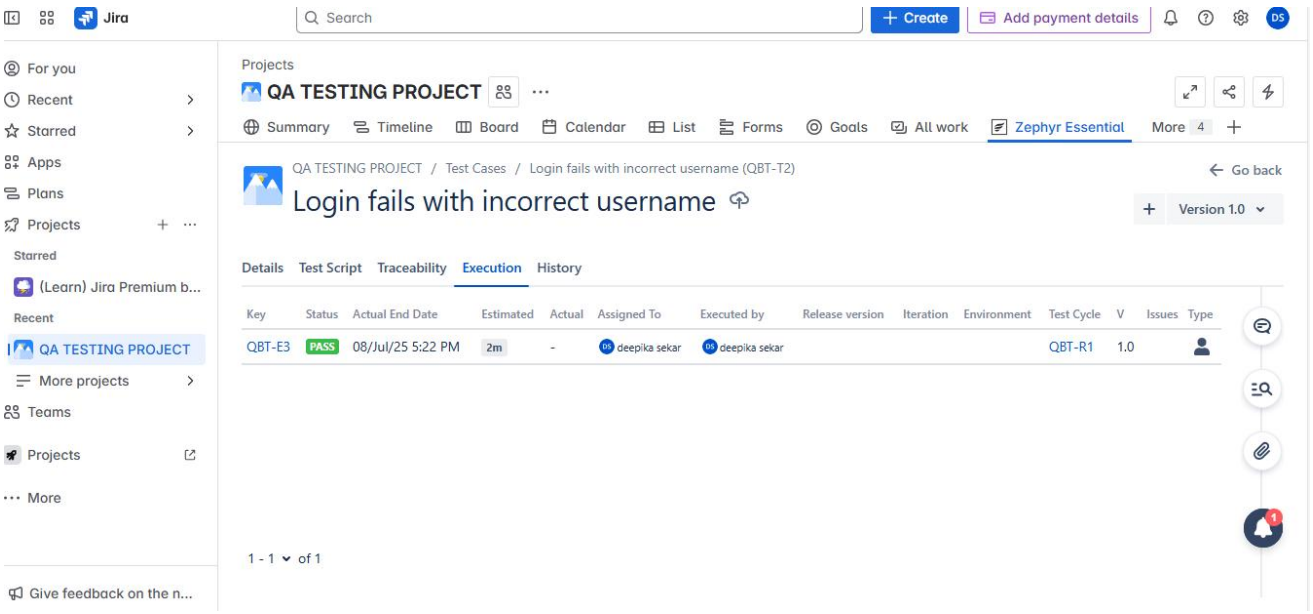


Figure 8: Zephyr test case execution – QBT-T2

QBT-T3: Login fails with incorrect password – Failed(Bug) (Herokuapp)

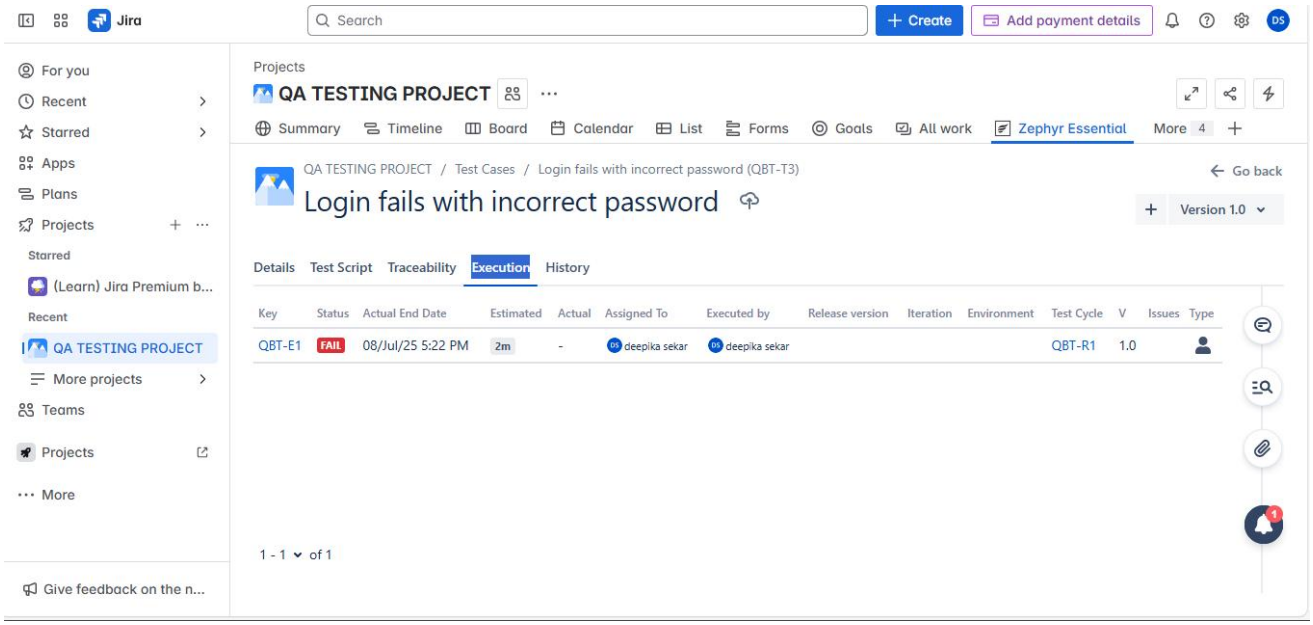


Figure 9: Zephyr test case execution – QBT-T3

QBT-T4: Profile update validation - Passed(OrangeHRM)

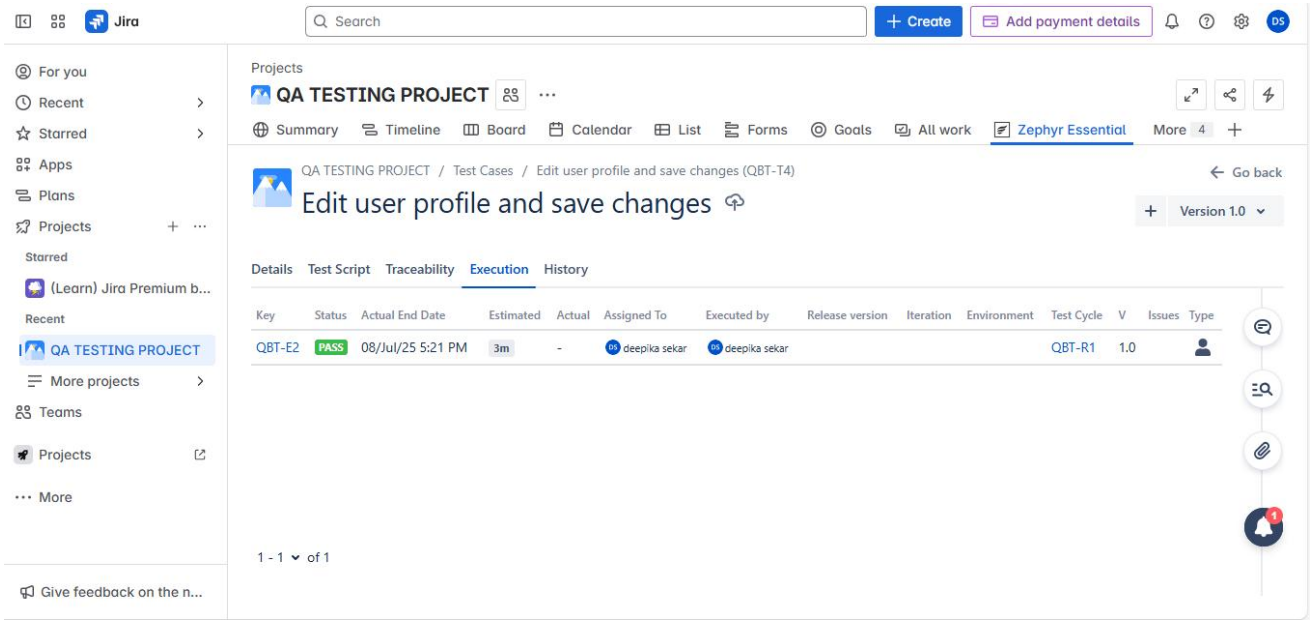


Figure 10: Zephyr test case execution – QBT-T4

Section 5: Bug Reports

QBT-3: Username field clears after failed login - Reported in Jira

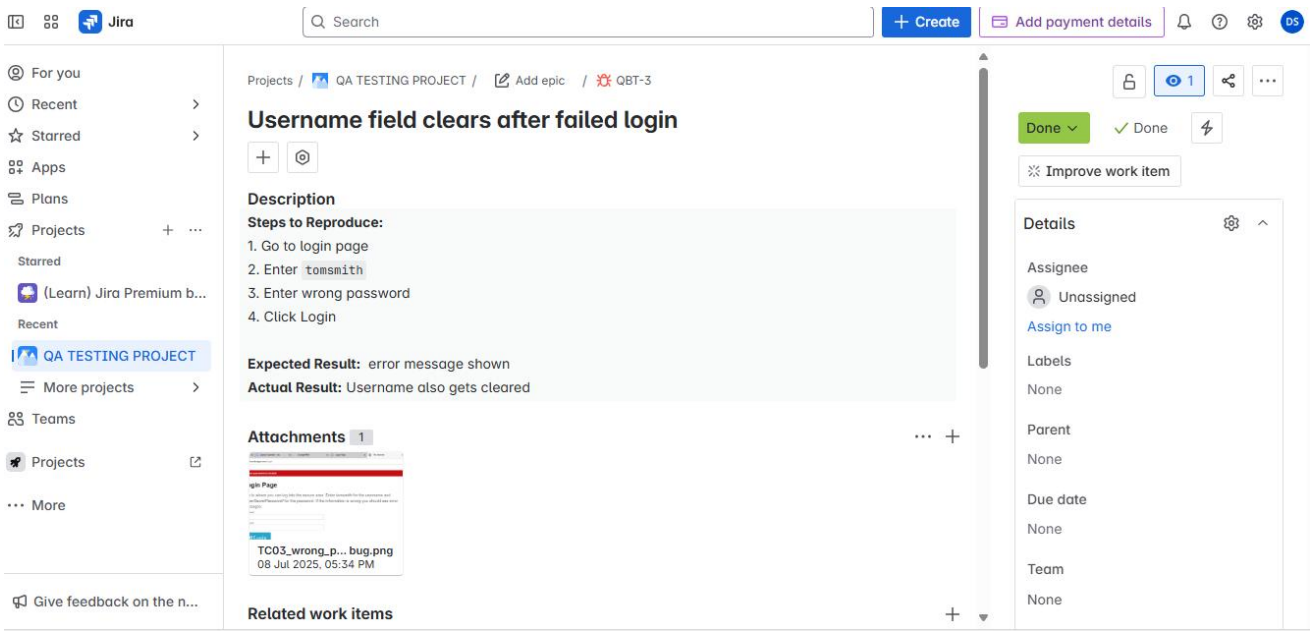


Figure 11: Bug Report in Jira – Username Field Clears on Failed Login

This screenshot displays a **logged bug** in **Jira** under the "QA TESTING PROJECT."
It highlights an issue where the **username field gets cleared** after submitting an incorrect password during login. The bug report includes:

- **Steps to Reproduce** the issue
- **Expected vs Actual Result**
- A **screenshot attachment** of the login page
- Status marked as **Done**

This demonstrates **effective bug documentation and tracking** using Jira.

Section 6:TestRail Login Module QA

Project Summary

This project focused on validating the Login functionality of a demo web application using both positive and negative test scenarios. The objective was to ensure that the system behaves correctly when provided with valid credentials and handles incorrect input gracefully.

The QA process involved writing and executing test cases in TestRail, simulating bugs using real browsers, and documenting actual vs expected results. A total of 3 test cases were executed covering:
- Valid login
- Invalid username
- Invalid password

Test execution results revealed a critical bug: The system allowed login with an incorrect username, indicating a validation failure on the client or server side.

This defect was captured and documented using screenshots and detailed steps for reproduction.

Tools used
- Test Management: TestRail
- Bug Tracking: jira
- Testing Type: Manual (Positive & Negative Testing)
- Browser Tools: Chrome (UI behavior validation)

Test Cases executed

TC ID	Title	Test Type	Expected Result	Status
T4	Login with valid credentials	Positive	User should log in successfully	Passed
T5	Login fails with wrong username	Negative	Error message should be shown	Failed
T6	Login fails with wrong password	Negative	Error message should be shown	Passed

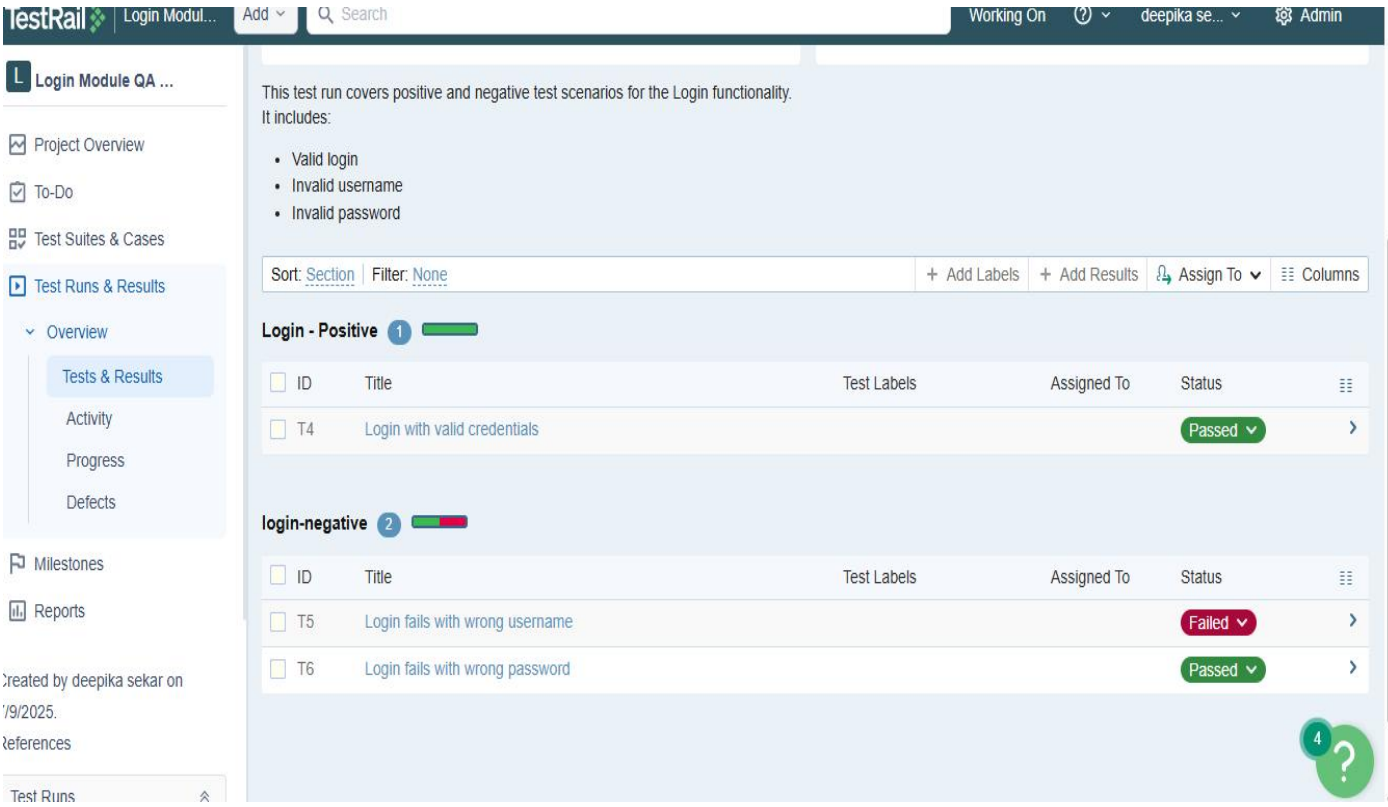


Figure 12: TestRail Test Run Report for Login Functionality

This report displays test results for the login module, covering both positive and negative test scenarios:

- ✅ T4: Login with valid credentials – **Passed**
- ❌ T5: Login with wrong username – **Failed** (Bug reported in Jira)
- ✅ T6: Login with wrong password – **Passed**

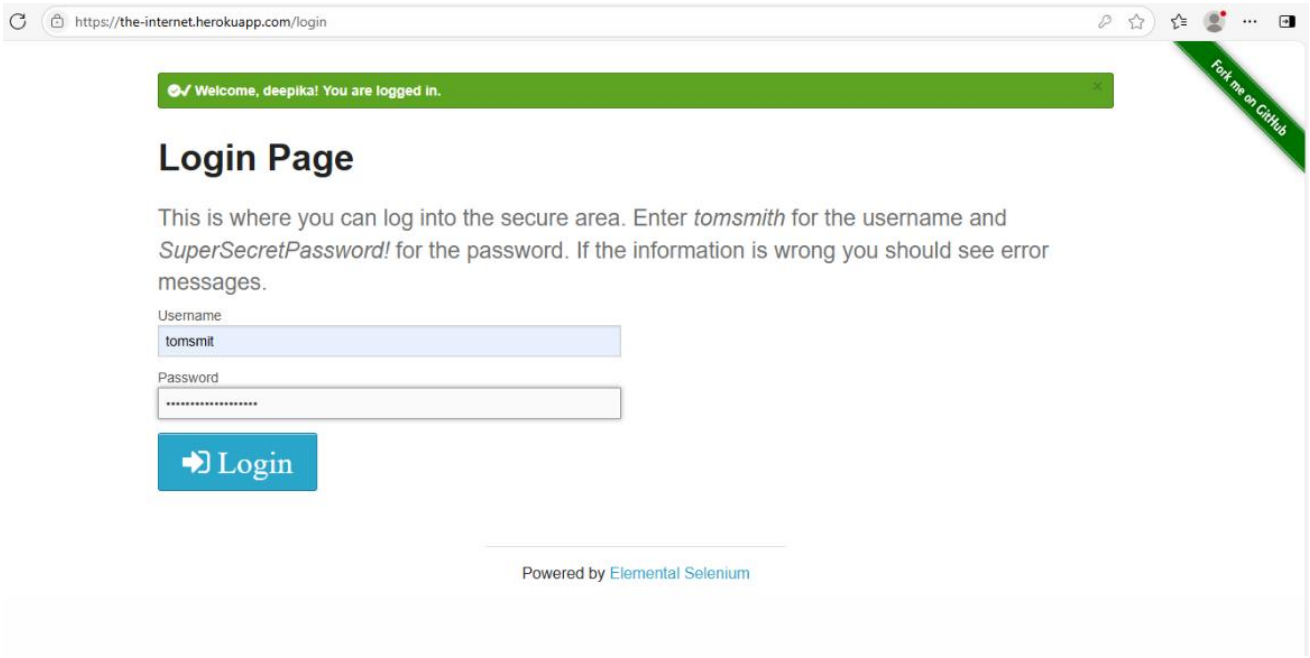


Figure 13: Web Application Login Page – Unexpected Success Message for Invalid Username
This screen shows a bug scenario where the user entered an invalid username (`tomsmit`) but still received a success message:
✅ *"Welcome, deepika! You are logged in."*
This is an incorrect behavior. The expected result was an error message: ❌ *"Invalid username."* The issue has been logged in Jira.

Bug report

- Bug Title: Invalid Username Accepts Login

Description: The system incorrectly allows login with an invalid username, displaying a success message.
Steps to Reproduce:
1. Go to the login page.
2. Enter 'tomsmit' (misspelled username).
3. Enter valid password 'SuperSecretPassword!'.
4. Click Login.
Expected Result: Should show error message like 'Invalid username or password'.
Actual Result: Shows success message: '✓ Welcome, deepika! You are logged in.'

Severity: High
Module: Login
Test Case Reference: T5
Test Execution Tool: TestRail
Linked Screenshot:

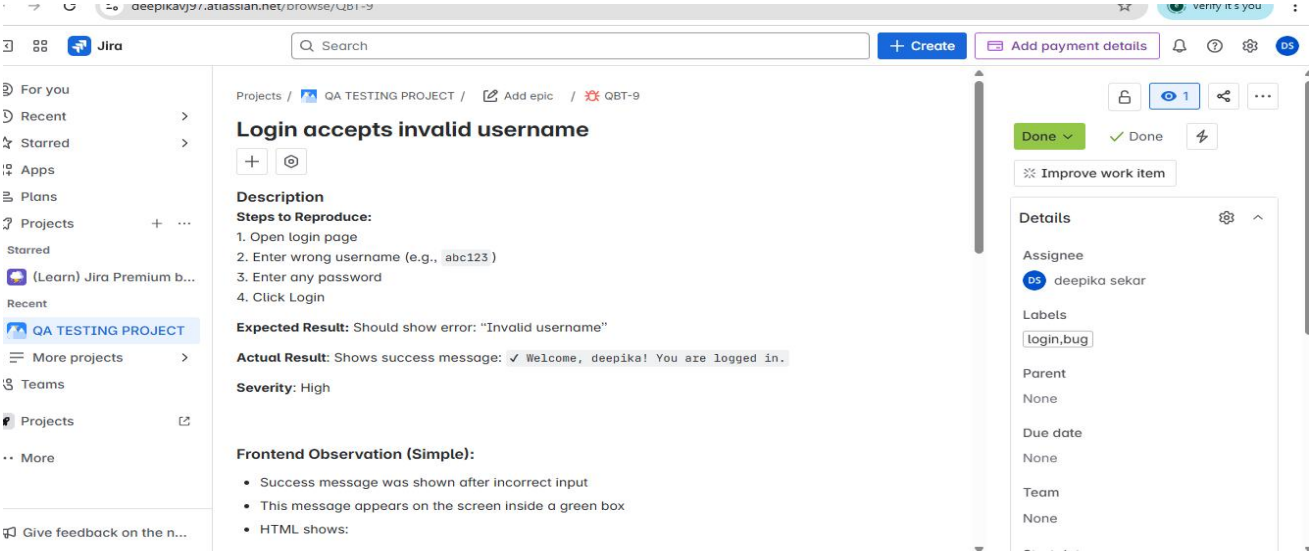


Figure 14: Jira Bug Report – Login Success with Invalid Username

This screenshot shows a Jira bug logged for a critical issue where the **login system accepts an invalid username** and incorrectly displays a success message:
✓ *"Welcome, deepika! You are logged in."*
The bug report includes:

- Summary and bug type
- Steps to reproduce
- Expected vs Actual behavior
- Status and severity

This demonstrates proper defect tracking and documentation in **real-time bug management using Jira**.

Status

COMPLETED – Deepika S. has successfully completed a manual testing test cycle using TestRail and reported a realistic login bug scenario.

Section 7: Skills Demonstrated

Project Summary:

This section highlights all key QA skills practiced during the completion of the portfolio.
It includes test design, test execution using Zephyr, API testing in Postman, DevTools analysis, bug reporting in JIRA, and result tracking in tools like TestRail.
It demonstrates practical understanding of both **manual** and **tool-based** testing workflows in real-time applications.

- Manual Test Case Design
- Writing Test Scenarios based on User Stories
- API Testing using Postman
- JSON Structure and Status Code Validation
- Using Chrome DevTools for API Observations
- Bug Reporting in JIRA (with severity, steps, screenshot)
- Zephyr/JIRA Test Execution Tracking
- Test Cycle Reporting using TestRail
- Real-Time Testing on Live Websites like:
  - Parabank
  - Herokuapp
  - OrangeHRM

Status: COMPLETED
Deepika S. has completed a real-time QA portfolio covering all stages of manual and API testing, from Epic to bug closure using professional tools like Zephyr, JIRA, Postman, and DevTools.

Project Title: Automation Exercise – Product Search Feature

QA Analyst: Deepika S.

Tools Used: Zephyr, Jira, DevTools (Chrome), Browser Testing

Project Type:Real-time Demo / Practice Project

Project Summary

This project involved end-to-end manual testing of the product search functionality on [AutomationExercise.com](https://automationexercise.com). The feature was tested from user story creation to bug logging using industry tools like Zephyr and Jira.

Epic Description:

This Epic covers all QA efforts related to product search functionality including keyword-based search, minimum character validation, API behavior, and empty search result UX handling.

User Story: Keyword-Based Product Search

As a user, I want to search for products using keywords, so that I can quickly find matching items from the store.

Test Cases (from Zephyr Execution)

Test Case ID	Test Name	Objective	Status
QBT-T5 (TC006)	Search with valid keyword	Verify products are returned for valid/partial keywords	✕ Bug – Valid input like "shoes" shows blank page
QBT-T7 (TC008)	Search with no matching results	Verify message is shown for unmatched keywords	✕ Bug – Blank page, no message

QBT-T6 (TC012)	Search with < 3 characters	Ensure system validates input length and prevents short searches	✗ Bug – Search executes incorrectly
QBT-T10 (TC010)	Verify API call for search	Ensure API request is made on search trigger	✓ Approved
QBT-T9 (TC011)	Case-insensitive search	Verify search works regardless of upper/lowercase input	✓ Approved

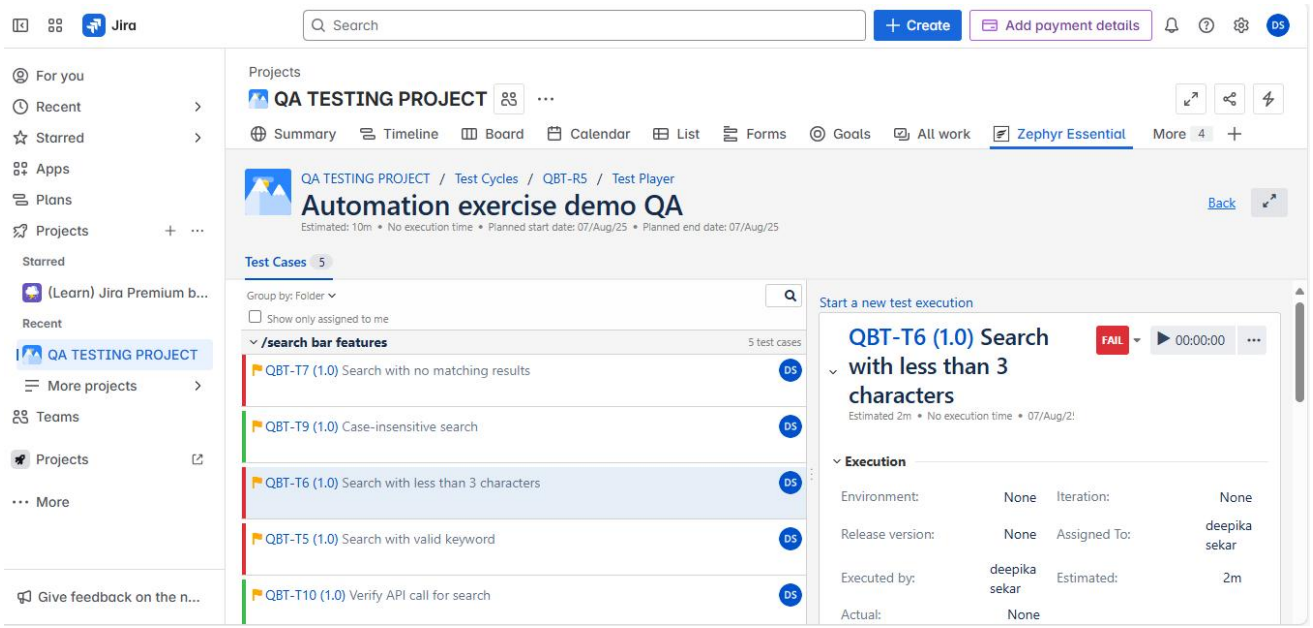
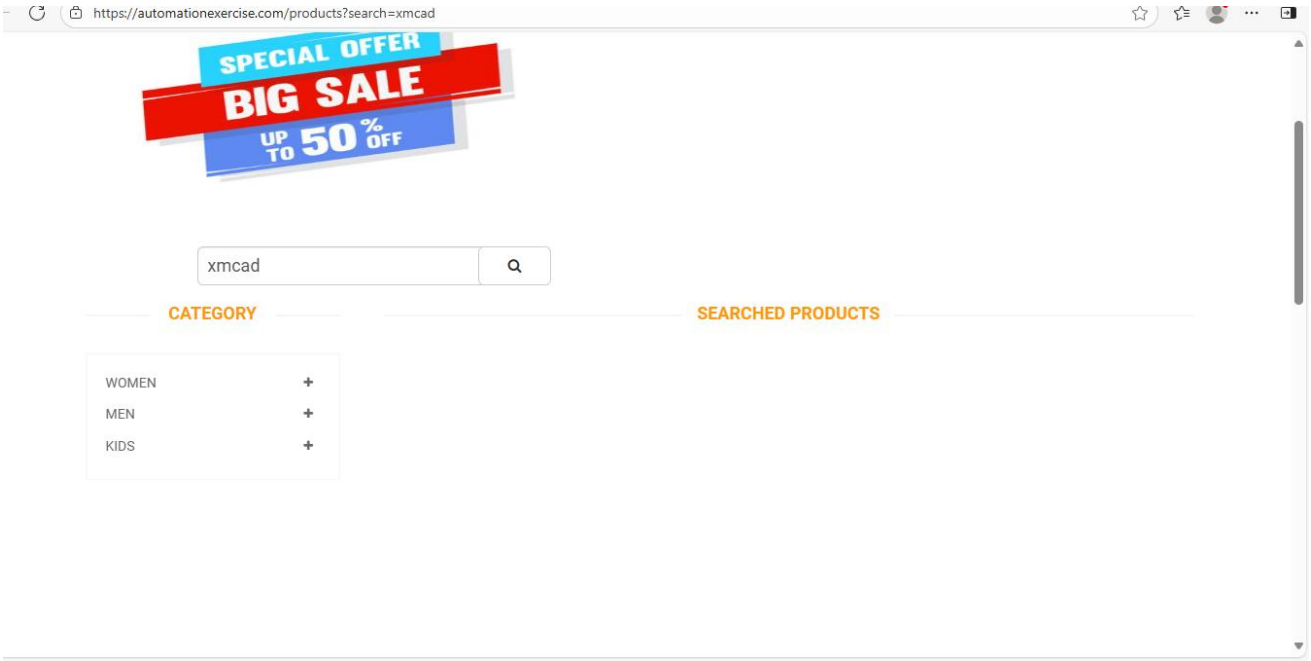


Figure 15: Zephyr Test Execution view showing Pass and Bug statuses for Search-related test cases.

Bug Reports

BUG001: No Feedback Message for Invalid Search

Description: System shows blank page with no "No products found" message
Linked TC: QBT-T7
Severity: Medium



BUG002: Search Executes with <3 Characters

Description: Search runs with 2-character inputs like "dr", even though 3-character minimum is expected

Linked TC: QBT-T6

Severity: Medium

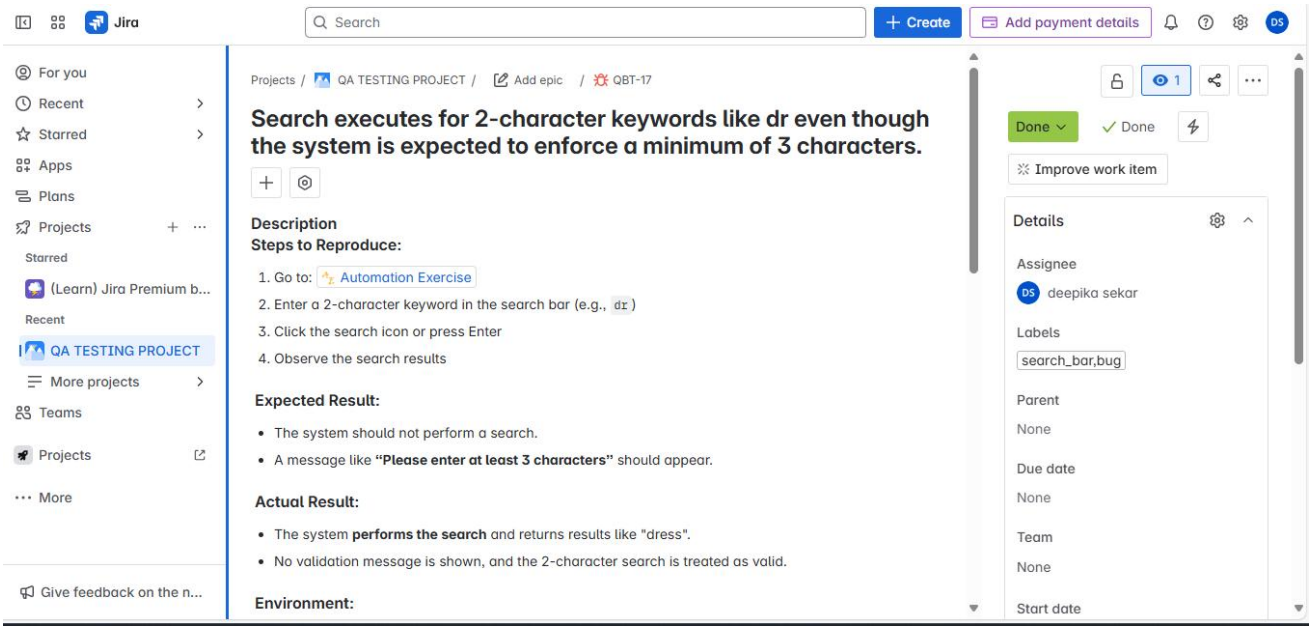


Figure 16**:** *Jira Bug Report – BUG002: The system executes a search request even when the input contains fewer than 3 characters (e.g., "dr"). Expected behavior is to block the search and prompt the user to enter at least 3 characters.*

BUG003: Valid Keyword (e.g., "shoes") shows blank page

Description: When searching for valid input like "shoes", the system shows a blank result page without a "No products found" message. Expected: Either show matching products or display a clear message.

Linked TC: QBT-T5

Severity: High (affects valid input handling)

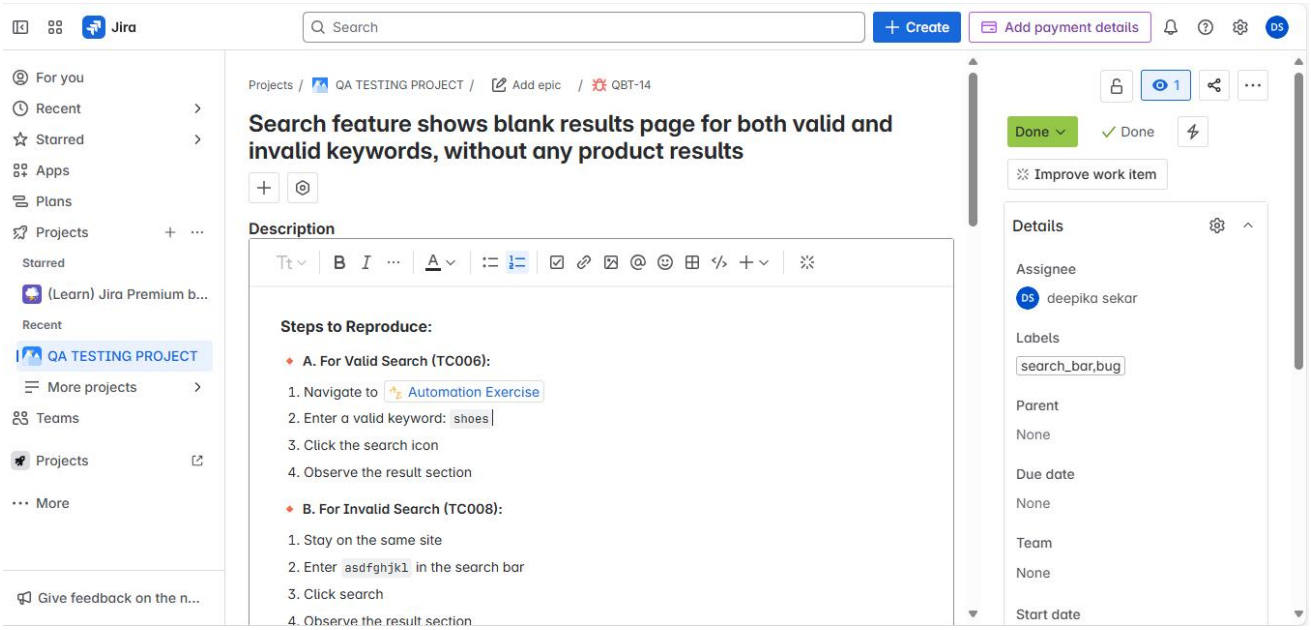


Figure 17: BUG003 and BUG001 in Jira – Search feature shows blank results page for both valid and invalid keywords, without any product results

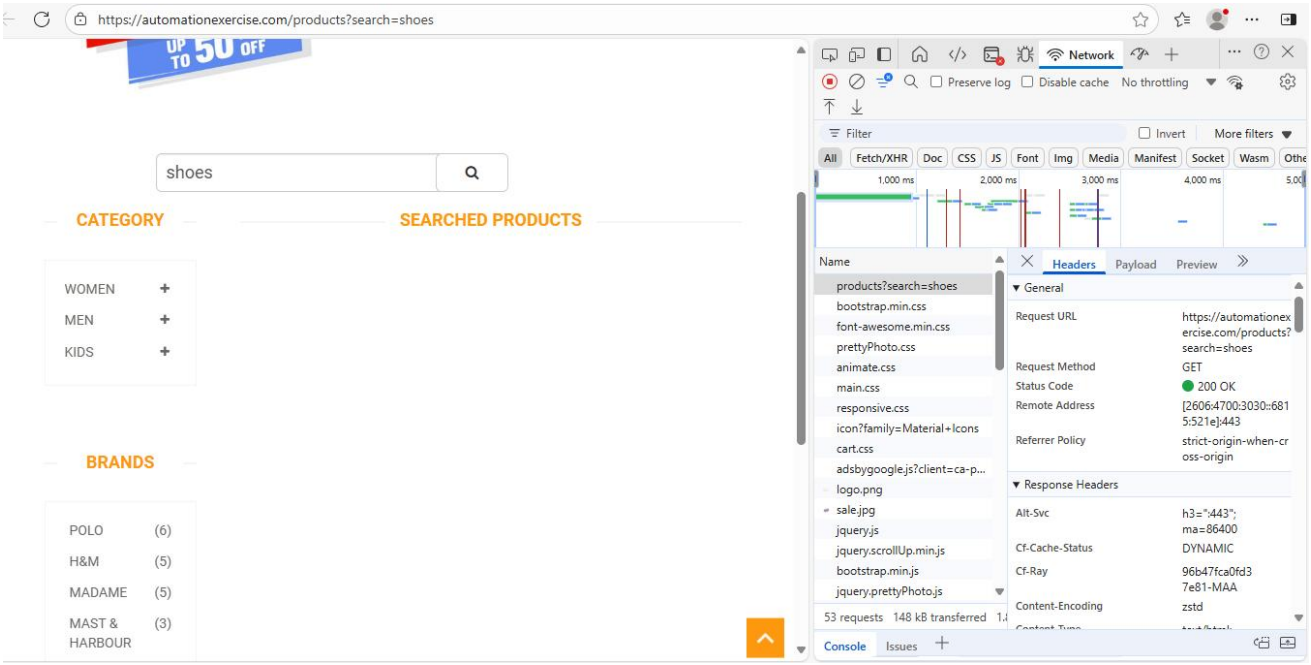


Figure 18: Live bug – Searching for "shoes" triggers a valid GET request (200 OK) to '/products?search=shoes', but results in a blank page with no matching products shown and no "No products found" message. This confirms a frontend issue despite successful backend response.

Test Execution Summary (Zephyr)

All test cases created and executed in Zephyr Essential under the project.
• Approved test cases: 2

• Bug-linked test cases: 3

• Test folder used: search bar features

Link Tested https://automationexercise.com/products?search=dress

Skills Demonstrated

- Writing user stories from features
- Creating detailed manual test cases
- Validating API and browser behavior via DevTools
- Reporting UI/Functional bugs in Jira format
- Executing and logging results in Zephyr

Status: COMPLETED

Deepika S. has successfully completed a real-time QA testing scenario using Zephyr + Jira. This hands-on example shows complete understanding of manual testing workflow from Epic to Bug closure.

1. Project Overview

Project Title: API Testing Portfolio – Dummy Post API
Name: Deepika S.
Date: July 2025
Tools Used: Postman, Zephyr, JSON Server, Jira, Microsoft Word
Objective:
To demonstrate my understanding of REST API testing using a dummy API. I created test cases in Zephyr, executed them in Postman, validated data in a simulated backend using JSON Server, and reported issues using Jira.

2. Zephyr Test Case Summary

The following test cases were designed and maintained in **Zephyr** for validating various REST API operations using `https://jsonplaceholder.typicode.com/posts`. All test cases were executed and approved.

TC ID	Test Case Title	Test Type	Status
TC_01	Create a Post (Valid Data)	Positive	Approved
TC_02	Get a Post by ID	Positive	Approved
TC_03	Update a Post	Positive	Approved
TC_04	Delete a Post	Positive	Approved
TC_05	Create Post with Missing Fields	Negative	Approved
TC_06	Read Post with Invalid ID	Negative	Approved
TC_07	Update Post with Invalid ID	Negative	Approved
TC_08	Delete Post with Invalid ID	Negative	Approved
TC_09	Delete the Same Post Twice	Edge Case	Approved
TC_10	Post with Invalid Content-Type	Negative	Approved
TC_11	Post with Invalid Data Type (userId as text)	Negative	Approved
TC_12	Wrong HTTP Method Used (PUT instead of POST)	Negative	Approved
TC_13	Update with Required Field Left Empty	Negative	Approved

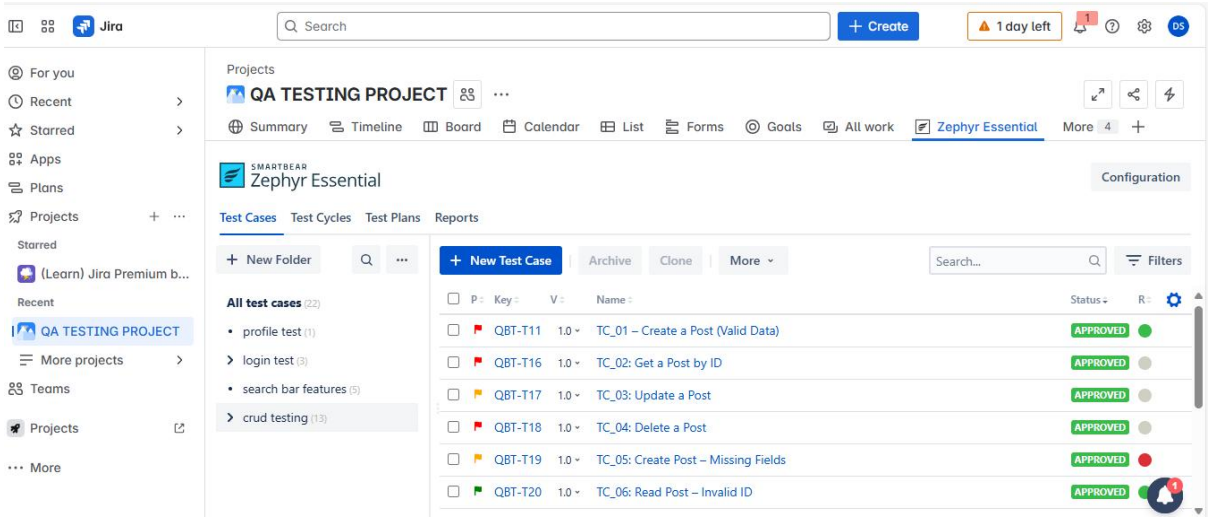


Figure 1: Zephyr Test Cases – TC_01 to TC_06

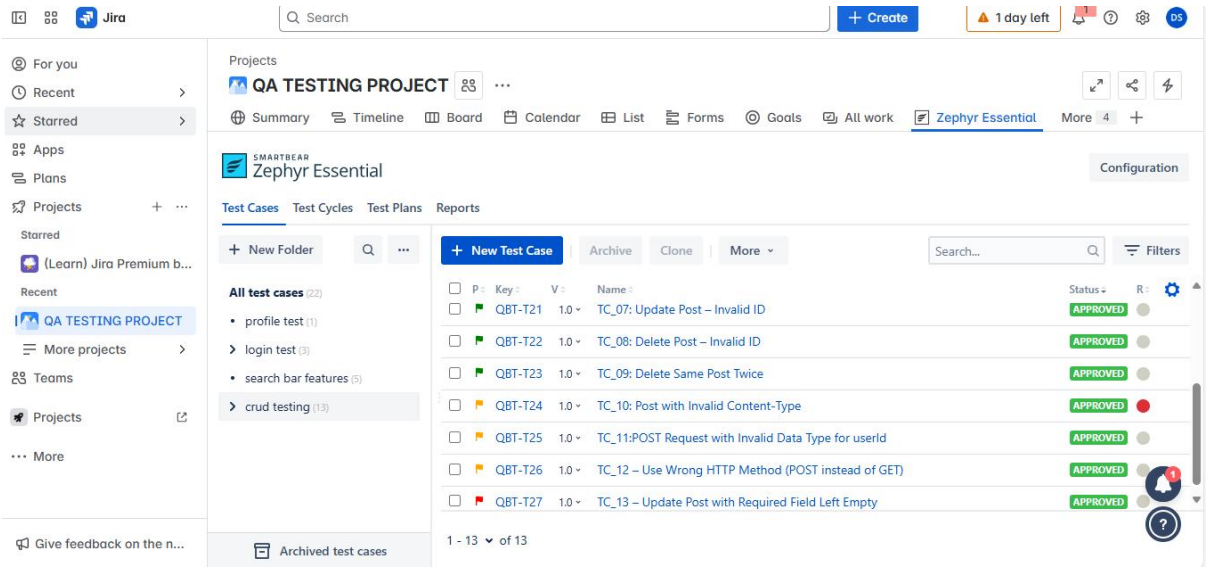


Figure 2: Zephyr Test Cases – TC_07 to TC_13

Test Case Documentation & Validation – POST Endpoint

1. Test Case Title

TC_01 – Create Post with Valid Data

Test case designed and executed using Zephyr for Jira.

2.Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send POST request to `/posts`	`{ "title": "QA Test", "body": "This is a dummy test post", "userId": 1 }`	Status `201 Created` and new `id` generated
2	Validate the response structure	–	Response includes `title`, `body`, `userId`, and a new `id` field
3	Verify DB record (if applicable)	–	DB contains new post with given details

3. **Zephyr Screenshot**

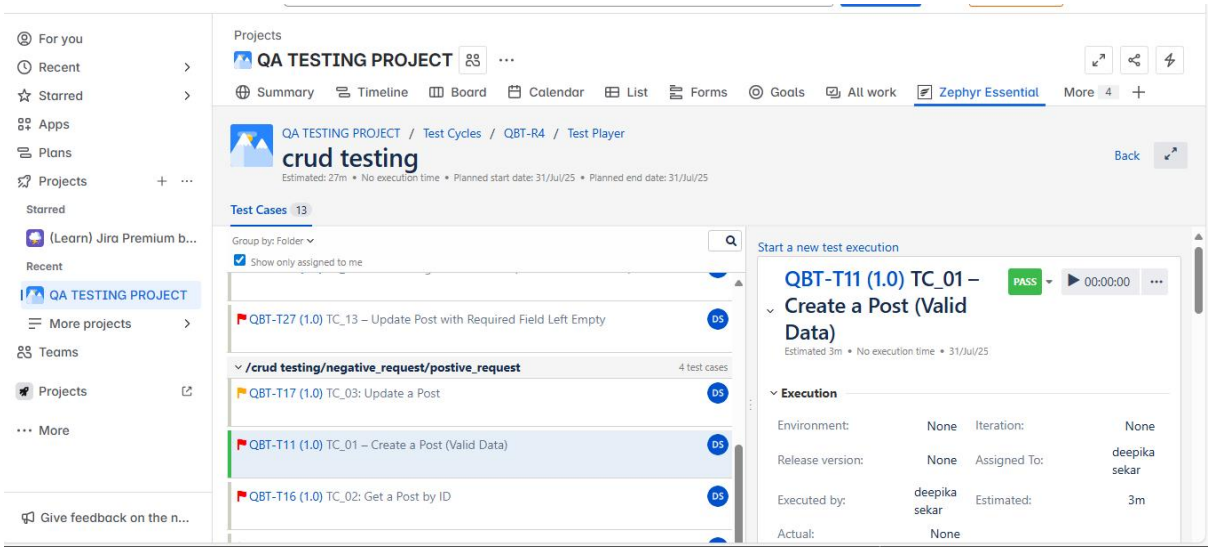


Figure 3: Zephyr Test Case Execution for TC_01

4.Postman Screenshot

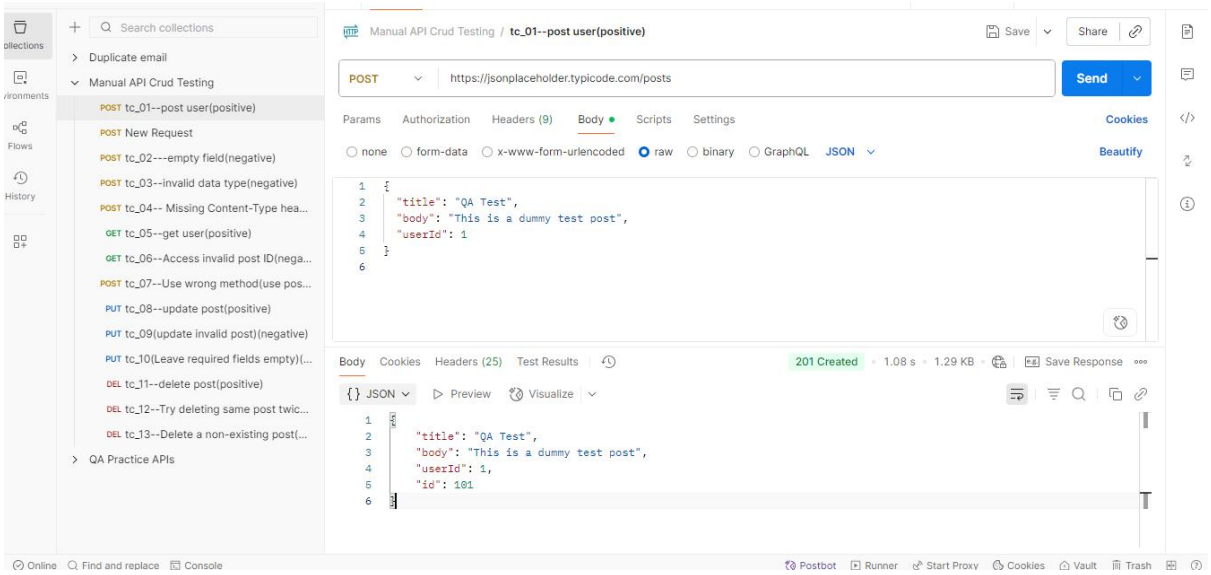


Figure 4: Postman response for TC_01 – 201 Created

5.Database Screenshot

Figure 4: DB entry after TC_01 POST request

In real-time, API backend handles DB insertion. SQLite DB used here only for simulation and DB validation practice.

6.Final Result

✅ Test Passed

The post was created successfully. Zephyr execution marked PASS after validating both the API response and simulated DB record.

---

1. Test Case Title
TC_02 – Get a post by ID
Test case designed and executed using Zephyr for Jira.

2. Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send GET request to /posts/1	ID = 1	Status **200 OK** and post details returned
2	Validate response content	–	Response includes valid `userId`, `id`, `title`, and `body` fields
3	Cross-check DB entry	–	DB contains matching record for `id = 1`

3. Zephyr Screenshot :

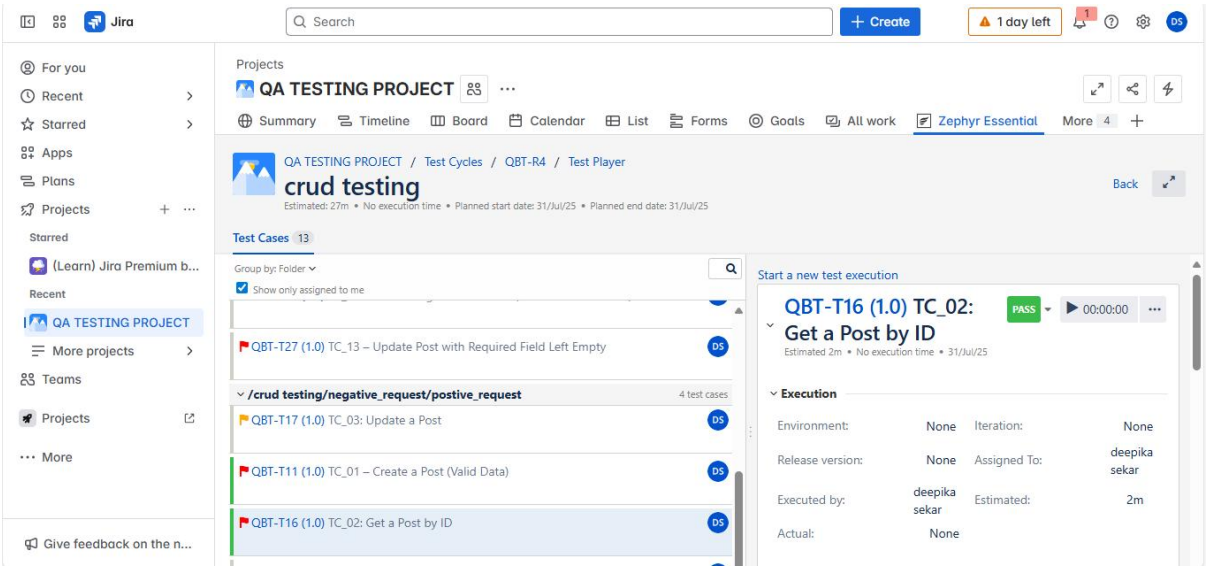


Figure 5: Zephyr Test Case Execution for TC_02

4. Postman Screenshot

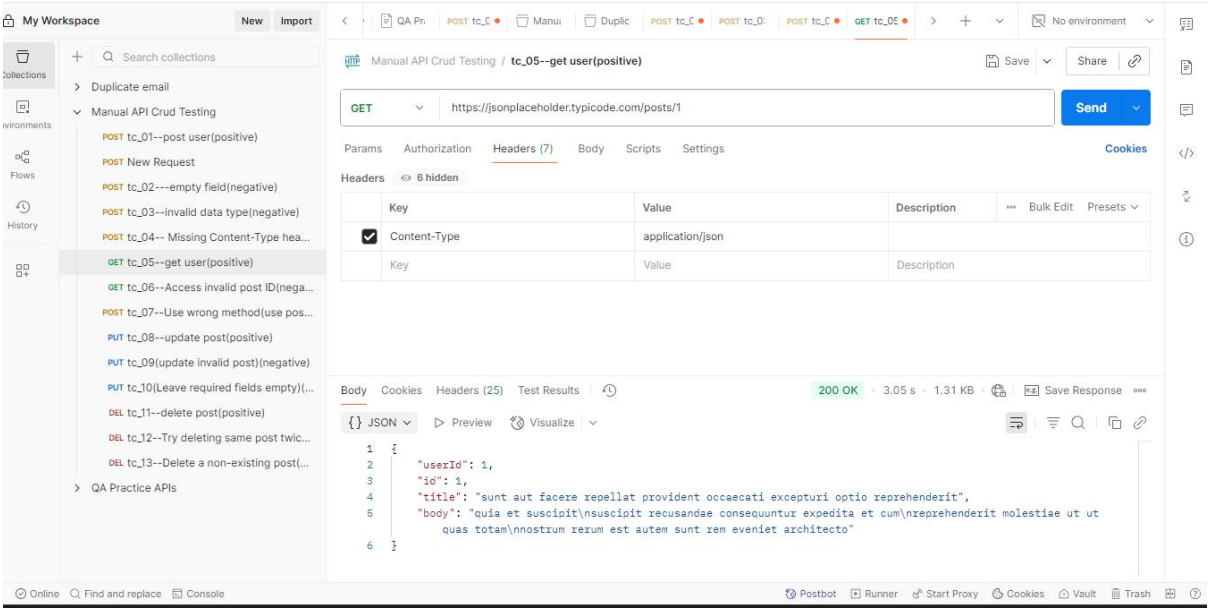


Figure 6: Postman response for TC_02 – 200 OK

5. Database Validation Screenshot

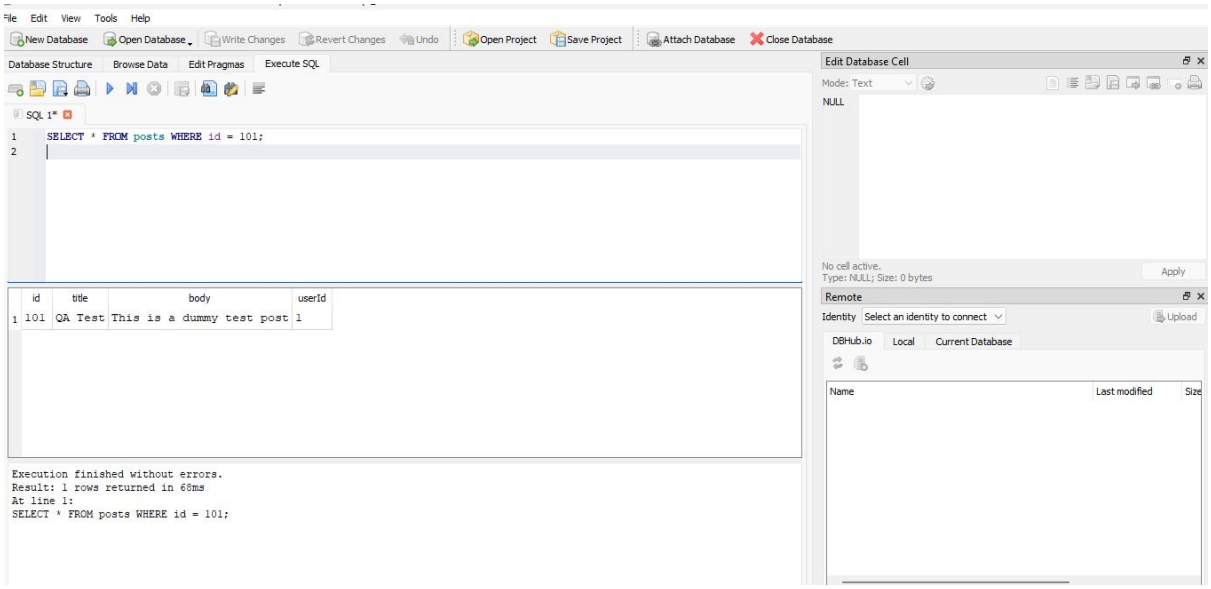


Figure 7: DB entry for Post ID 1 after GET request

✓ 6. Final Result

Test Passed ✓

The post with ID = 1 was successfully retrieved via API. Response body and DB record were verified. Zephyr execution marked as **PASS**.

---

1. Test Case Title

TC_03 – Update an post

Test case designed and executed using **Zephyr for Jira**.

2. Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send PUT request to `/posts/101`	`{ "id": 101, "title": "Updated Title", "body": "Updated content", "userId": 1 }`	Status 200 OK and updated content in response
2	Validate the response content	–	Response includes updated title and body fields
3	Cross-check DB entry	–	Database record with `id = 101` shows the updated values (title & body fields)

3. Zephyr Screenshot

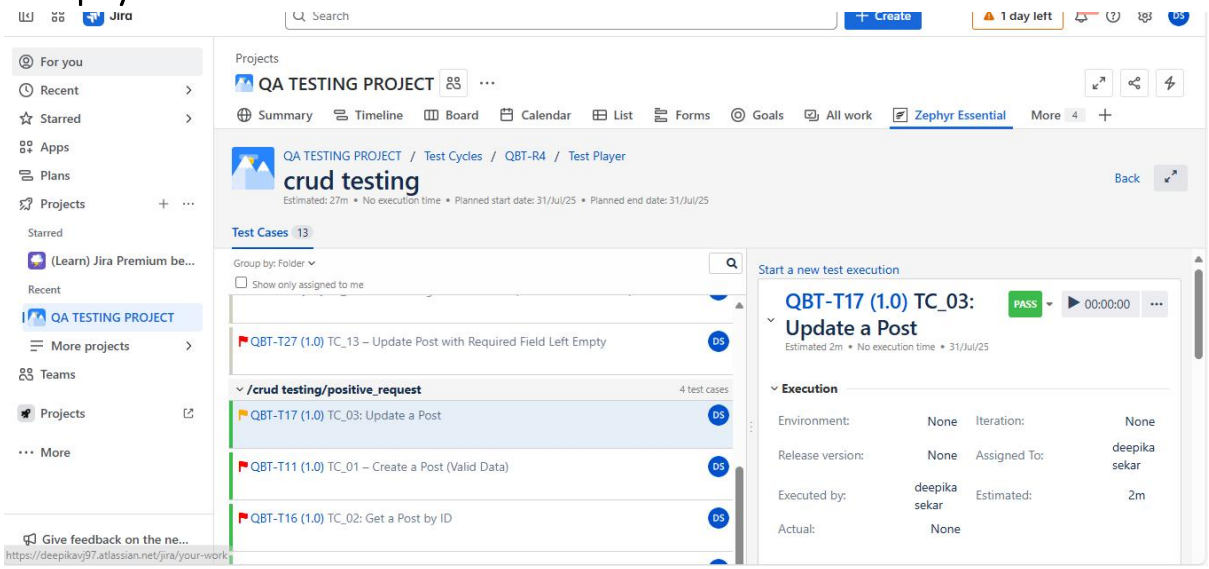


Figure 8: Zephyr Test Case Execution for TC_03 – Update Post

4. Postman Screenshot

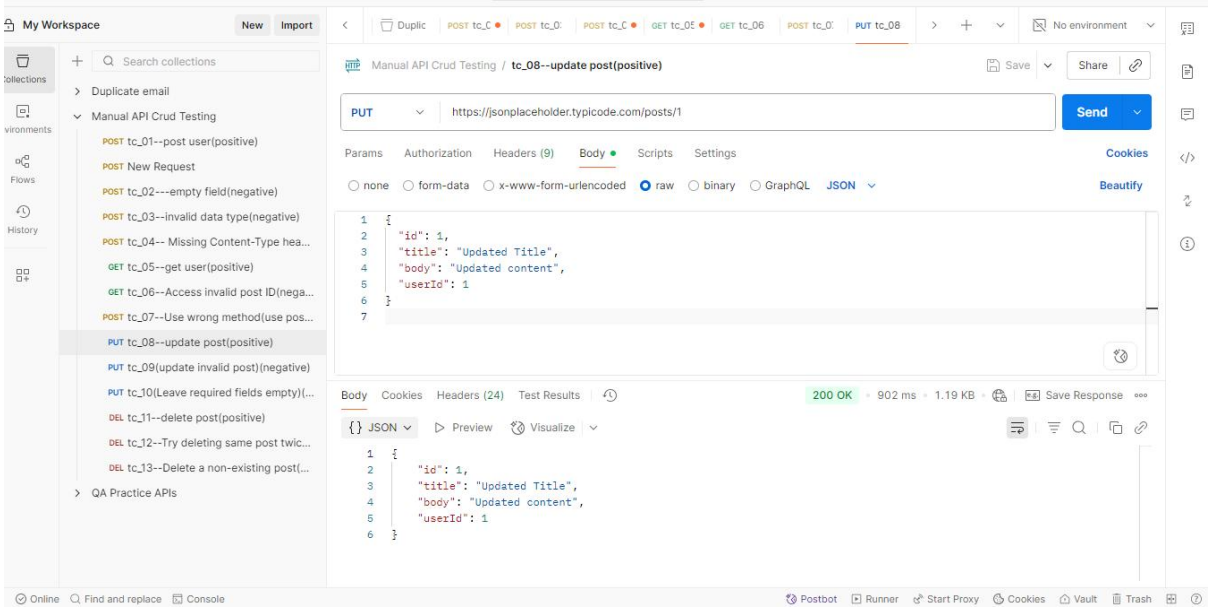


Figure 9: Postman Response for TC_03 – Updated Post Details

5. Database Validation Screenshot

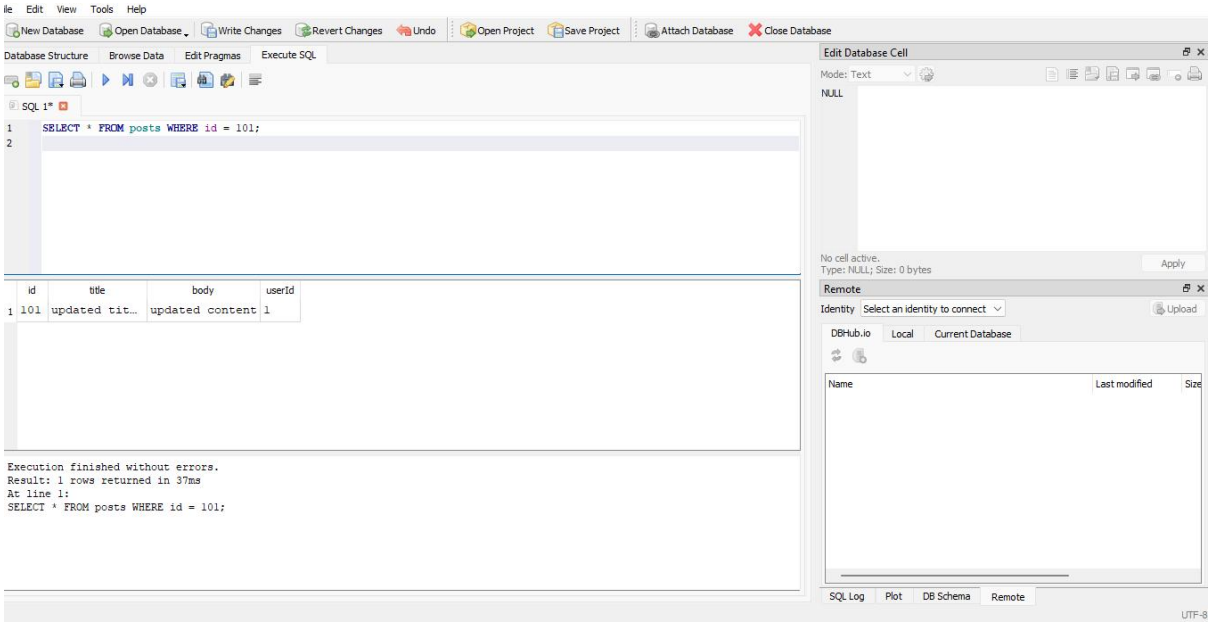


Figure 10: DB record after TC_03 PUT request – Updated Title and Body

6. Final Result

✅ **Test Passed**
The existing post was successfully updated via API.
Updated data was verified in both the API response and the database.
Zephyr execution marked as **PASS** ✅

---

1. Test Case Title

TC_04 – Delete a Post
Test case designed and executed using **Zephyr for Jira**.

2. Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send DELETE request to /posts/101	–	Status **200 OK** (or 204 No Content depending on API)
2	Validate the response status	–	Response status confirms successful deletion
3	Cross-check DB entry	–	Database no longer contains a record with `id = 101`

3. Zephyr Screenshot

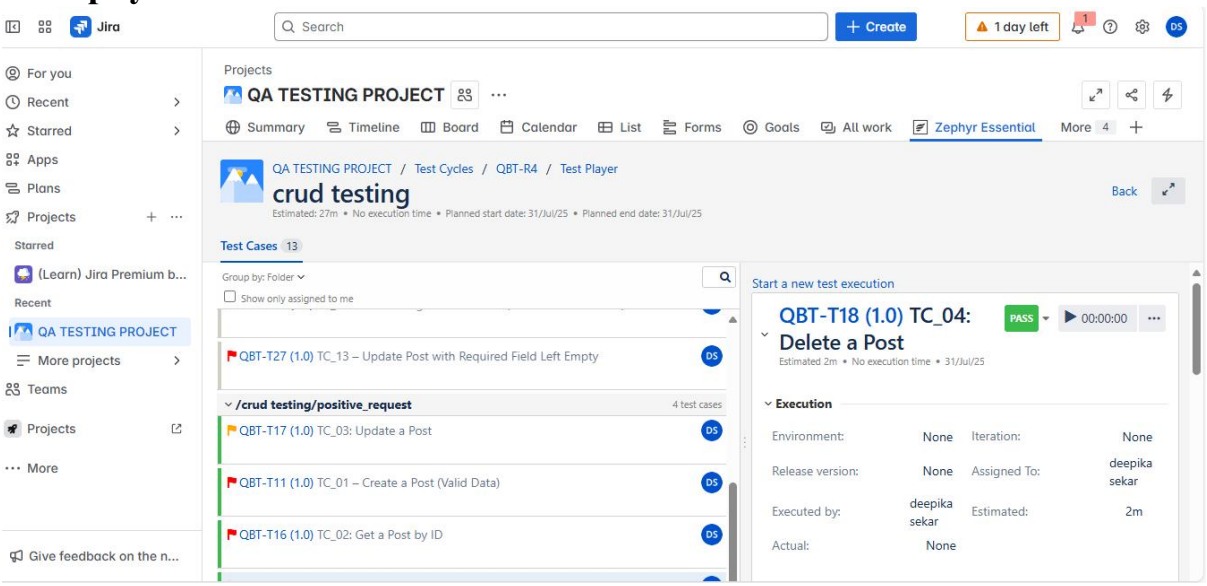


Figure 11: Zephyr Test Case Execution for TC_04 – Delete Post

4. Postman Screenshot

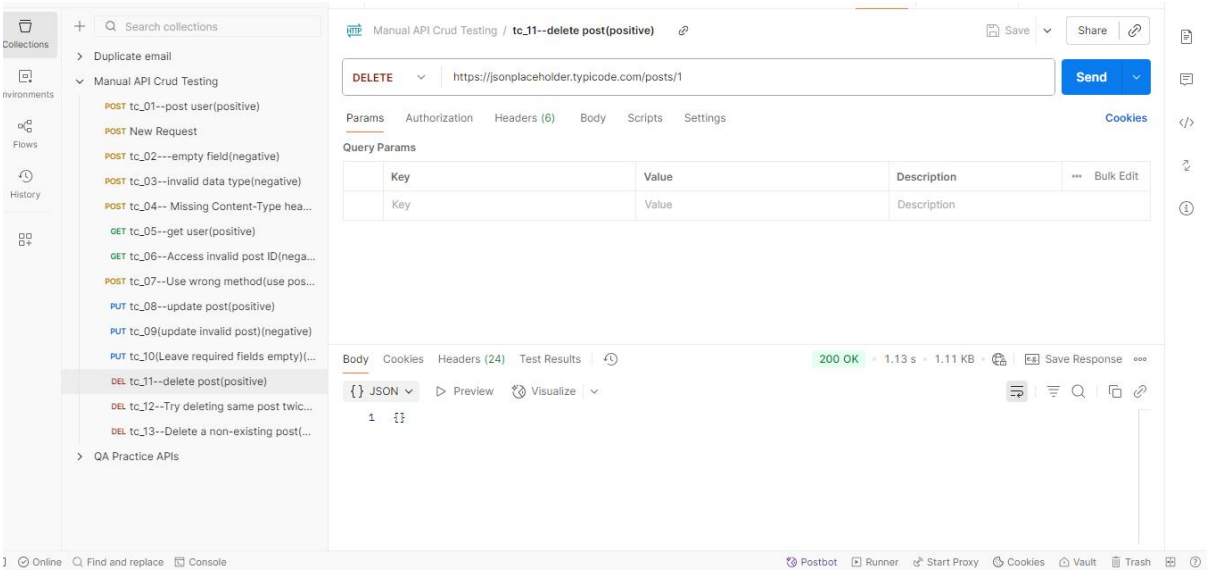


Figure 12: Postman Response for TC_04 – Post Deleted Successfully

5.Database Validation Screenshot

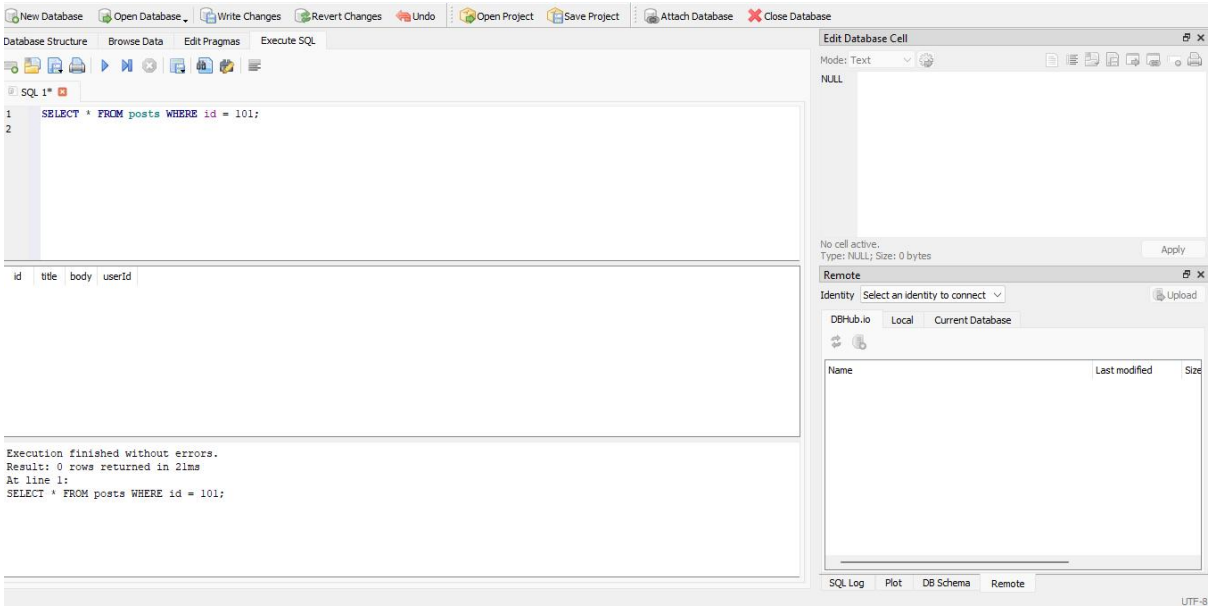


Figure 13: DB Validation for TC_04 – Post with ID 101 No Longer Exists

6. Final Result

✅ **Test Passed**
The existing post was successfully deleted via API.
Verified deletion in both API response and database.
Zephyr execution marked as PASS ✅

---

1. Test Case Title

TC_05 – Create Post with Missing Field
Test case designed and executed using **Zephyr for Jira**.

2. Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send `POST` request to `/posts`	`{ "body": "This is a dummy test post", "userId": 1 }`	API should reject request (400/422), OR accept with `null`/missing `title`
2	Validate the response	–	Response returns `id: 101`, but `title` field is missing
3	Cross-check DB entry	–	DB contains record with `id = 101`, `title` is `NULL` or not present

---

3. Zephyr Screenshot

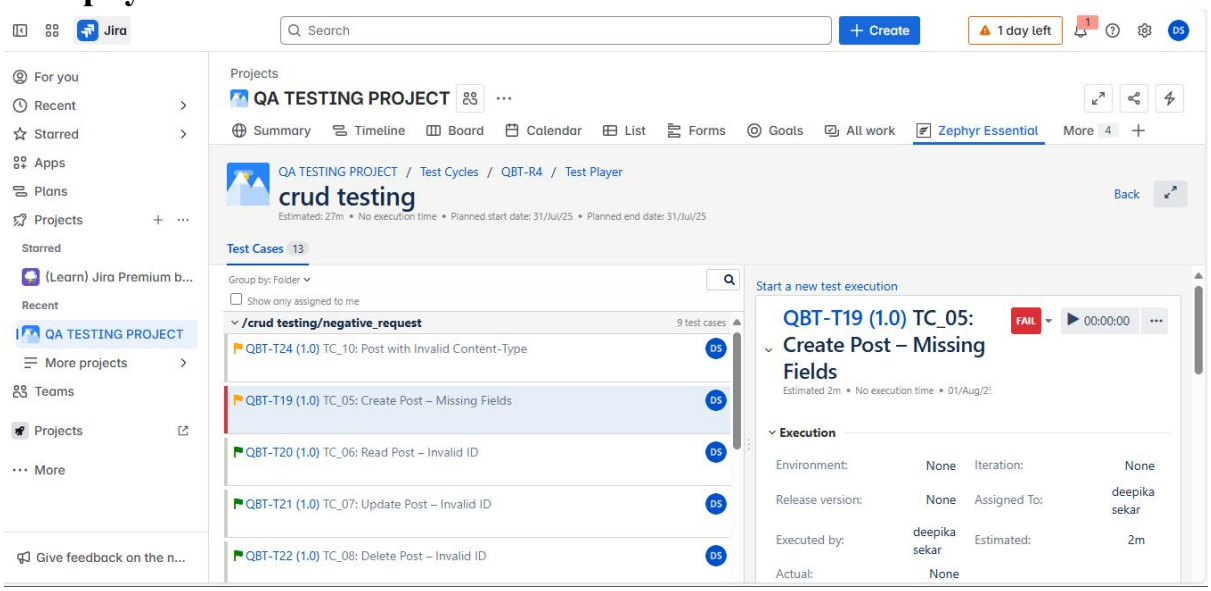


Figure 14: Zephyr Test Case Execution for TC_05 – Missing Title Field

4. Response Screenshot (Postman)

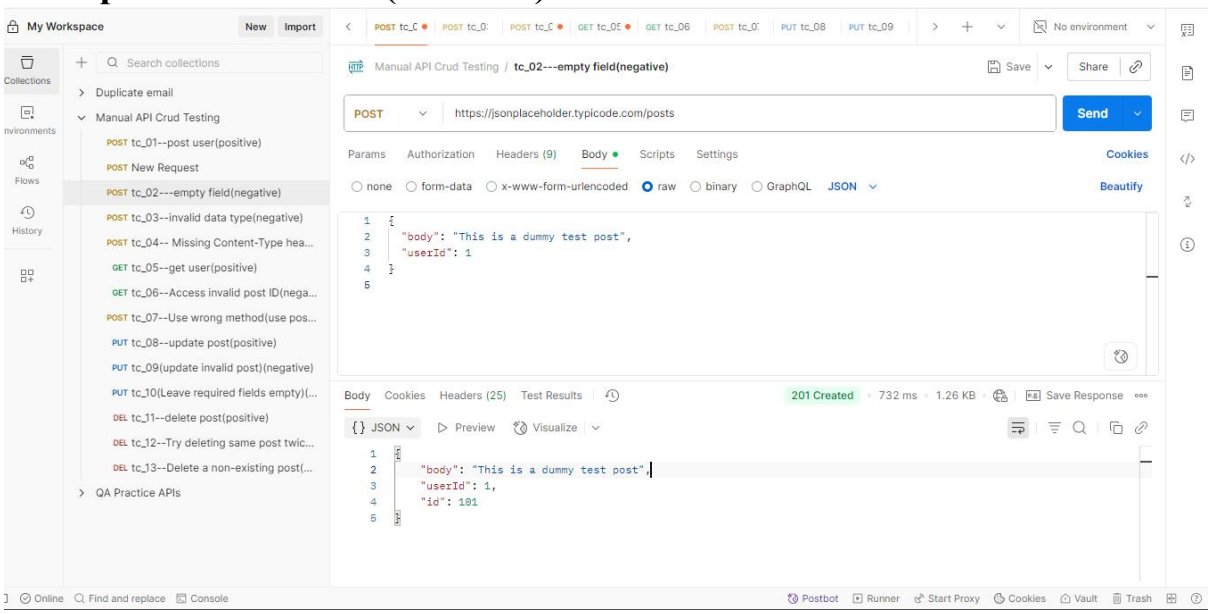


Figure 15: Postman Response – Title Field Missing, Still Accepted

5. Database Validation

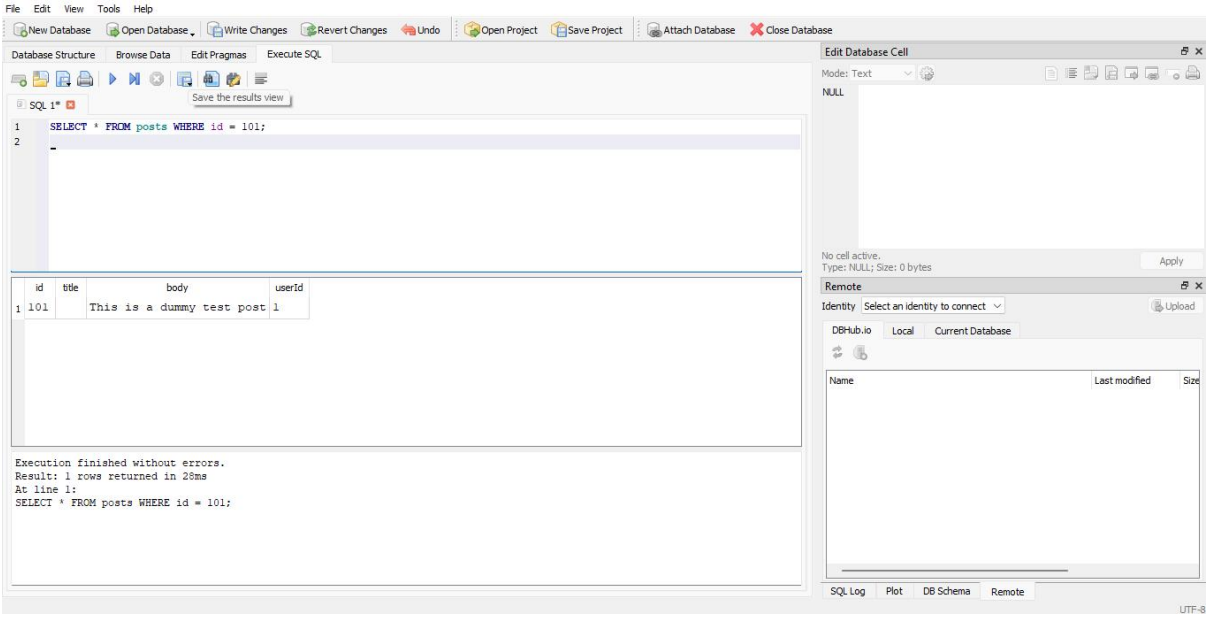


Figure 16: DB record with missing title (TC_05)

6. Final Result

✕ Test Failed (Expected Failure)
The API allowed creation of a post without a `title`. In real-world APIs, this should trigger a validation error (e.g., 400 Bad Request).

✓ verified this both via API response and DB record.
Zephyr execution marked as **FAIL,bug reported**

7.Bug Report – Linked to TC_05

Bug Title: API accepts POST request with missing `title` field
Bug ID: QBT-21
Reported in: JIRA
Linked Test Case: TC_05 – Create Post with Missing Title
Priority: Medium
Severity: High
Status: Resolved

Bug Summary

The API does not validate required fields. When sending a POST request without the `title` field, it still returns a `201 Created` response and inserts a record into the database with an empty `title`.

Attachments

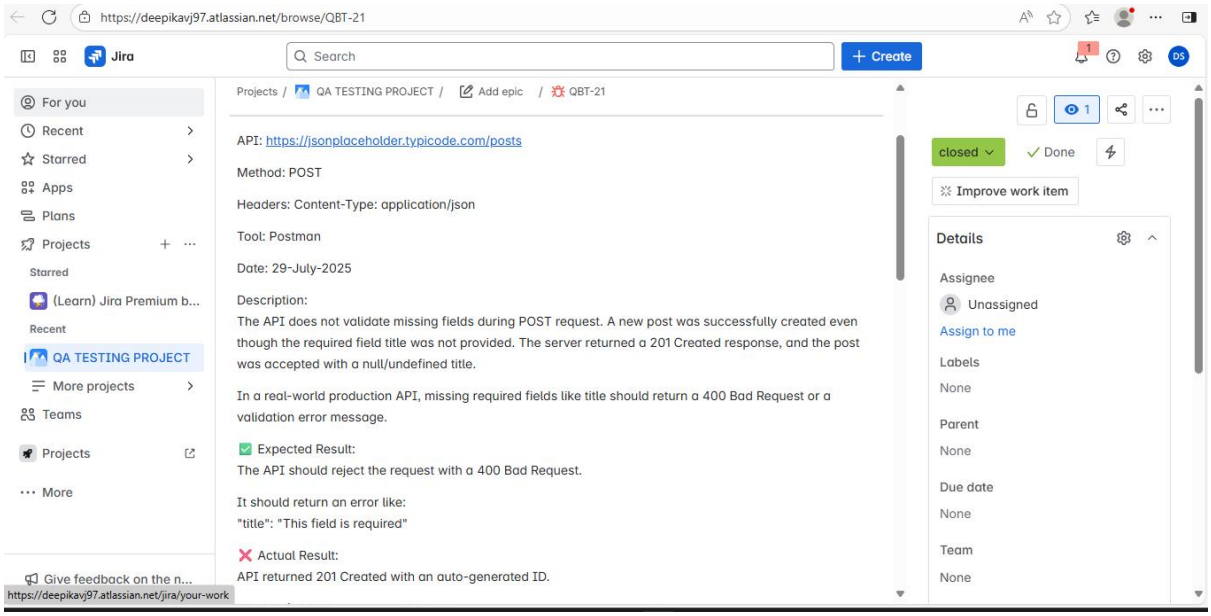


Figure 17: JIRA Bug Report Linked to TC_05

---

1. Test Case Title

TC_06 – Read Post- Invalid ID

Test case designed and executed using Zephyr for Jira.

2. Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send GET request to `/posts/9999`	Invalid ID (e.g., 9999 or non-existent ID)	Status **404 Not Found** or empty `{}` response indicating post does not exist
2	Validate the response	–	API should not return any data for an invalid ID
3	Check DB for corresponding entry	–	DB should not contain any record with `id = 9999`

3. Zephyr Screenshot

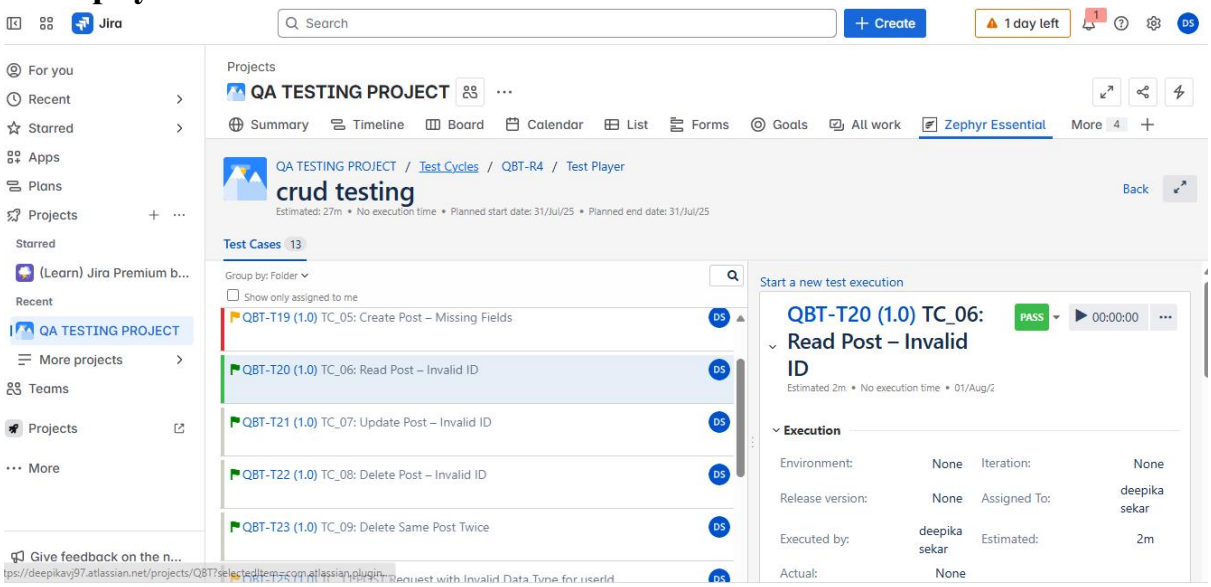


Figure18 – Zephyr Test Case Execution for TC_06 – Invalid Post ID

4. Postman Screenshot

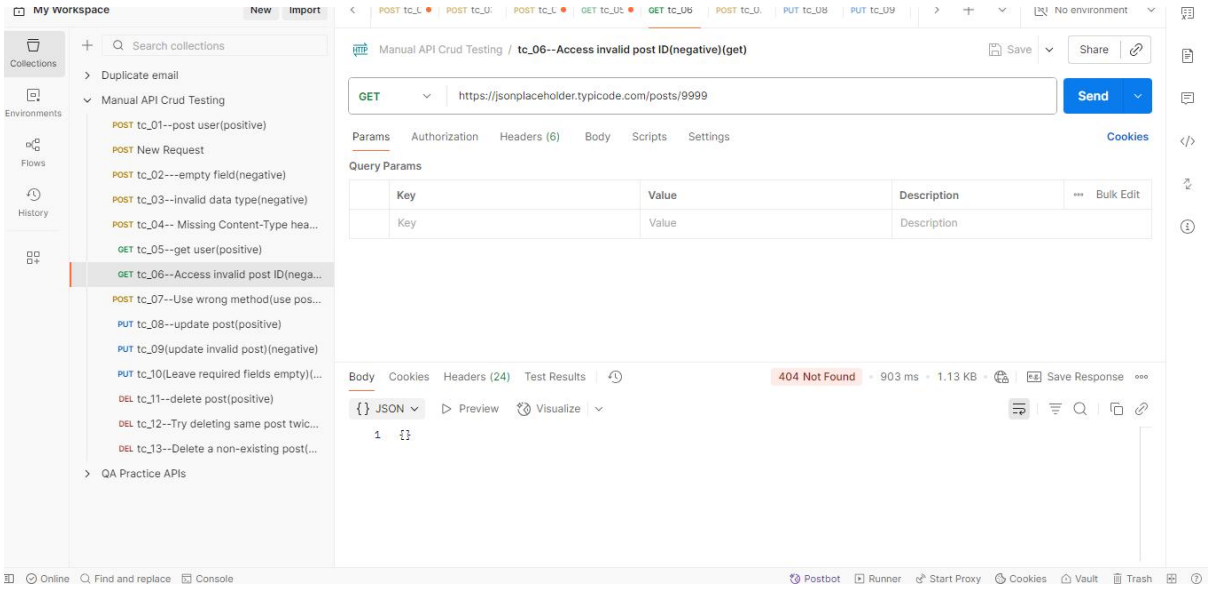


Figure 19 – Postman Response for TC_06 – 404 Not Found / Empty Response

5. Database Validation Screenshot

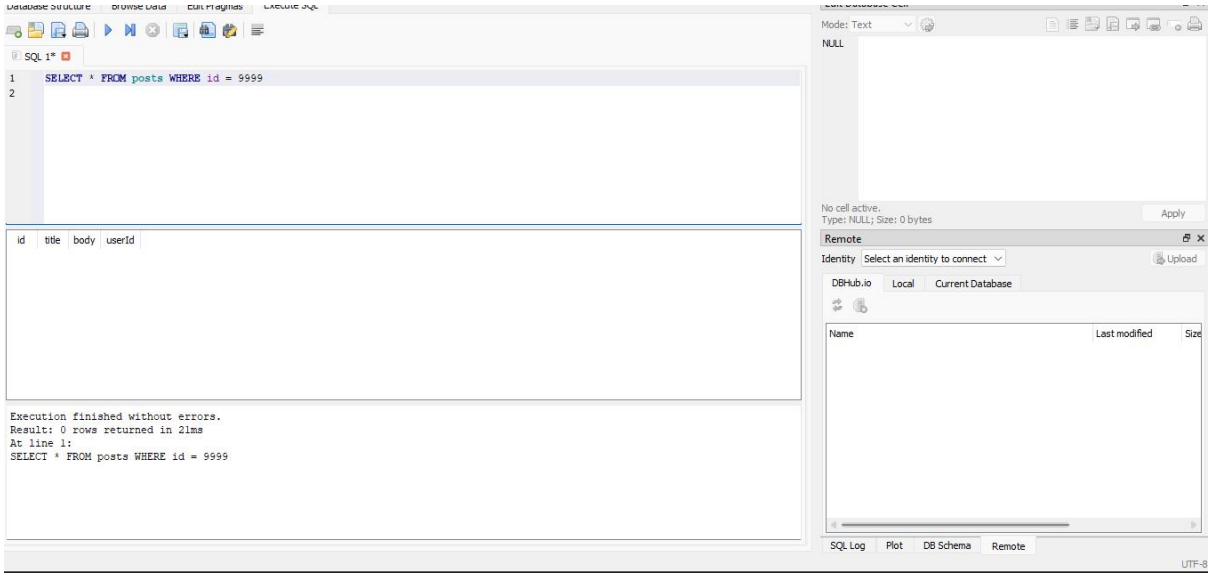


Figure 20 – DB Validation for TC_06 – No Record Found with ID 9999

6. Final Result

✓ **Test Passed**
The system handled the invalid post ID correctly. The API returned 404 error, and the database had no matching entry. Zephyr test case marked as **PASS** ✓.

---

1. Test Case Title

TC_07– Update Post-Invalid ID
Test case designed and executed using Zephyr for Jira.

2. Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send PUT request to `/posts/9999`	`{ "id": 9999, "title": "Test Invalid Update", "body": "Dummy", "userId": 1 }`	API should return **404 Not Found** or **error message**
2	Validate the response	–	API should indicate **record not found** / update not possible
3	Check DB for record	–	No DB record with `id = 9999` should exist

❌ Actual Result

- API returned **500 Internal Server Error** instead of a meaningful response like 404 or validation message.

3.Postman Screenshot

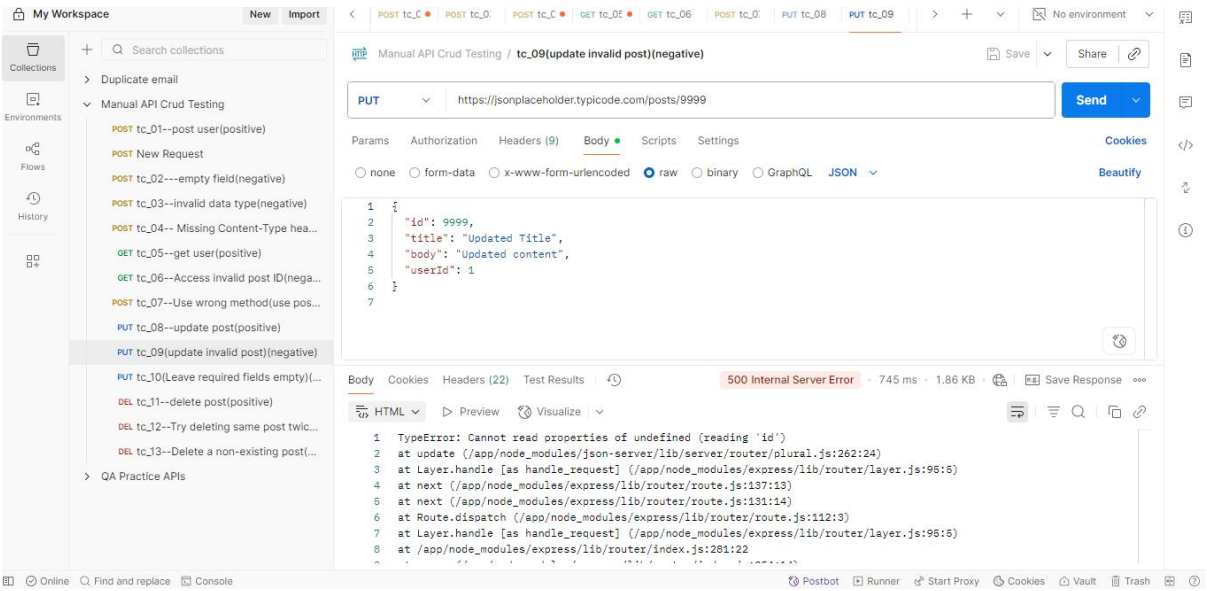


Figure 21 – 500 Error for TC_07– Invalid PUT Request

4.Zephyr Screenshot

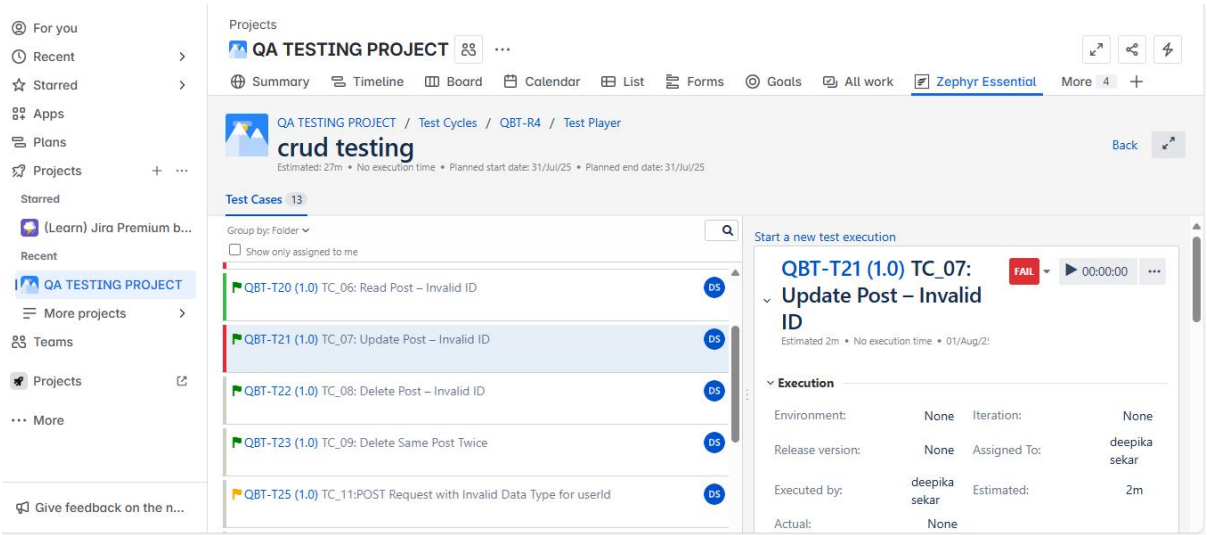


Figure 22 – Zephyr Execution: TC_07 – Update with Invalid ID → FAIL

5.DB Screenshot

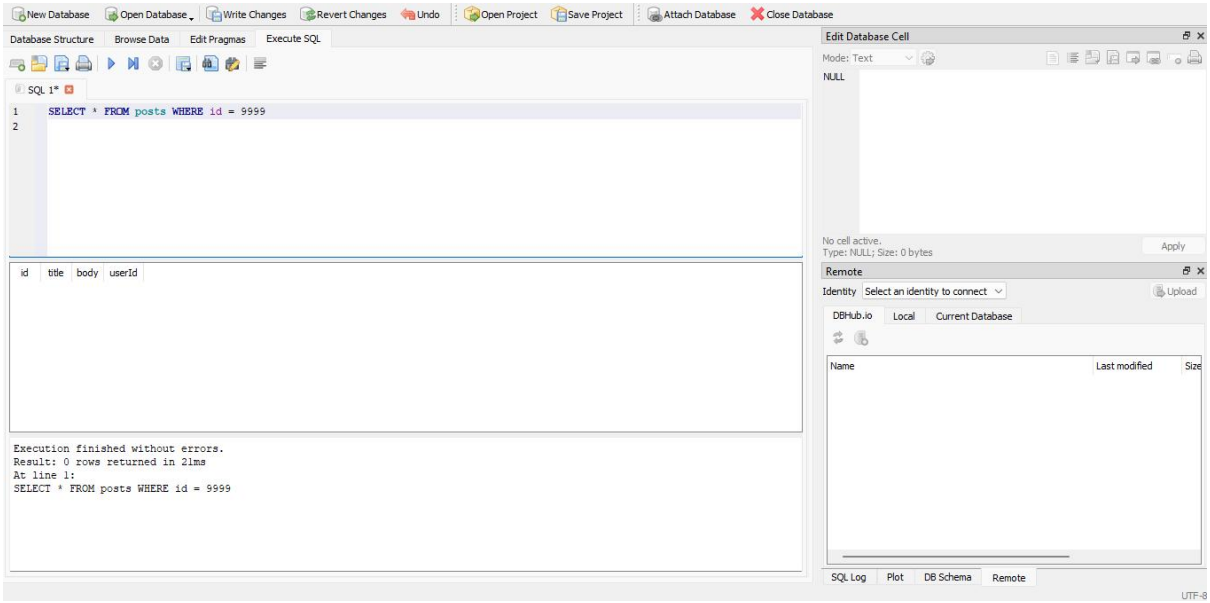


Figure 23 – No Record Found with ID 9999 in DB

6. Final Result

✕ Test Failed
API crashed with 500 error for invalid ID update.
Reported as a bug in Jira and Zephyr marked as **FAIL ✕**.

7.Bug Report – Linked to TC_06

- **Bug Title**: API returns 500 Internal Server Error for invalid post ID
- **Bug ID**: *QBT-22*
- **Reported in**: JIRA
- **Linked Test Case**: TC_06 – Read Post- Invalid ID
- **Priority**: High
- **Severity**: Major
- **Status**: Resolved

Bug Summary

When sending a **GET request to `/posts/9999`**(non-existent post), the API should return a **404 Not Found** error.
Instead, it throws a **500 Internal Server Error**, which indicates a server-side failure instead of proper validation or handling of missing records.

- **Expected**: 404 Not Found with a clear message (e.g., “Post not found”)
- **Actual**: 500 Internal Server Error

This may affect user experience and indicates that backend error handling for invalid IDs is missing.

Attachments

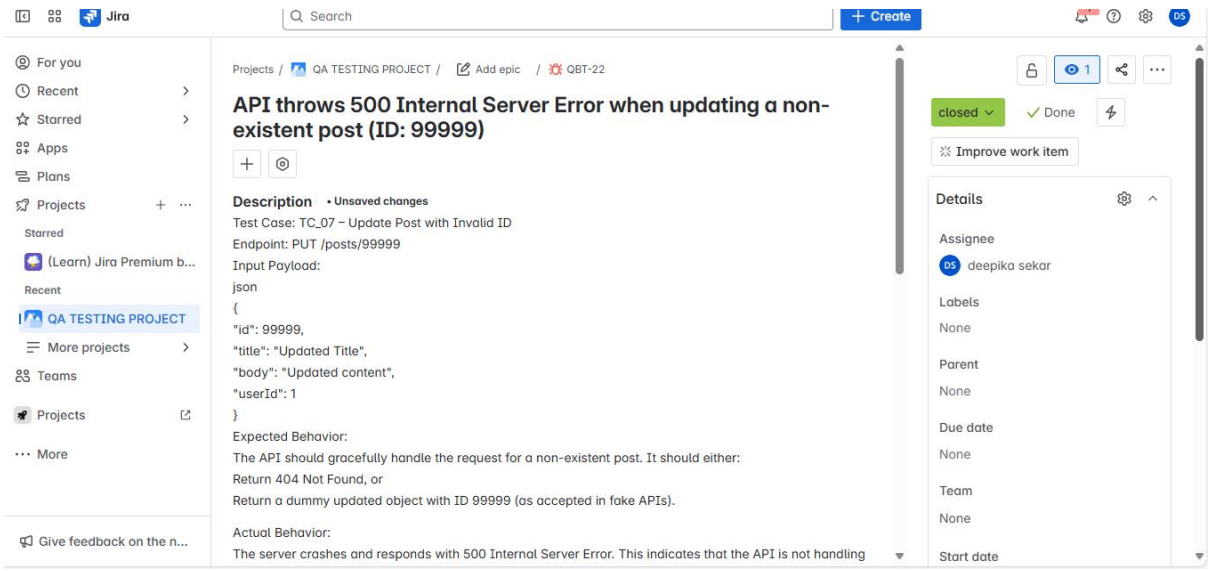


Figure 24: JIRA Bug Report Linked to TC_06 – 500 Error for Invalid ID

---

1. Test Case Title

TC_08 – Delete Post - Invalid ID

Test case designed and executed using Zephyr for Jira.

2. Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send DELETE request to `/posts/99999`	ID = 99999 (non-existent)	❌ Expected: 404 Not Found (API should reject invalid ID) ✅ Actual: Status 200 OK
2	Validate the response	–	❌ No error message returned; response implies successful deletion
3	Cross-check DB entry	–	✅ DB has no matching record for ID 99999 – nothing deleted (as expected)

3.Zephyr Screenshot

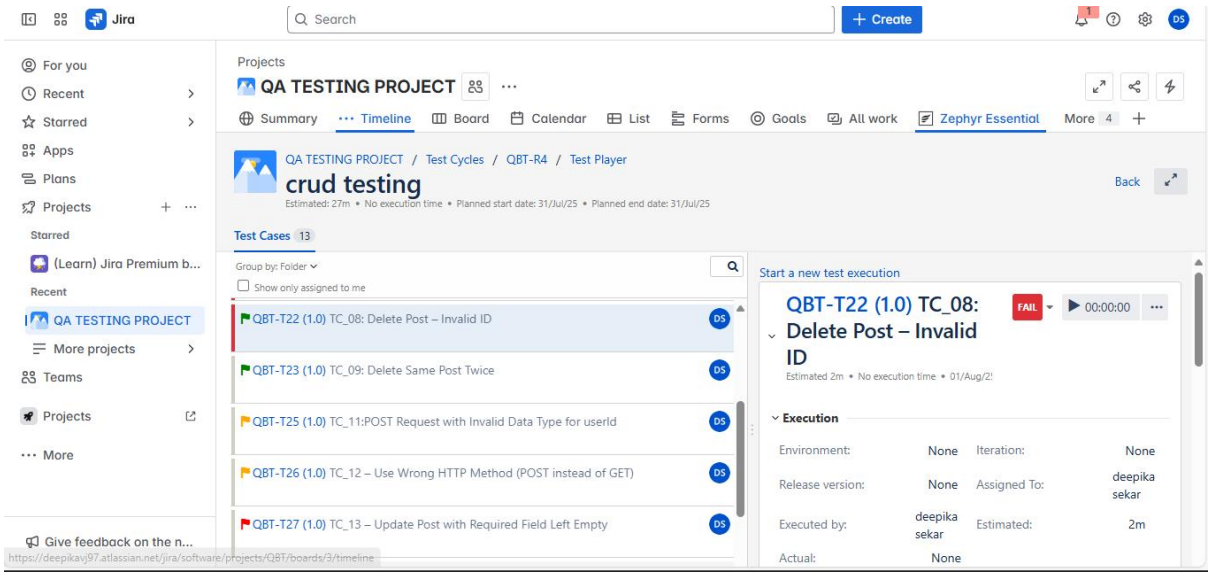


Figure 25: Zephyr Execution – TC_08 Delete Invalid ID (API returned 200 OK instead of 404 Not Found)

4. Postman Screenshot

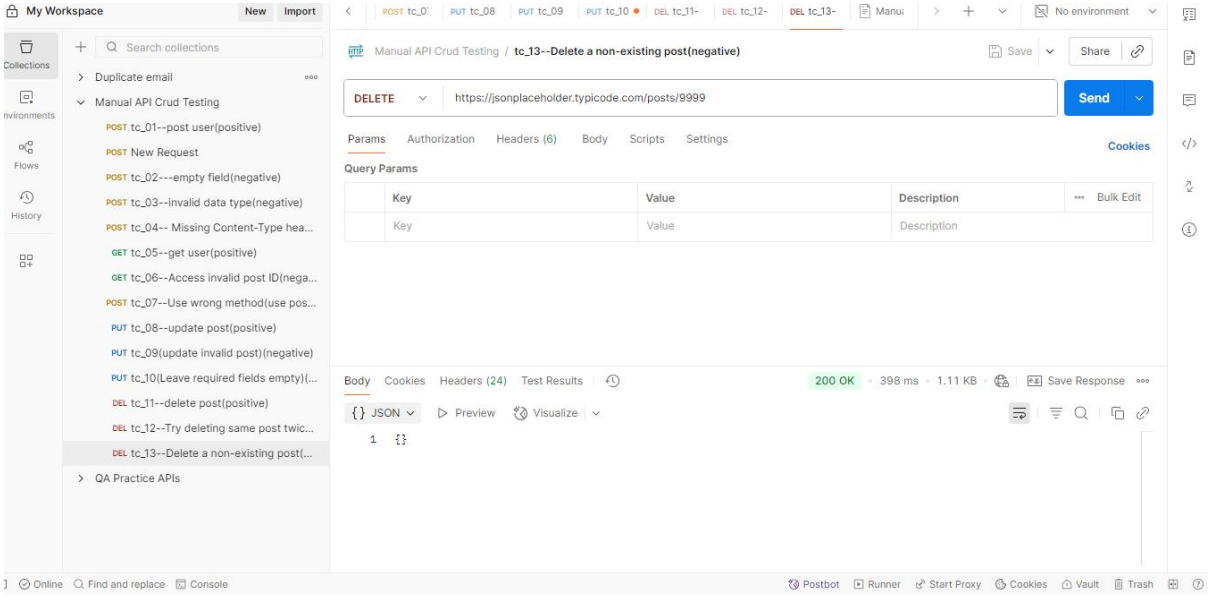


Figure 26: Postman – 200 OK Returned for Invalid Delete ID (TC_08)

5. Database Validation Screenshot

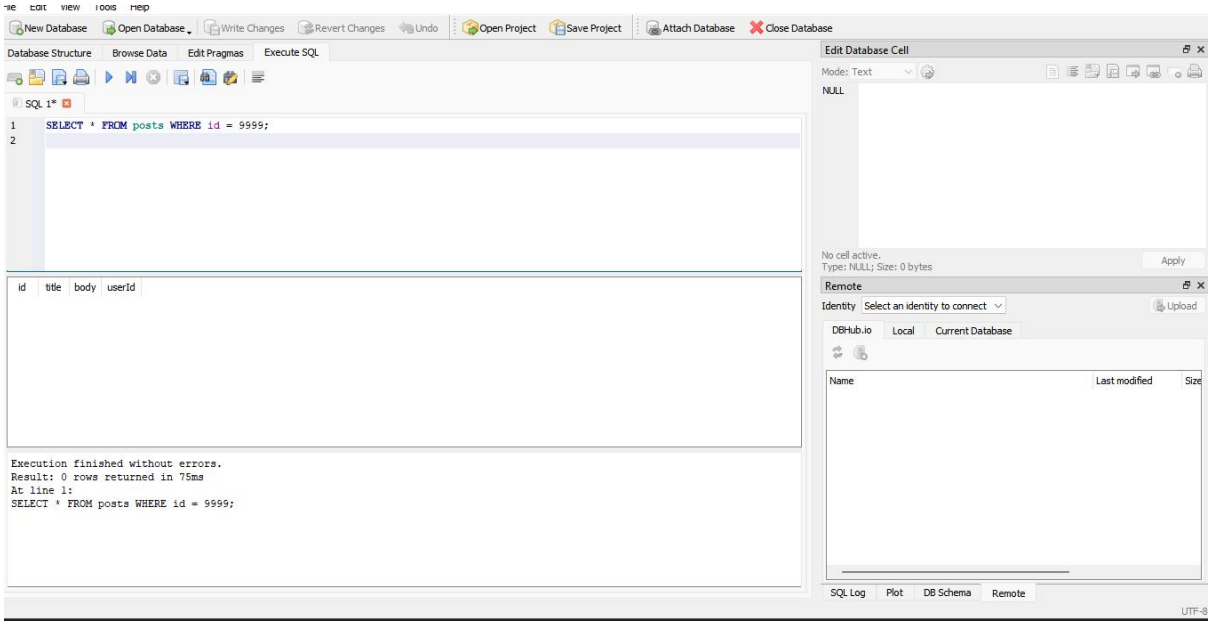


Figure 27: DB Shows No Record for ID 99999 (TC_08)

6. Final Result

✗ Test Failed (as expected)
The API returned **200 OK** even though the post ID didn't exist.
This is misleading and violates proper REST API behavior – should return `404 Not Found`.

✓ **Test execution completed successfully** – validated through response and DB.
Zephyr marked as **FAIL**, and **bug reported in JIRA**.

7.Bug Report – Linked to TC_08

Bug Title: API returns 200 OK for DELETE request with non-existent ID
Bug ID: QBT-23
Reported in: JIRA
Linked Test Case: TC_08 – Delete Post-Invalid ID
Priority: Medium
Severity: Major
Status: Resolved

🔎 Bug Summary
When sending a `DELETE` request to `/posts/99999` (invalid ID), the API still returns a **200 OK** response, implying successful deletion even though the record does not exist.
The expected behavior is a **404 Not Found**, ensuring clarity in response messaging.

Attachments

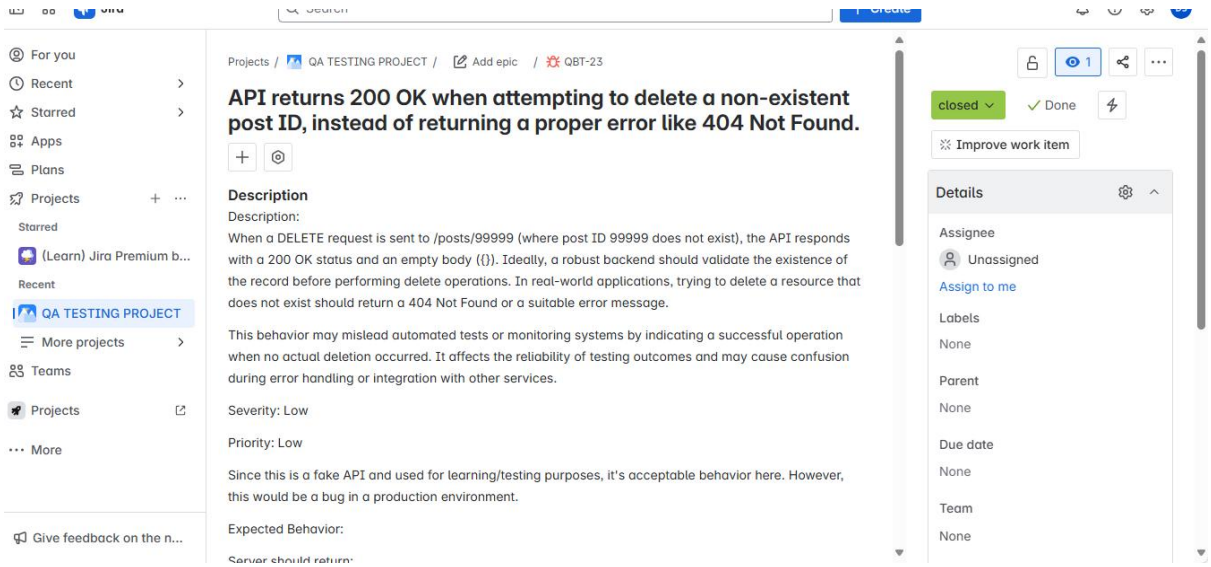


Figure 28: JIRA Bug Report Linked to TC_08

.

TC_09 – Delete Same Post Twice (ID: 101)

Test case designed and executed using **Zephyr for Jira**.

2. Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send DELETE request to `/posts/101`	–	✓ **Expected**: Status **200 OK**, Post with ID=101 should be deleted
2	Send DELETE request to `/posts/101` again	–	✗ **Expected**: Status **404 Not Found** (as post no longer exists) ✓ **Actual**: Status **200 OK**

3	Validate via database	`SELECT * FROM posts WHERE id=101;`	✓ DB shows **no record with ID=101** (deleted already)

3.Zephyr Screenshot

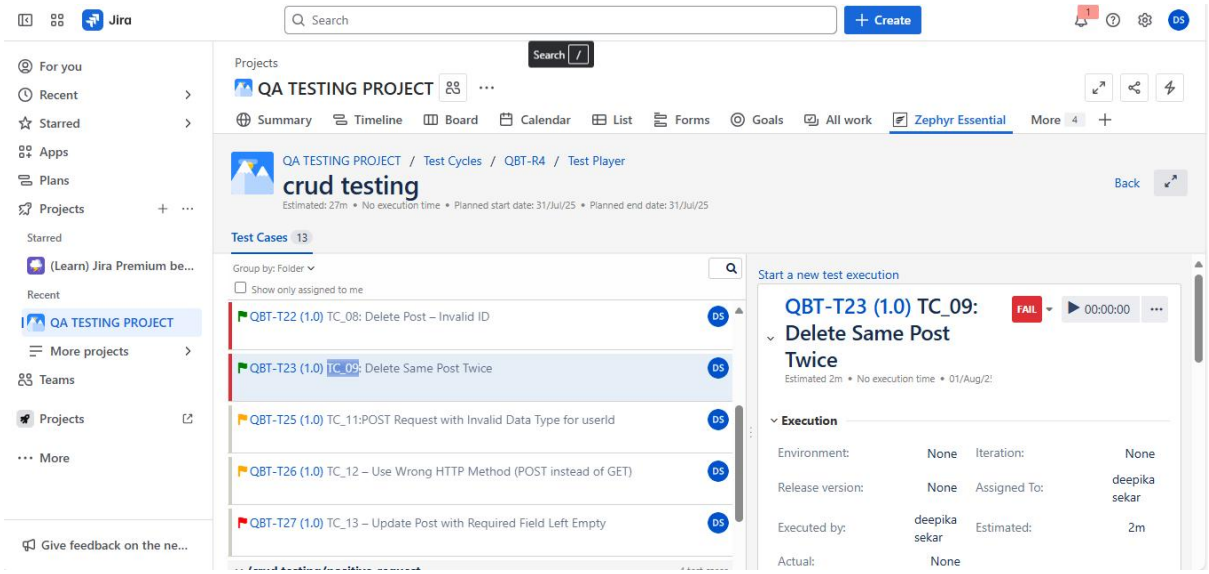
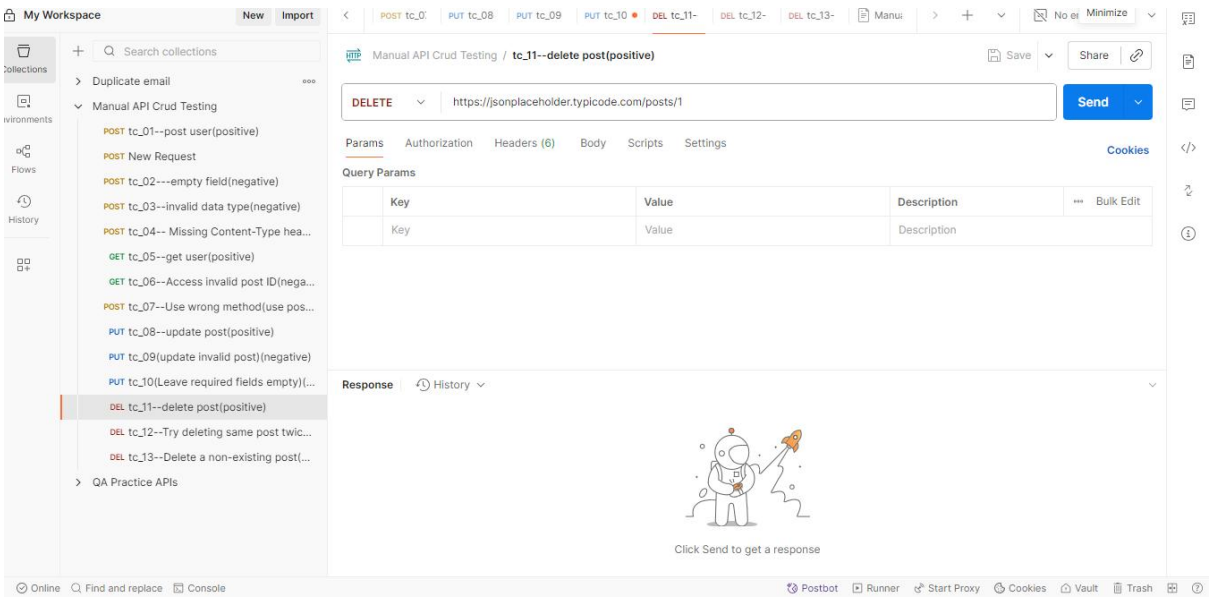


Figure 29: Zephyr Execution – TC_09 Double Delete (Second delete of same post ID 101 also returned 200 OK)

4. Postman Screenshot



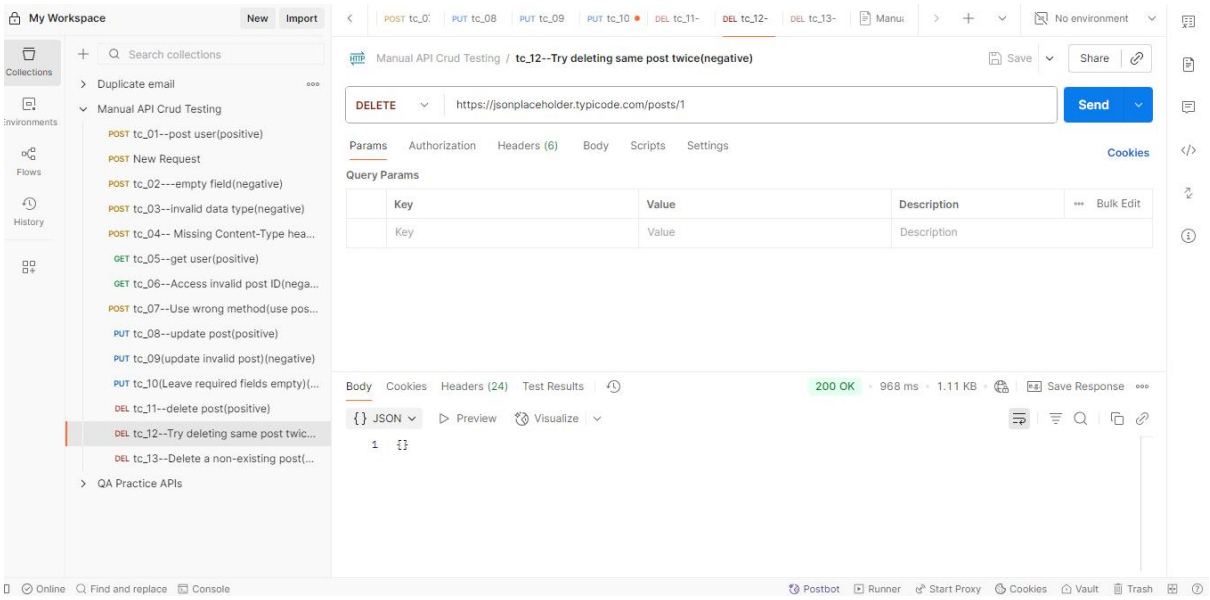


Figure 30: Postman – Second DELETE on same ID still returns 200 OK (TC_09)

---

5. Database Validation Screenshot

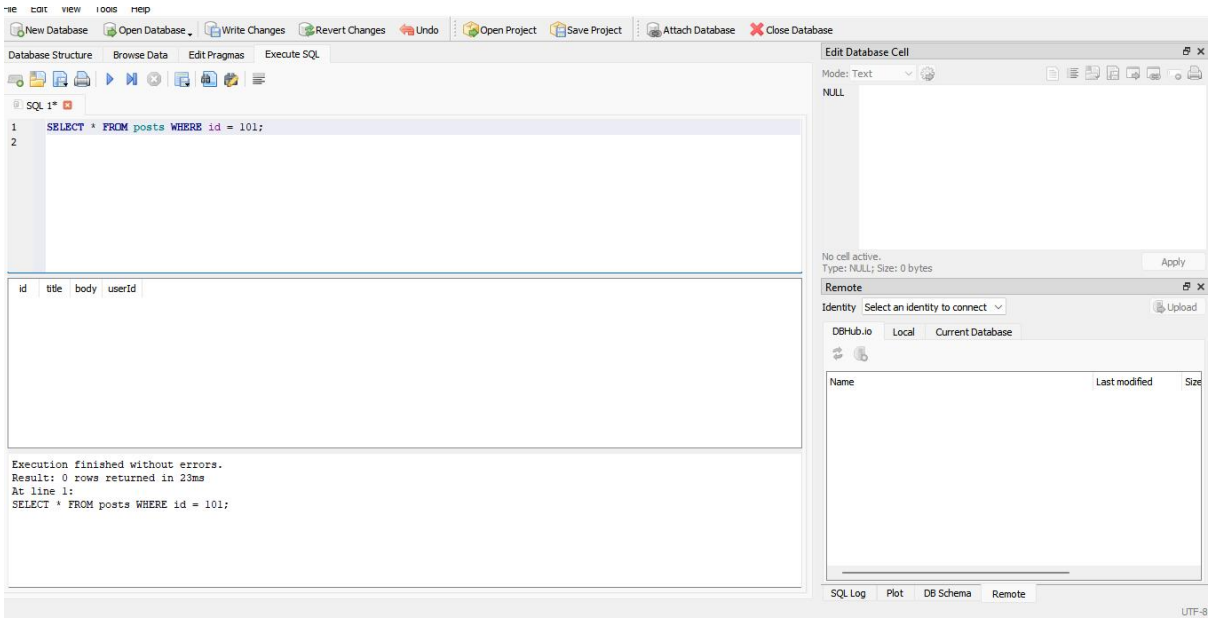


Figure 31: DB shows post ID 101 no longer exists after first DELETE (TC_09)

6. Final Result

❌ Test Failed (as expected)

- API should not return **200 OK** for a second DELETE on the same ID.
- It should ideally return **404 Not Found** or another error indicating the post doesn't exist.

✅ Test execution verified via Postman response and DB check.
Zephyr marked as **FAIL**, **bug reported** in Jira.

7.Bug Report – Linked to TC_09

Bug Title: API allows multiple DELETE operations on same resource – returns 200 OK even after deletion
Bug ID: QBT-24
Reported in: JIRA
Linked Test Case: TC_09 – Delete Same Post Twice
Priority: Medium
Severity: Moderate
Status: Resolved

🔎 Bug Summary
After deleting a post (e.g., ID = 101), sending a second DELETE request to the same ID still returns **200 OK**. Ideally, once deleted, a second delete should return **404 Not Found** to reflect that the resource no longer exists.

📎 Attachments

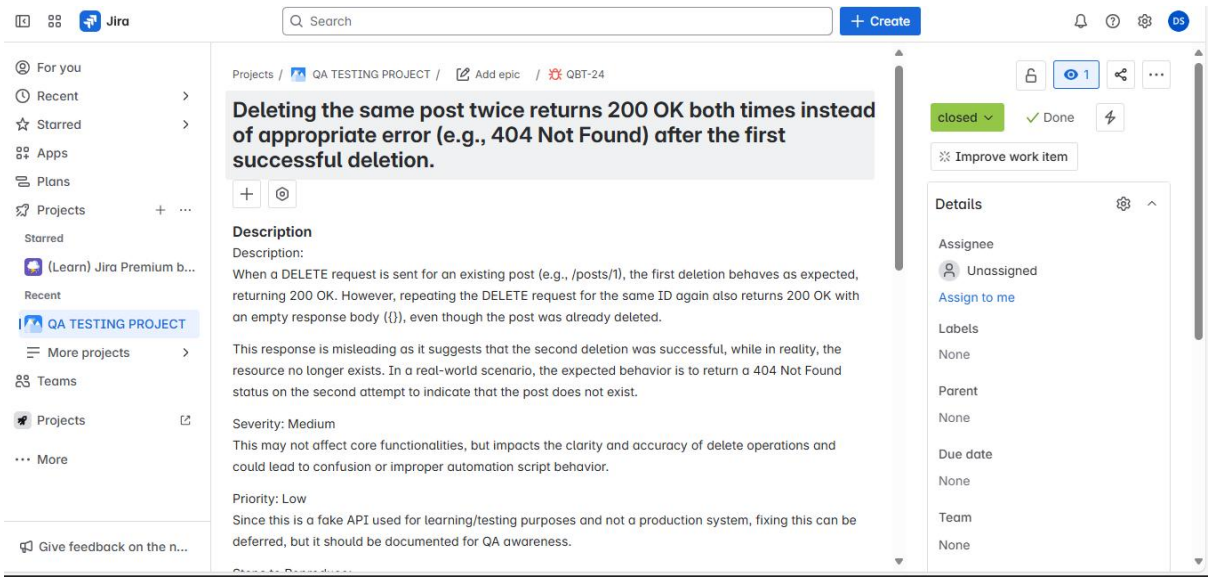


Figure 32: JIRA Bug Report Linked to TC_09

1. Test Case Title

TC_10 – Post with Invalid Content-Type

2. Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send POST request to `/posts` **without** `Content-Type: application/json`	`{ "title": "Test", "body": "Invalid header", "userId": 1 }`	API should return **415 Unsupported Media Type** or **400 Bad Request**
2	Observe server response	No Content-Type	API should **reject** the request
3	Check response body and status code	–	Status should **not** be 201; error message expected
4	DB/GET validation	–	No new post should be created with id=101

3.Zephyr screenshot

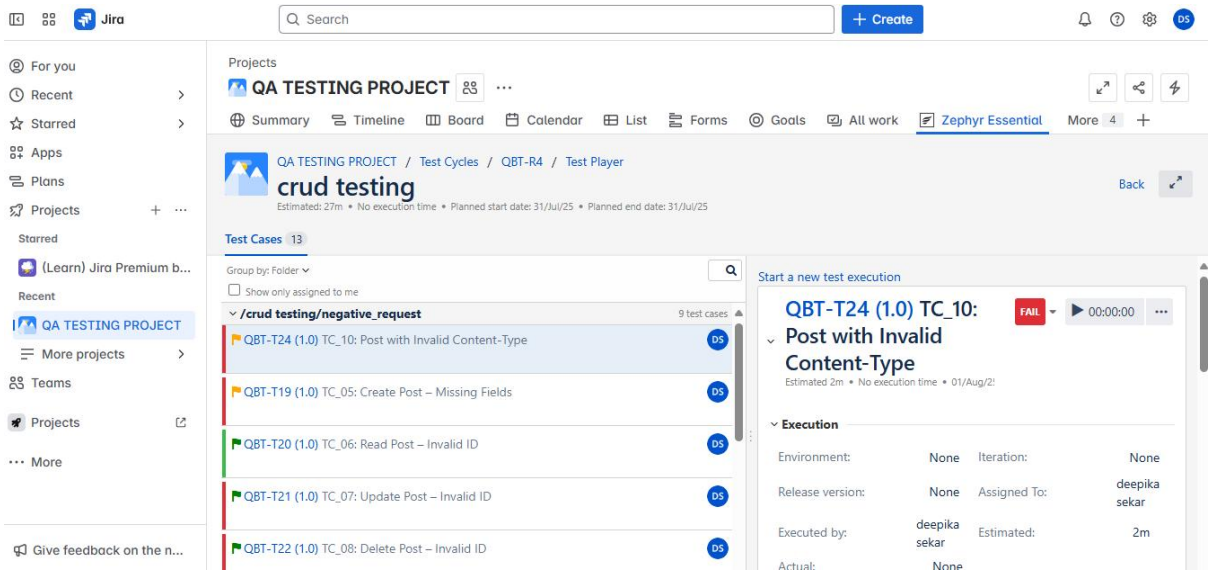


Figure 33: Zephyr Test Case Execution for TC_10 – Invalid Content-Type POST (Marked FAIL)

4. Postman Screenshot

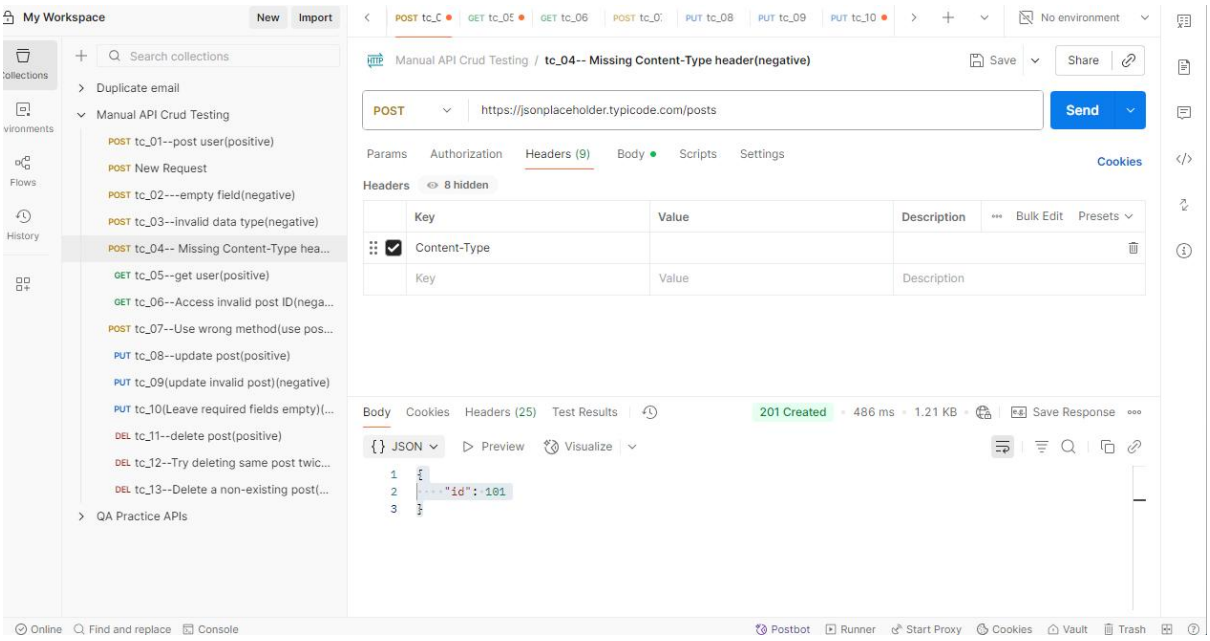


Figure 34: Postman Response for TC_10 – 201 Created Without Content-Type Header

5. Database Validation Screenshot

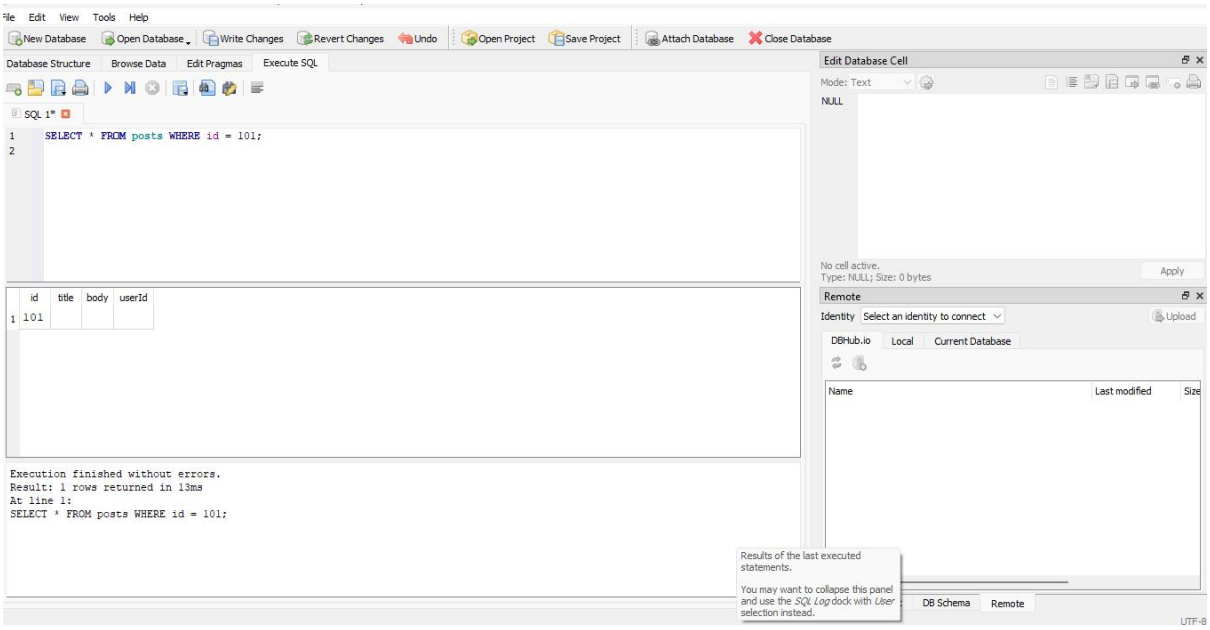


Figure 35: DB Entry for TC_10 – Unvalidated Post Inserted

6. Final Result

✗ Test Failed

The API accepted a POST request without the proper `Content-Type` header. It returned a `201 Created` response with an ID, which violates header validation rules.

7. Bug Report – Linked to TC_10

- **Bug Title:** API accepts POST request without valid Content-Type header
- **Linked Test Case:** TC_10 – POST with Invalid Content-Type
- **Reported In:** JIRA
- **Bug ID:** BUG_TC_10_001 *(example ID, use your actual Jira ID)*
- **Priority:** Medium
- **Severity:** Major
- **Status:** Open

Bug Summary:

The API incorrectly returns `201 Created` and accepts data even when the `Content-Type: application/json` header is **missing**. This can lead to data corruption and misuse.

Attachments:

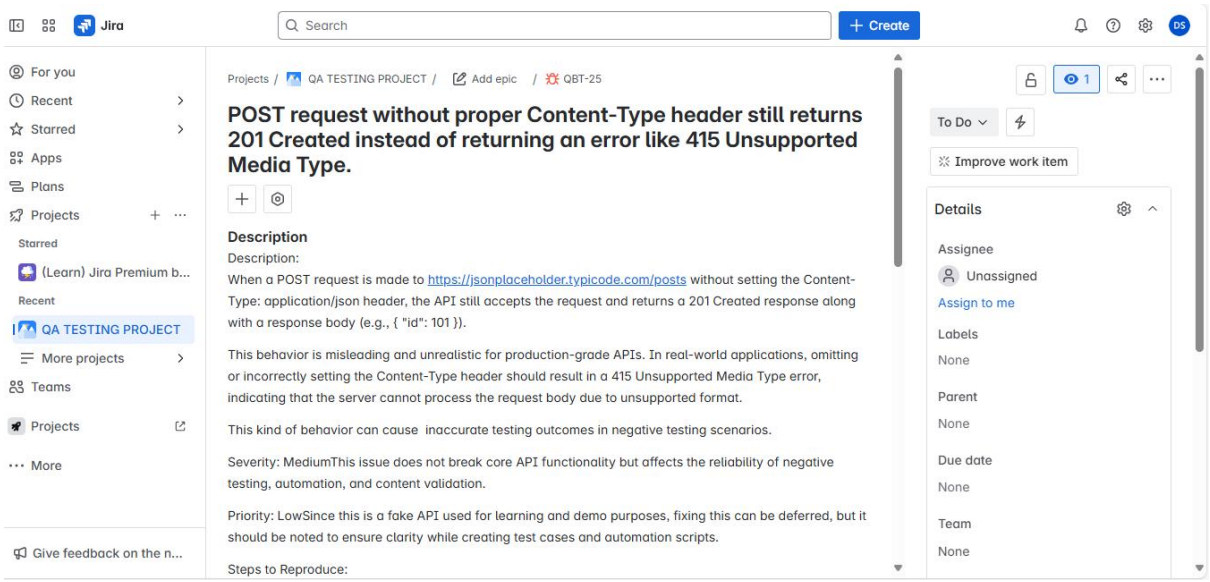


Figure 36: JIRA Bug Report for TC_10 – API accepts POST with invalid Content-Type

---

1. Test Case Title

TC_11 – Create a Post with Invalid Data Type for `userId`

Test case designed and executed using **Zephyr for Jira**.

2. Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send POST request to `/posts` with `userId` as a string instead of a number	`{ "title": "Invalid userId", "body": "Testing data type", "userId": "abc" }`	API should reject the request with 400 Bad Request or validation error
2	Observe server response	`userId: "abc"` (string)	Status should not be 201 Created
3	Check response body and status code	–	Error message indicating type mismatch
4	DB/GET validation	GET `/posts/101`	No new post should be inserted with incorrect data

3. Zephyr Screenshot

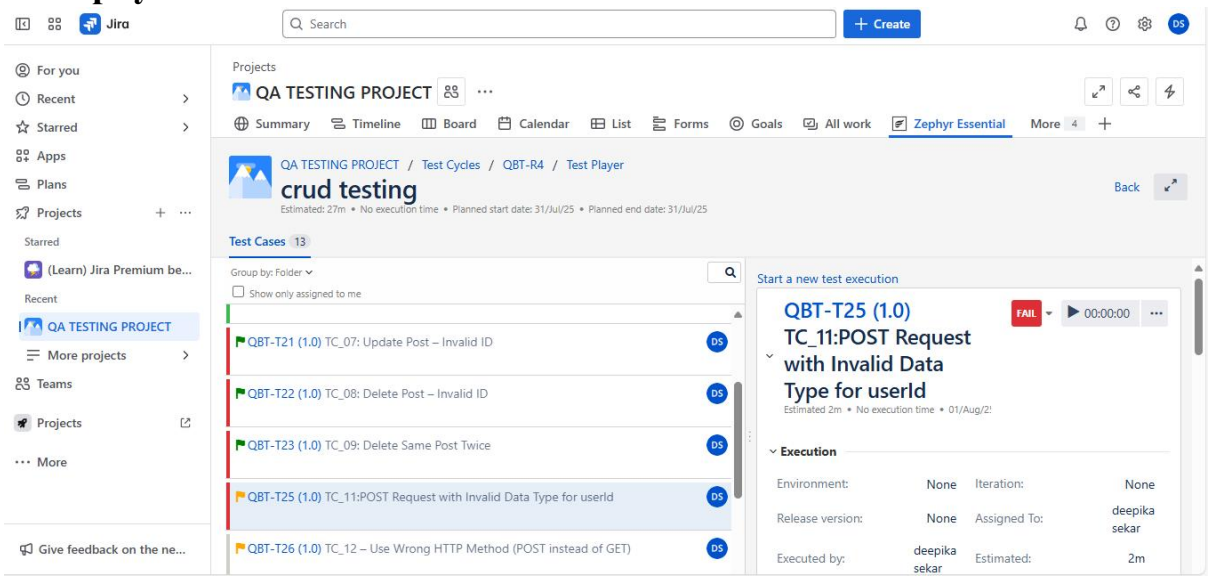


Figure 37: Zephyr Test Case Execution for TC_11 – Invalid userId Data Type (Marked FAIL)

4. Postman Screenshot

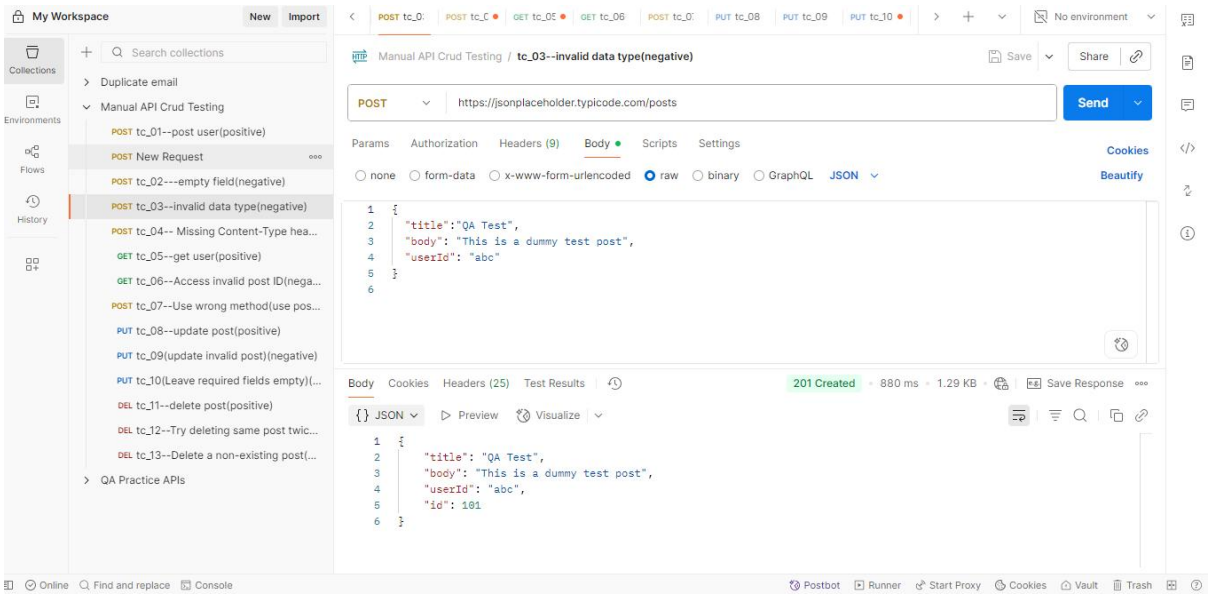


Figure 38: Postman Response for TC_11 – API Accepted String as userId (201 Created)

5. Database Validation Screenshot

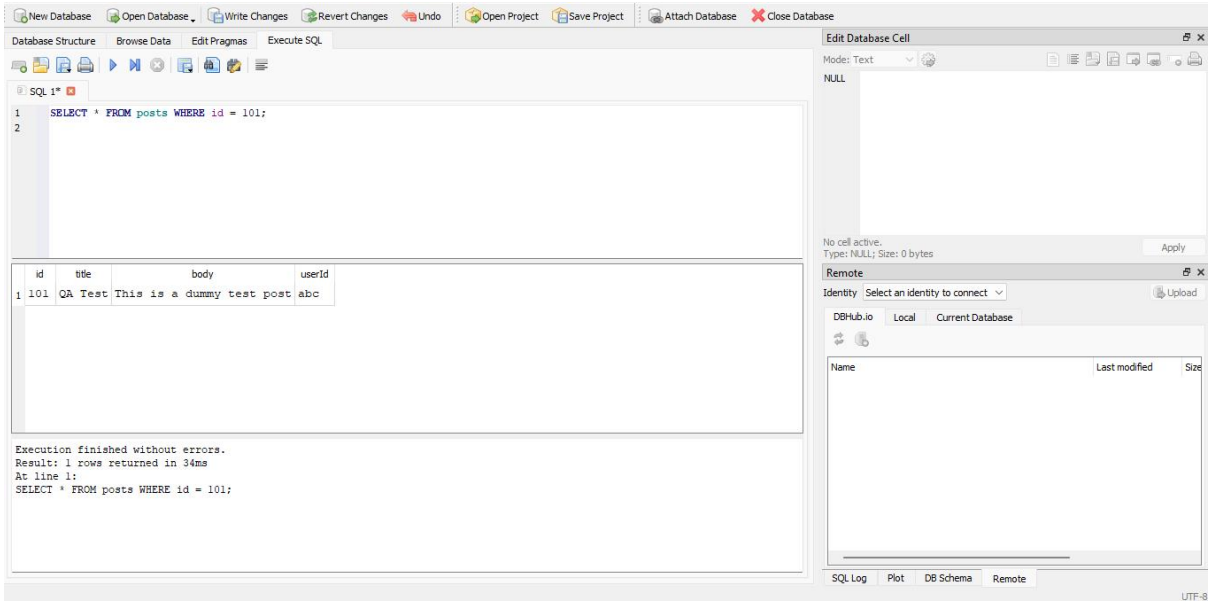


Figure 39: DB Entry for TC_11 – Post Inserted with String userId (Invalid)

6.Final Result

✗ Test Failed

The API accepted a `POST` request where `userId` was a string instead of an integer.
It returned a **201 Created** response and stored the post, which **violates proper input validation rules**.

7.Bug Report – Linked to TC_11

Bug Title: API accepts POST request with invalid data type for `userId`
Bug ID: `QBT-26`
Reported in: JIRA
Linked Test Case: `TC_11 – Create Post with Invalid userId (String Instead of Integer)`
Priority: Medium
Severity: Major
Status: Resolved

Bug Summary

The API does not validate input data types properly. When a POST request is sent with `userId` as a **string** (`"abc"` instead of an integer), the server still responds with **201 Created**, and the record is inserted into the database. This indicates a lack of input data type validation on the server side.

✅ Expected Behavior

The API should return **400 Bad Request** or a validation error when the data type of `userId` is incorrect (string instead of integer).

❌ Actual Behavior

The API responds with `201 Created`, and a new post is created with `userId = "abc"` in the database.

📎 Attachments

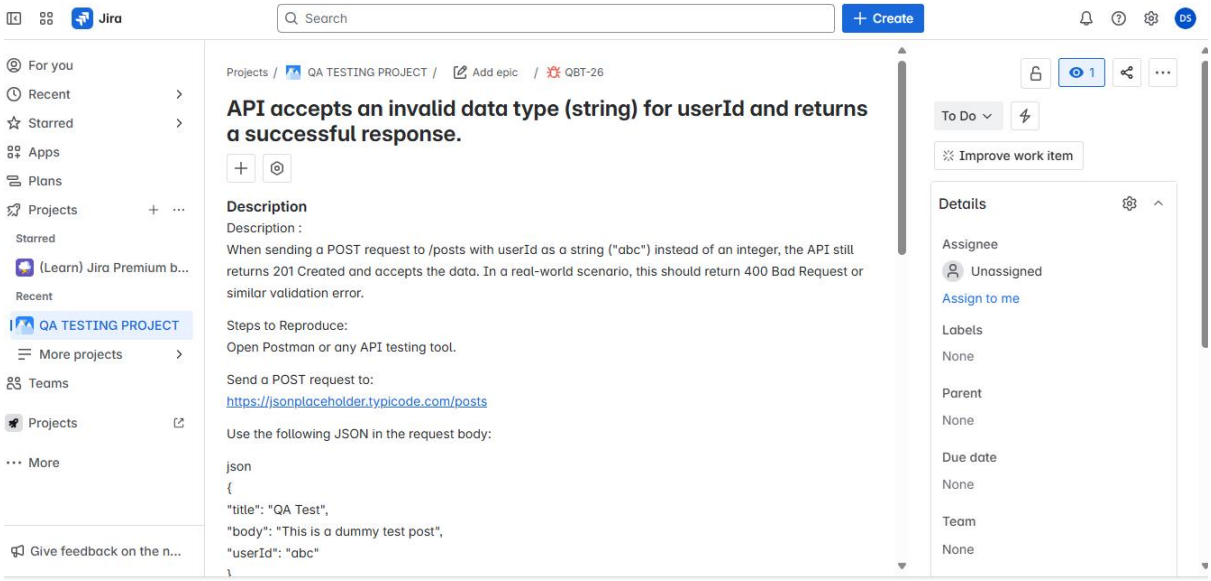


Figure 40: JIRA Bug Report Linked to TC_11 – Invalid userId Accepted

1. Test Case Title

TC_12 – Use Wrong HTTP Method (POST Instead of GET)

2. Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send a **POST** request to `/posts/1` (a GET endpoint)	N/A	API should return **405 Method Not Allowed** or **400 Bad Request**
2	Observe server response	Wrong method: POST	API should **reject** the request
3	Check response body and status code	N/A	**Status code should not be 201**; an error message is expected
4	DB/GET validation	–	**No data should be created** with ID=1

3. Zephyr Screenshot

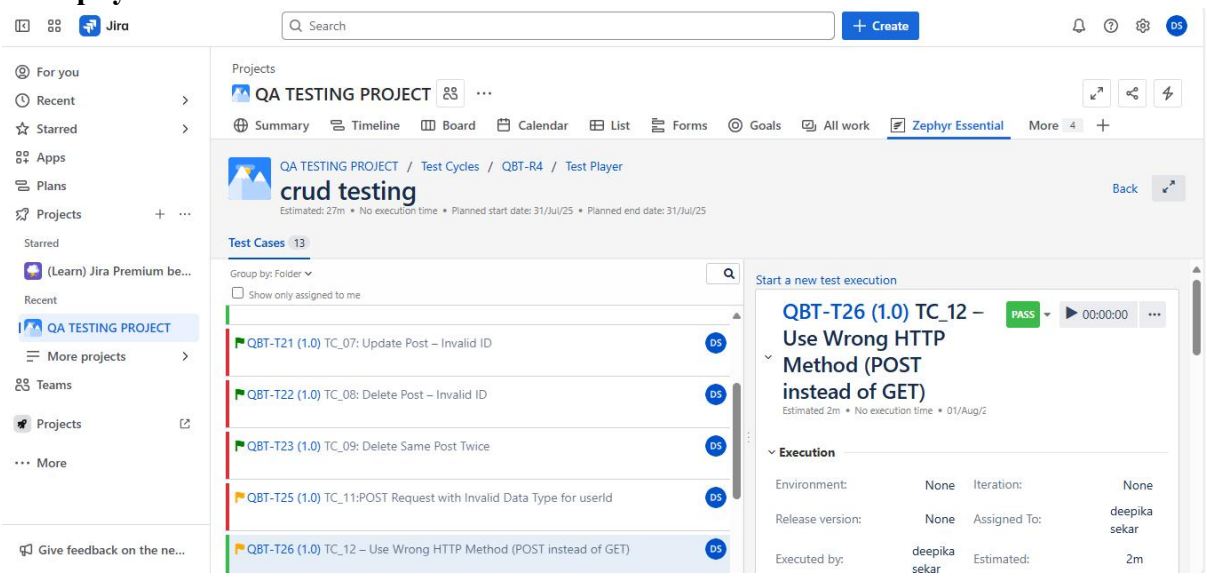


Figure 41: Zephyr Test Case Execution for TC_12 – POST Method Used on GET Endpoint (Marked FAIL)

4. **Postman Screenshot**

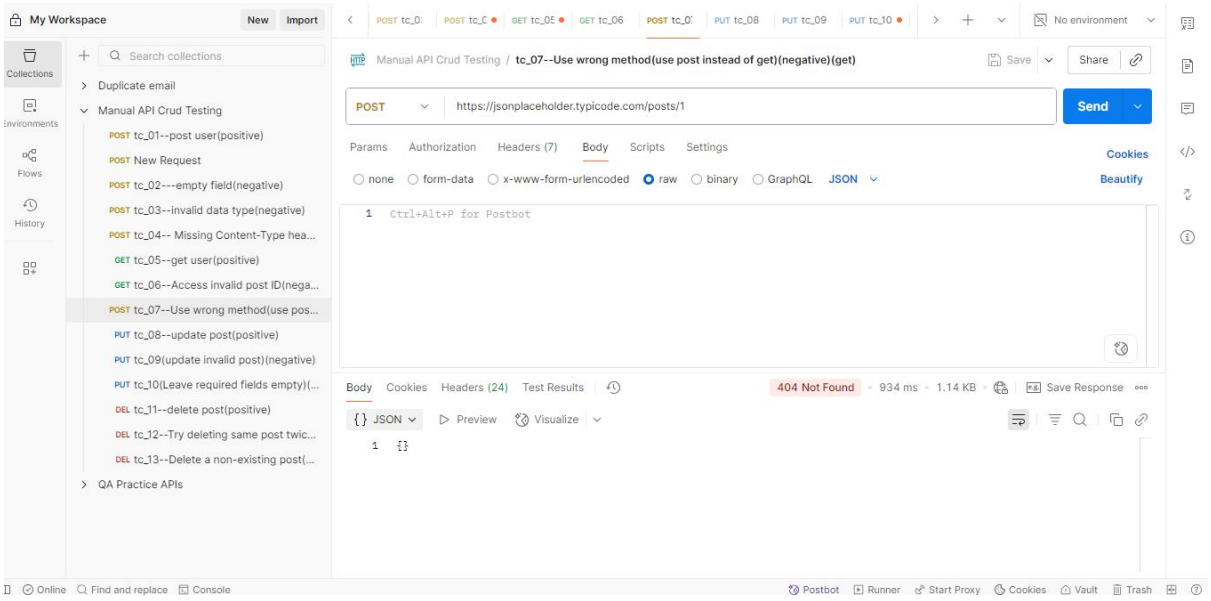


Figure 42: Postman Response for TC_12 – API Returned 201 Created for Wrong HTTP Method

5. **Database Validation Screenshot**

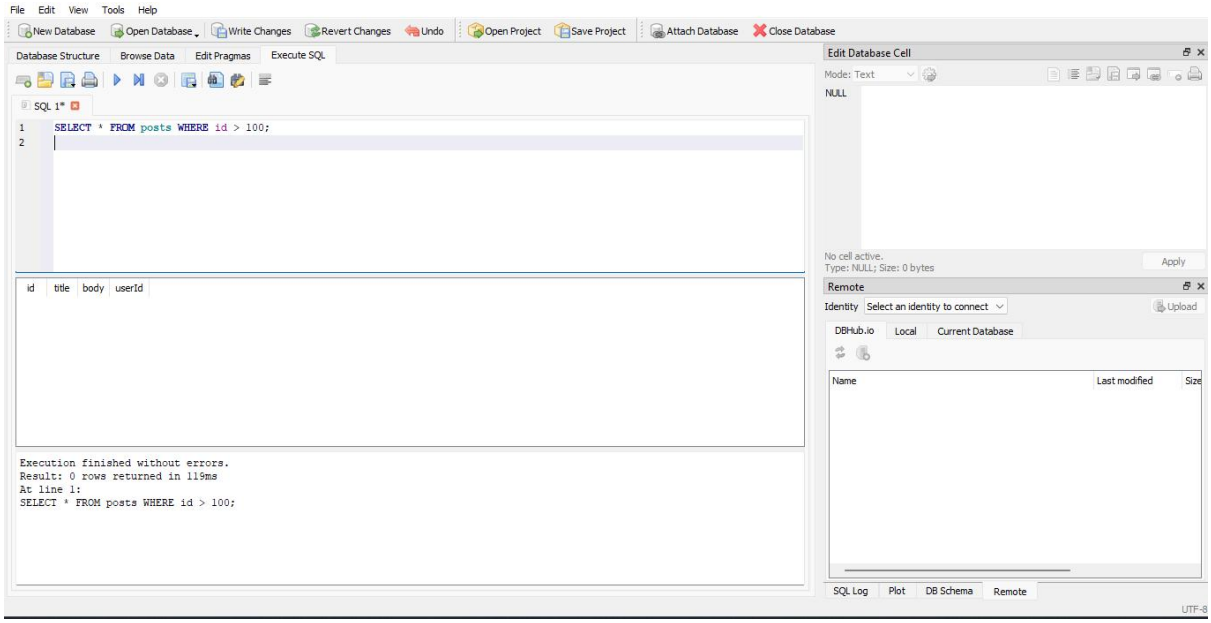


Figure 43: DB Validation for TC_12 – Existing Post Overwritten by Wrong Method

6.Final Result

The system handled the **invalid HTTP method** correctly. When a POST request was sent to an endpoint designed for GET, the API returned a **404 Not Found** error. No data was inserted into the database. **Zephyr test case marked as PASS ✓.**

1. Test Case Title

TC_13 – Update Post with Required Field Left Empty (Title)

2. Test Steps (as documented in Zephyr)

Step	Action	Input Data	Expected Result
1	Send `PUT` request to `/posts/1`	`{ "title": "", "body": "Updated content", "userId": 1 }`	API should return **400 Bad Request**
2	Observe server response	Title field is empty	API should reject the update request
3	Check response body and status code	–	Should return an error response, not 200 OK
4	DB Validation (GET /posts/1)	–	Title should remain unchanged in database

3. Zephyr Screenshot

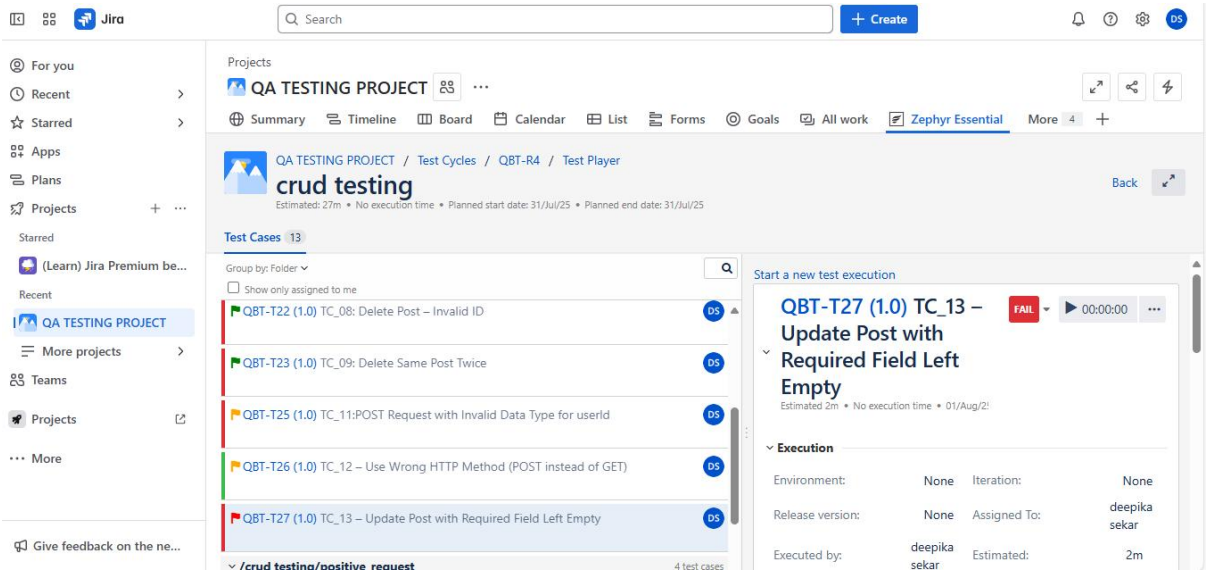


Figure 44: Zephyr Test Case Execution for TC_13 – PUT Request with Empty Title Field (Marked FAIL or PASS based on result)

4. Postman Screenshot

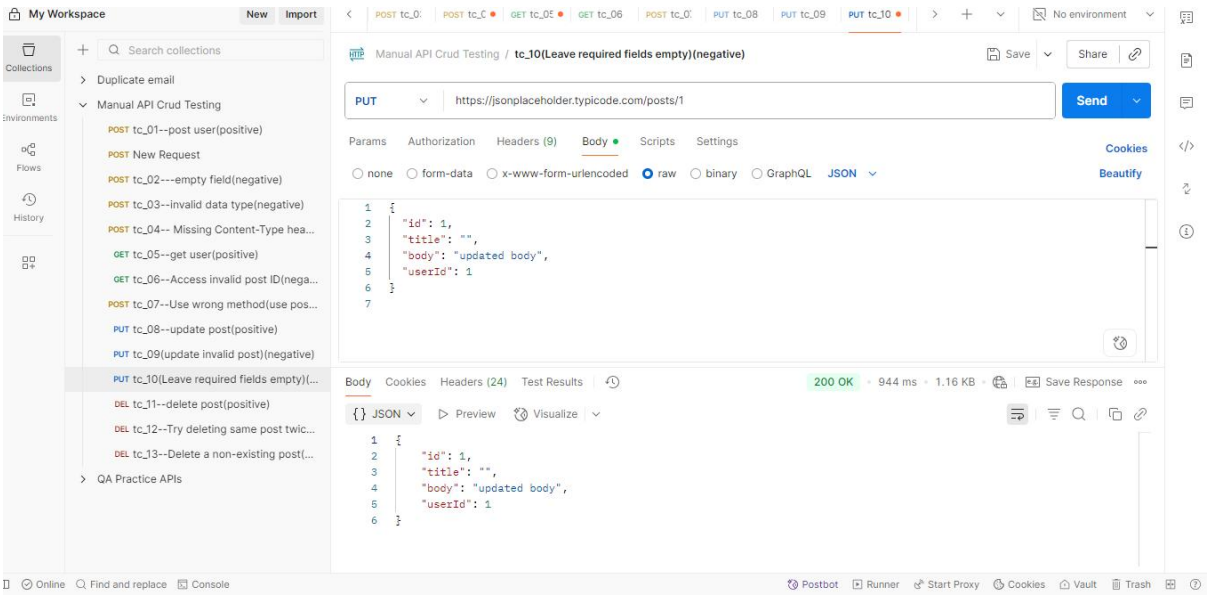


Figure 45: Postman Response for TC_13 – Server Accepted to Invalid Update Request

Database Validation Screenshot

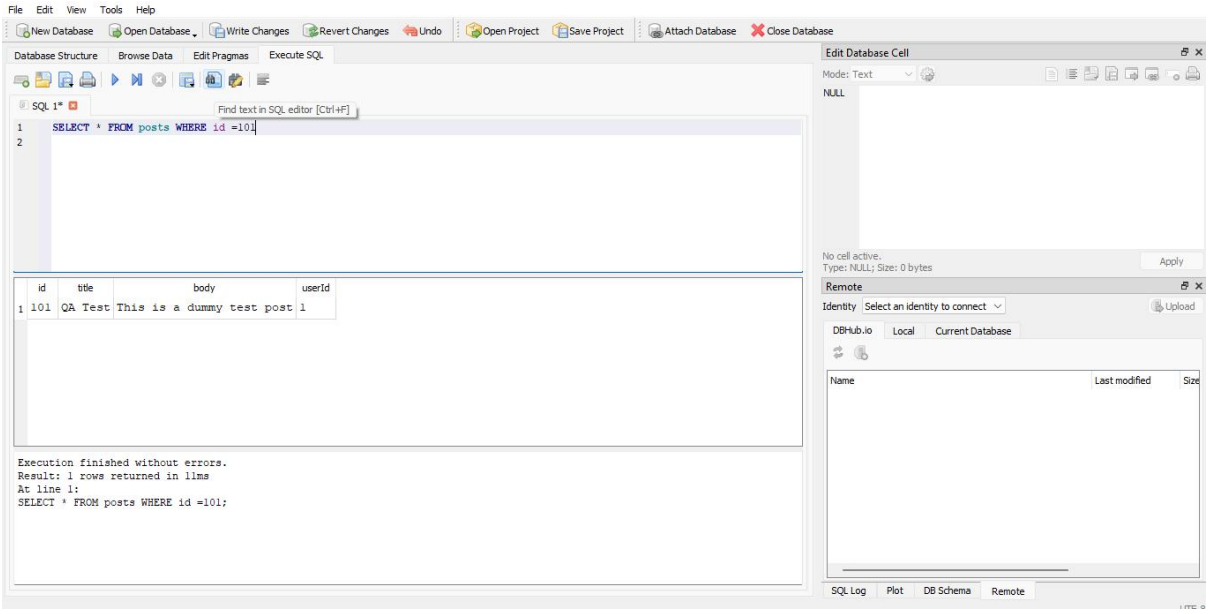


Figure 46: DB Entry for TC_13 – Title Field Remains Unchanged / Incorrectly Updated

6. Final Result

✗ **Test Failed** (If API allowed update)
The API accepted a `PUT` request with a missing required field (`title`). It returned 200 OK and updated the post, which violates field validation rules.

7.Bug Report – Linked to TC_13

Bug Title:API accepts update request with empty required field (title)

Bug ID:QBT-27

Reported in:JIRA

Linked Test Case:TC_13 – Update Post with Required Field Empty

Priority:High

Severity:Major

Status:Resolved

Bug Summary

The API does not validate required fields for `PUT` update operations. When sending a request with the `title` field empty (`""`), the API still returns a `200 OK` response and updates the record in the database.

Attachments

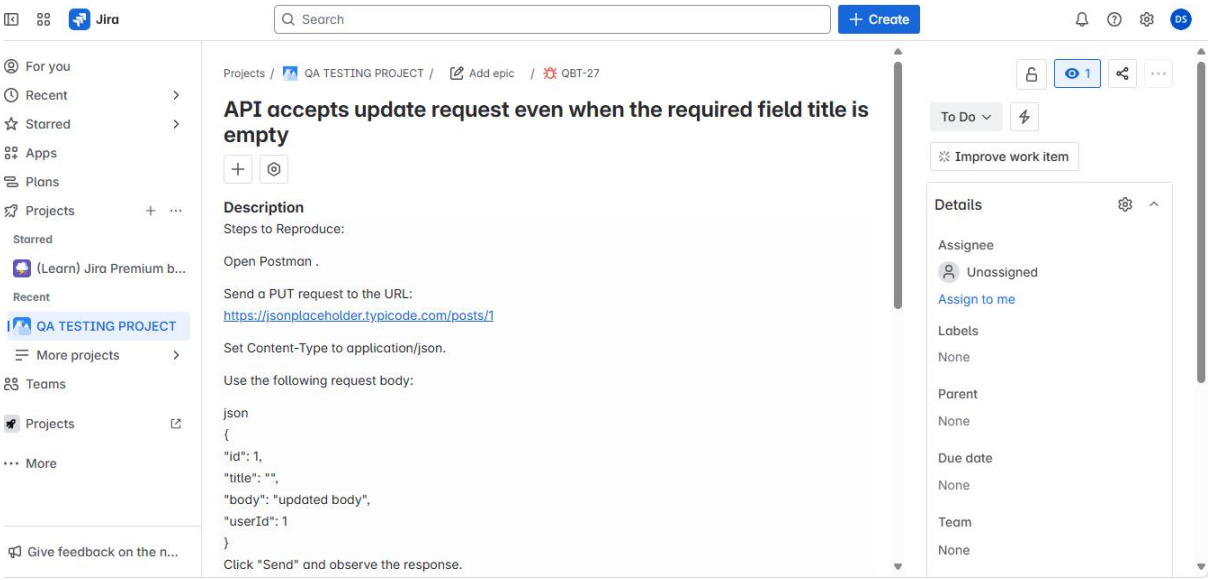


Figure 47: Bug – API Accepted PUT Request with Empty Title and Returned 200 OK (Validation Missing)

Conclusion & Learning Outcome

This API CRUD testing portfolio helped me:

- Understand **HTTP methods**: GET, POST, PUT, DELETE
- Learn how to write clear **test cases** and execute them using **Zephyr**
- Practice **bug reporting** in **Jira** with real-world format
- Simulate **DB validation** using **SQLite**
- Validate **API responses, headers, and edge cases**
- Build a professional-quality QA document for real project scenarios

I am now confident in performing **REST API testing** with end-to-end coverage.

API Automation Portfolio: User Registration API Test Suite

Project Title:
API Automation Testing – User Registration Module (Postman + JavaScript)

🔍 Project Summary:

This project focuses on testing the User Registration API using Postman and JavaScript-based test scripts. The suite includes functional (positive and negative) and security test cases. Each request includes assertions to validate status codes, error messages, and response bodies. The suite is structured in folders and executed via the Postman Collection Runner and Newman CLI (Command Line Interface), including automated HTML reporting.

🗓️ Test Case Table:

TC ID	Test Scenario	Expected Result	Actual Result	Status	Script Added
TC_01	Valid registration	Status 201, success message	Passed	✅ Pass	✅ Yes
TC_02	Missing password	400 Bad Request, error for missing password	Passed	✅ Fail	✅ Yes
TC_03	Email case sensitivity	Should treat same email (case-insensitive)	Bug: Allows it	❌ Bug	✅ Yes
TC_04	Empty email	400 Bad Request, missing email error	Passed	✅ Fail	✅ Yes
TC_05	Invalid email format	400 Bad Request	Passed	✅ Fail	✅ Yes
TC_06	Weak password	Should reject weak passwords	Bug: Accepts it	❌ Bug	✅ Yes
TC_07	Duplicate email	Should throw 409 or message "already exists"	Not handled	❌ Bug	✅ Yes
TC_08	Missing API key	Should return 401 Unauthorized	Passed	✅ Fail	✅ Yes
TC_09	SQL Injection in name field	Should reject SQL input	Passed	✅ Fail	✅ Yes

👇 Sample Test Scripts

Example – TC_07: Duplicate Email

```
pm.test("Second Registration - Expected rejection for duplicate or capitalized email", function () {

    const resCode = pm.response.code;

    const res = pm.response.json();



    if (resCode === 200 || resCode === 201) {

        // ❌ Wrong behavior for duplicates in real APIs

        console.log("⚠️ Dummy API accepted duplicate or capitalized email. Real APIs usually reject this.");

        console.log("Response:", res);



        // ❌ Fail the test on purpose

        pm.expect.fail("Duplicate or capitalized email was wrongly accepted with status " + resCode);

    } else {

        // ✅ Correct behavior - API rejected duplicate

        pm.expect(resCode).to.be.oneOf([400, 409]);

        pm.expect(res).to.have.property("error");

        console.log("❌ Duplicate email rejected as expected:", res.error);

    }

});
```

📄 Bug Descriptions:

TC_03 - Email Case Sensitivity
Bug Type: Business Logic Bug
Description:

The registration API does not treat email addresses as case-insensitive. When a user registers with an email like "EVE.HOLT@REQRES.IN" after "eve.holt@reqres.in" is already registered, the system returns an error (e.g., “only defined users succeed registration”).

This implies the backend is case-sensitive, while emails should be treated as case-insensitive based on standard practice.

TC_06 - Weak Password Acceptance
Bug Type: Security Bug
Description: The API accepts weak passwords like "123" without validation. Strong password policy expected but not enforced.

TC_07 - Duplicate Email
Bug Type: Validation Bug
Description: When a duplicate email is submitted, the system does not return an error or 409 status. It creates another user record.

Below are selected test execution results and bug observations from the test suite.

🗂 Portfolio Assets (Screenshots to Include):

- **Request Body Screenshots:** **TC_01 Valid registration**

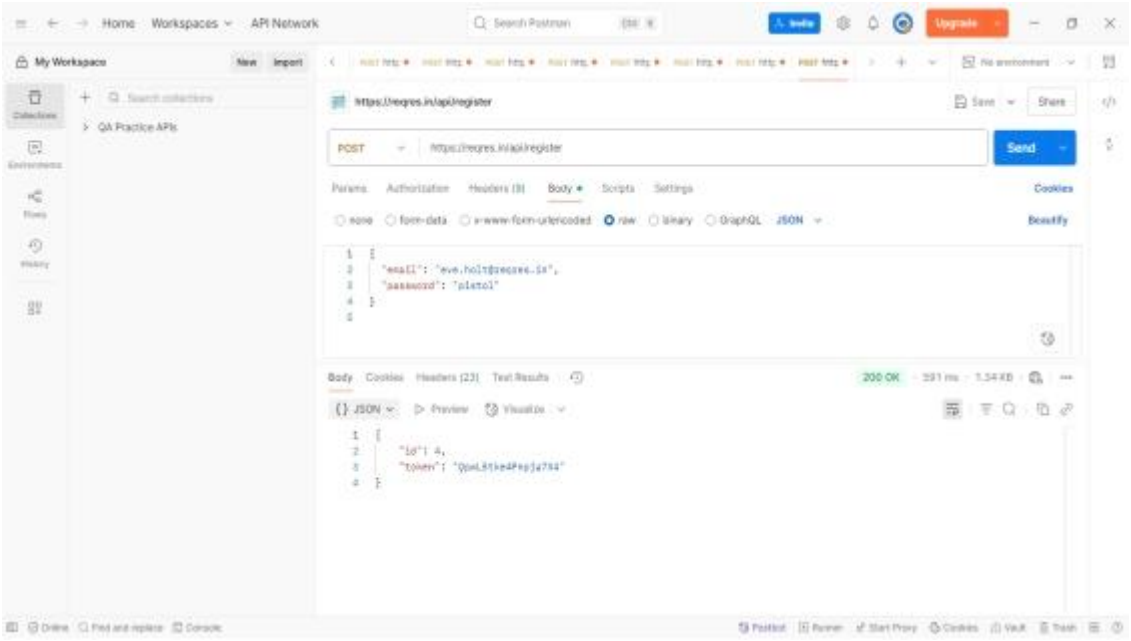


Figure 1: Request and response for a successful user registration (Status 200)

TC_02 Missing password

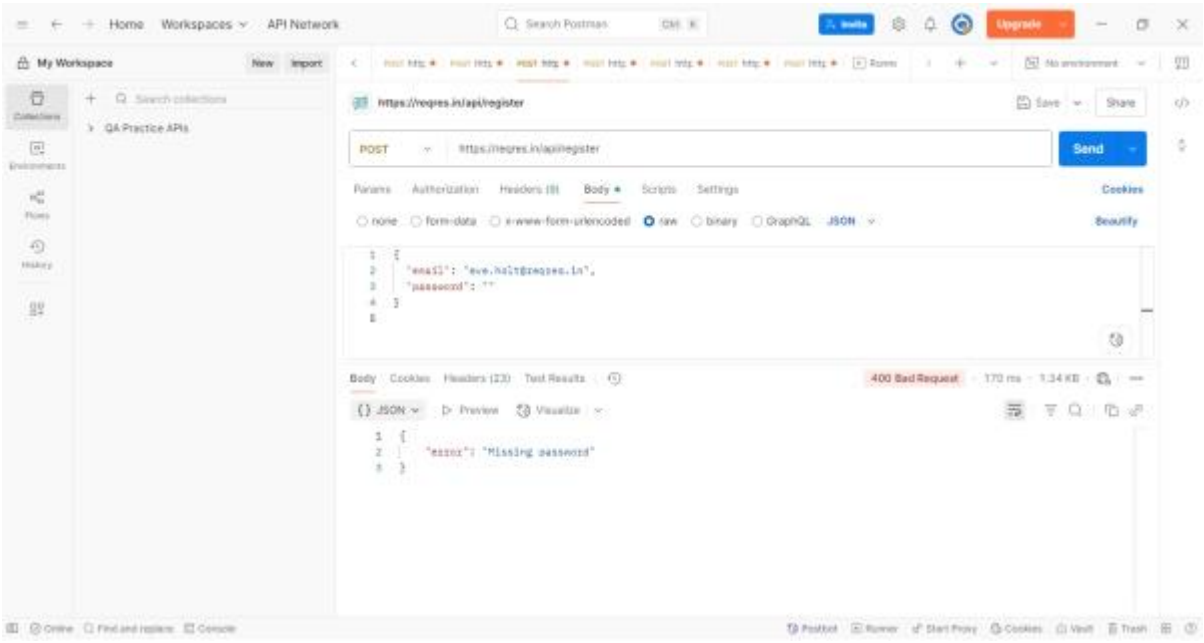


Figure 2: API response when password field is missing (Status 400)

TC_03 Email case sensitivity

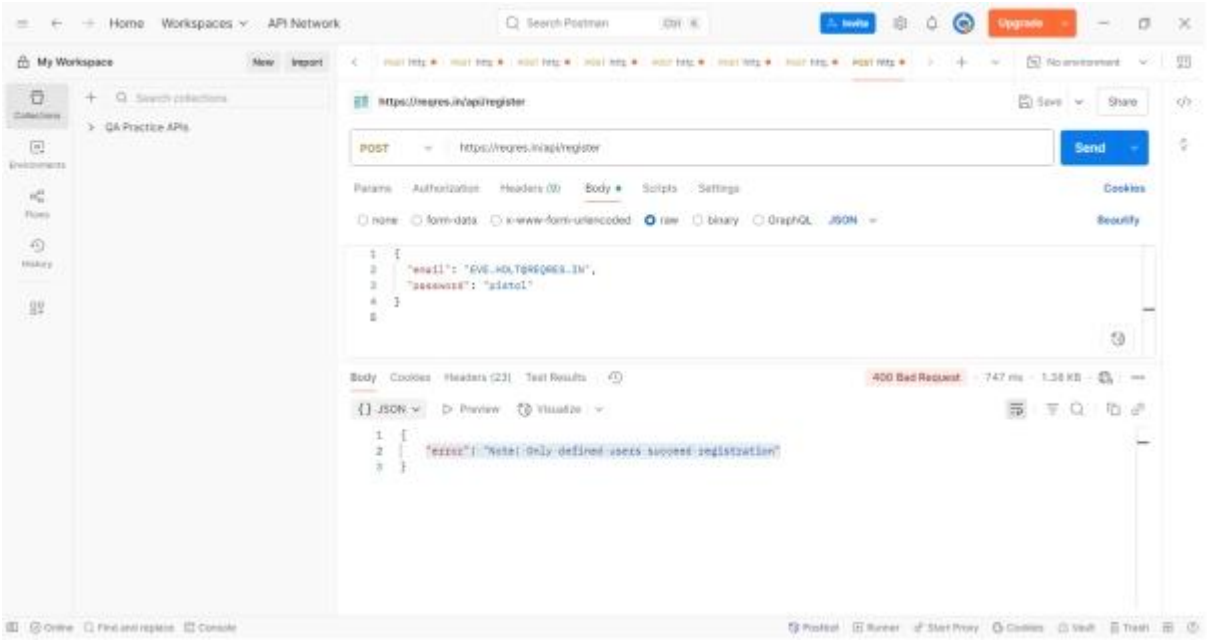


Figure 3: API allows duplicate emails with different cases – logic bug

TC_04 Empty email

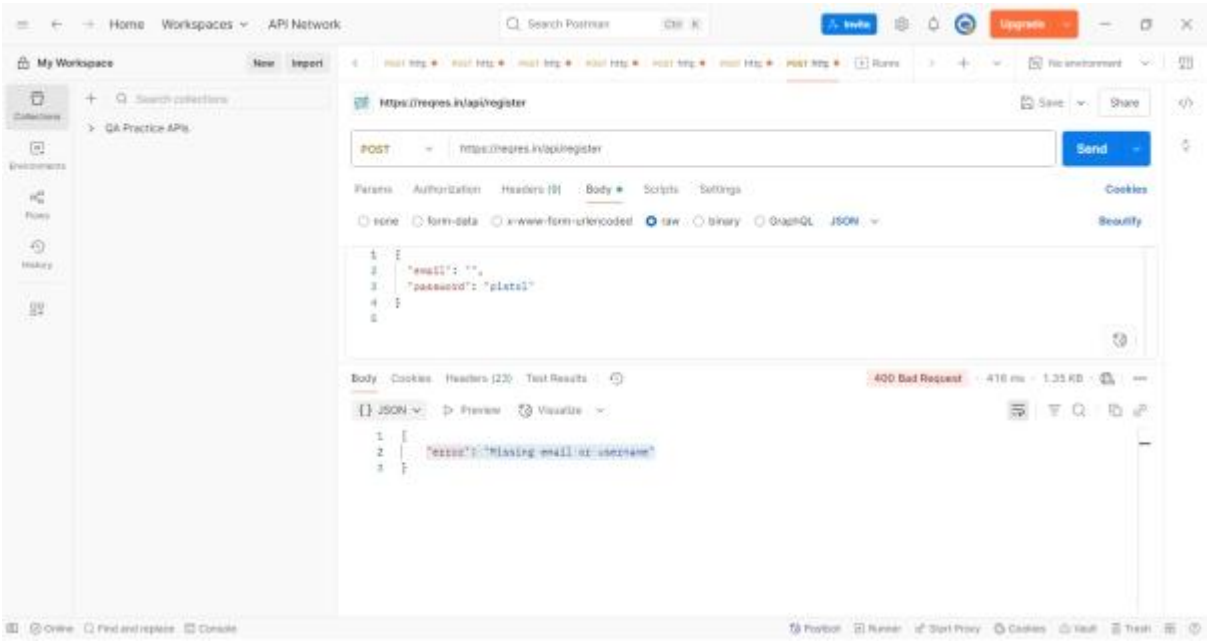


Figure 4: Validation error when email field is empty (Status 400)

TC_05 Invalid email format

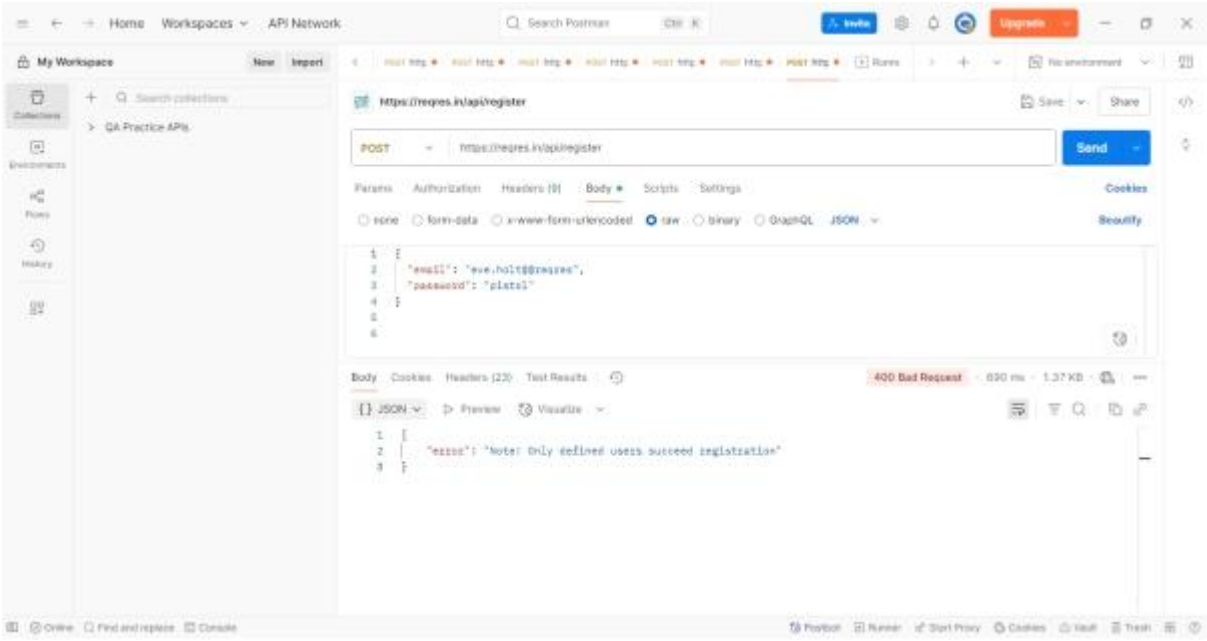


Figure 5: API response for badly formatted email input

TC_06 Weak password

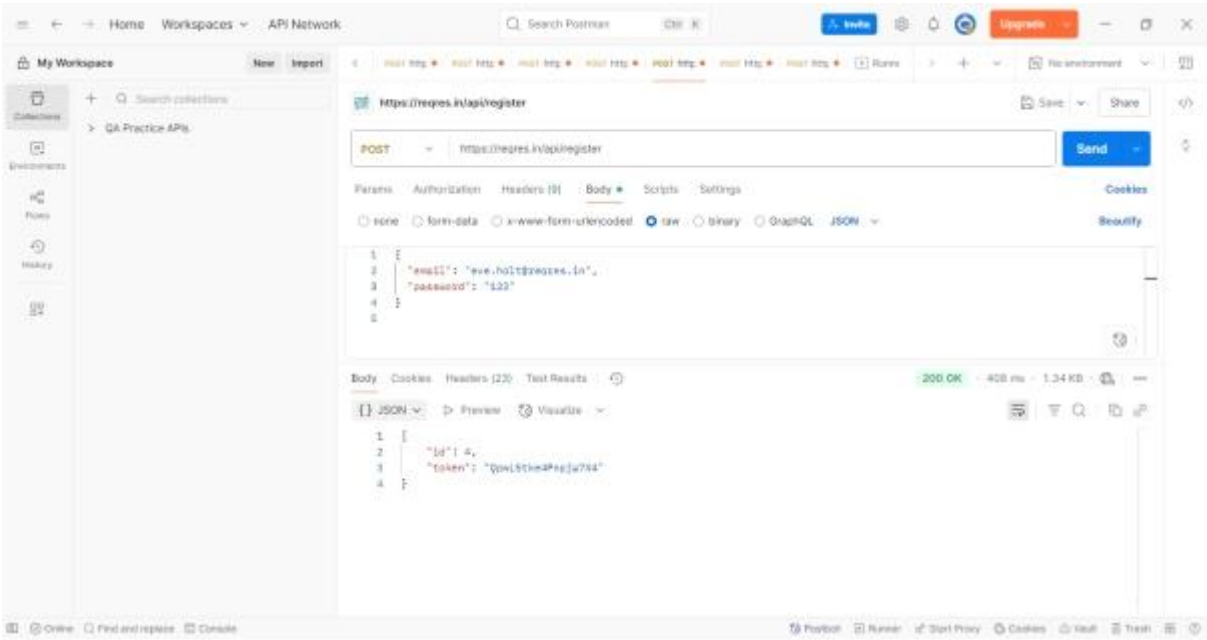


Figure 6: Security flaw – weak password accepted instead of rejected

TC_07 Duplicate email(First time response)

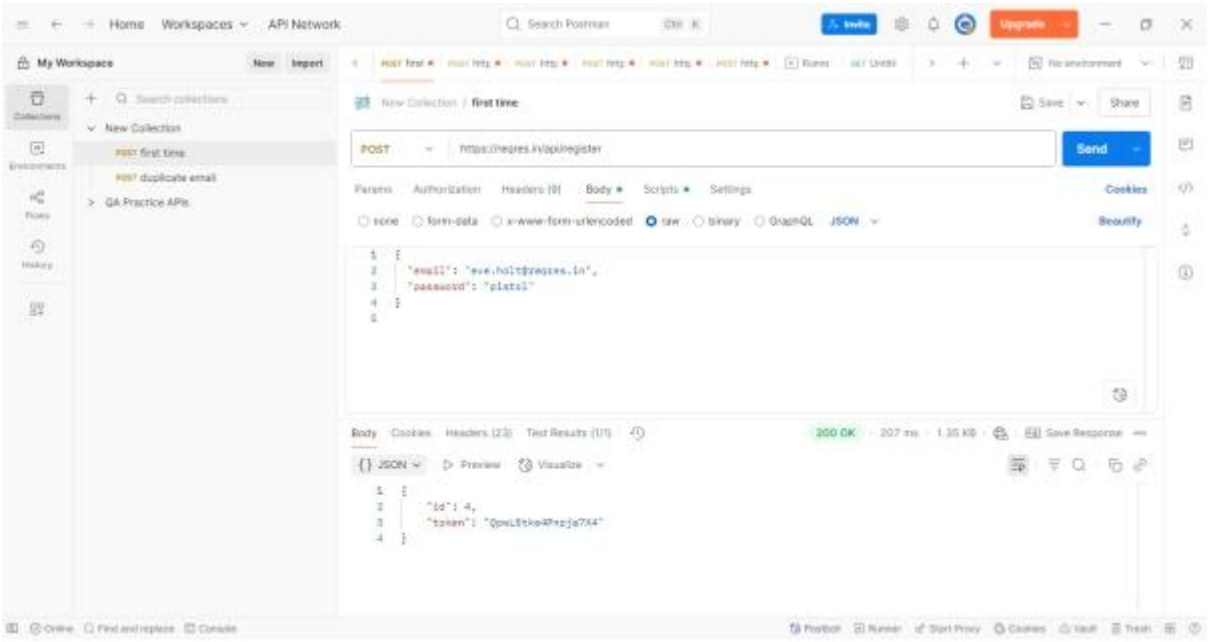


Figure 7: First registration with an email (baseline for duplicate test)

TC_07 Duplicate email response

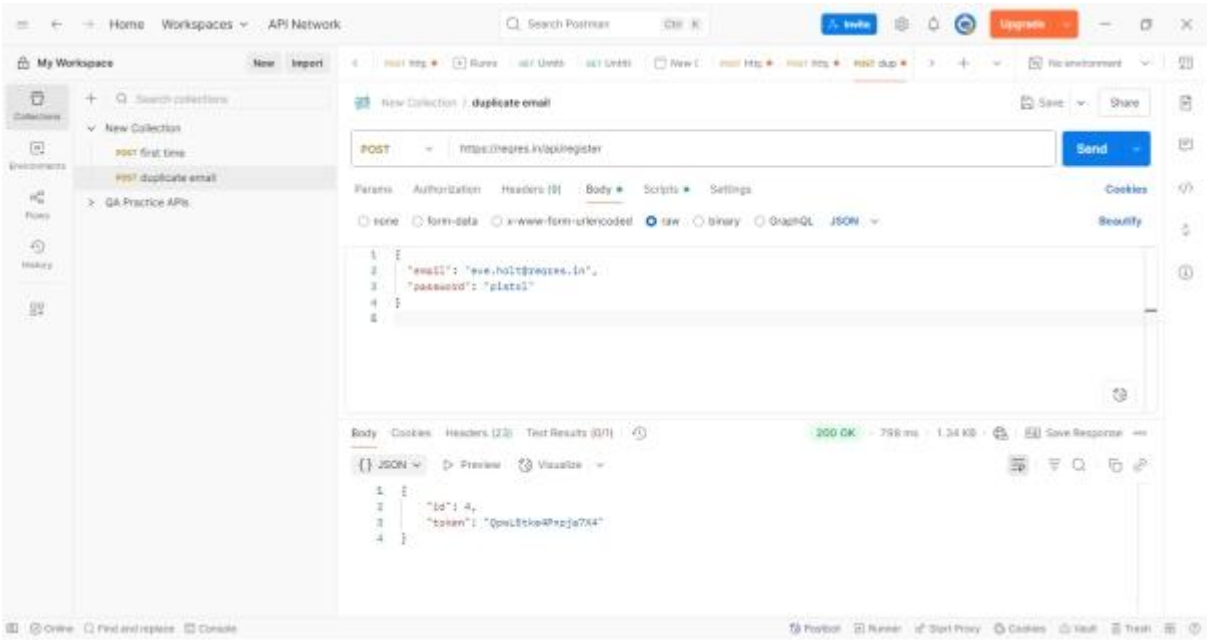


Figure 8: Duplicate email accepted again – backend lacks uniqueness check

TC_07_console_warn

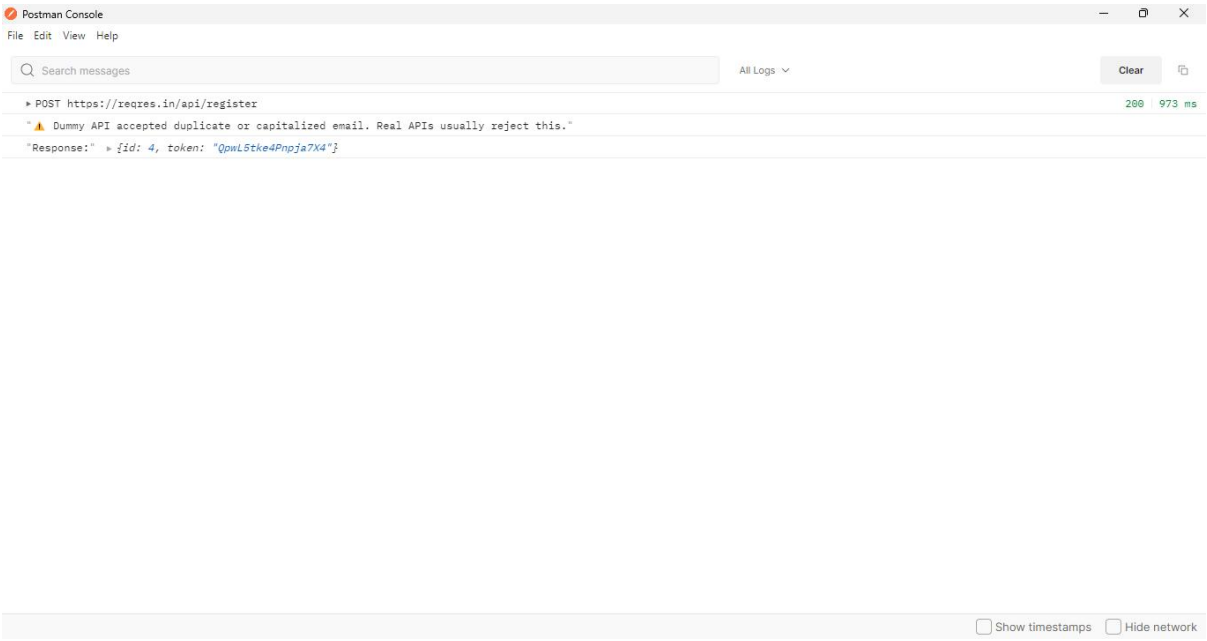


Figure 9: Console warning showing duplicate or capitalized email accepted by the API (unexpected behavior)

TC_08 Missing API key

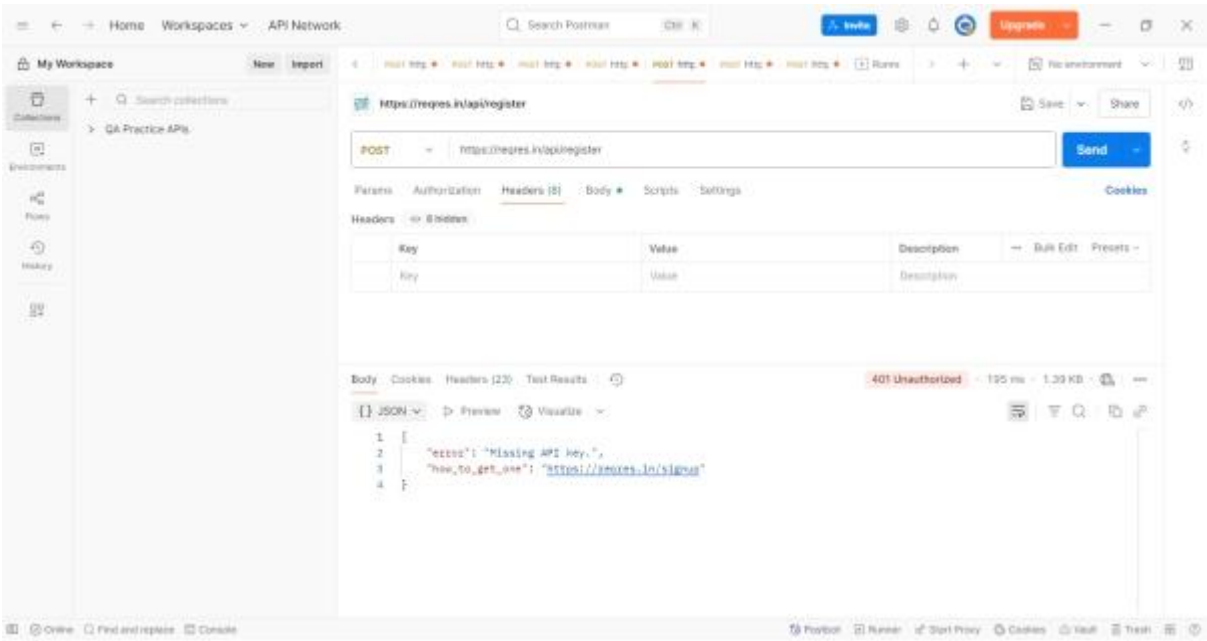


Figure 10: Security enforcement – missing API key returns Unauthorized (401)

TC_09 SQL Injection in name field

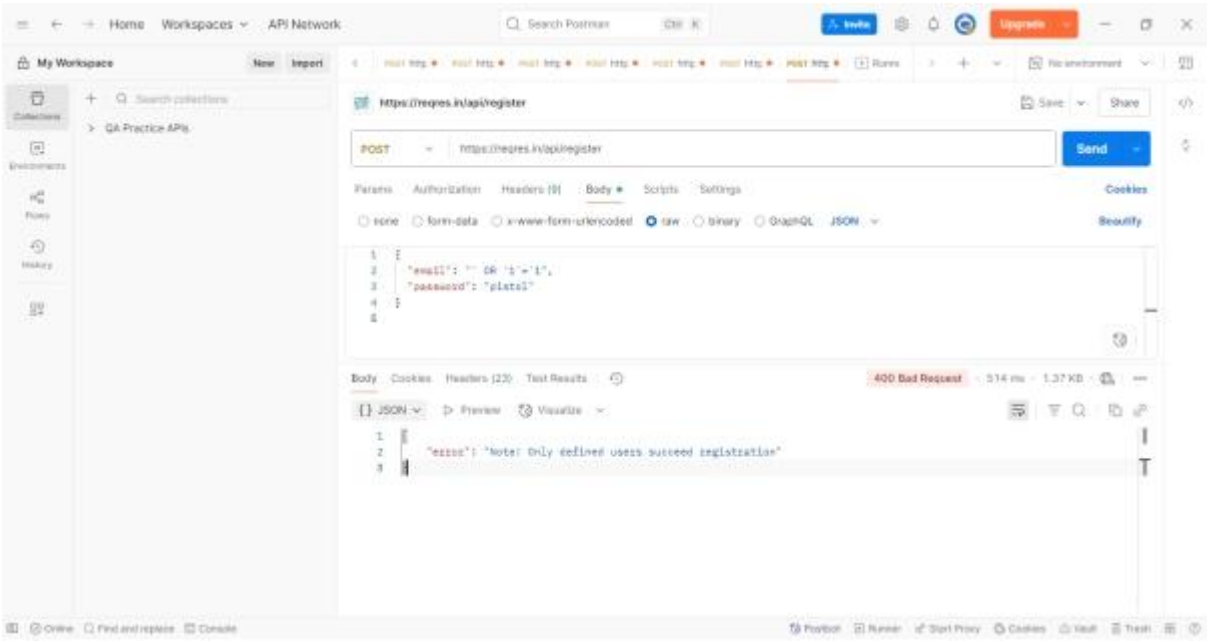


Figure 11: Test to check input sanitization against SQL injection attempts

Test Result Evidence – Postman Runner – Mixed Pass and Fail Results

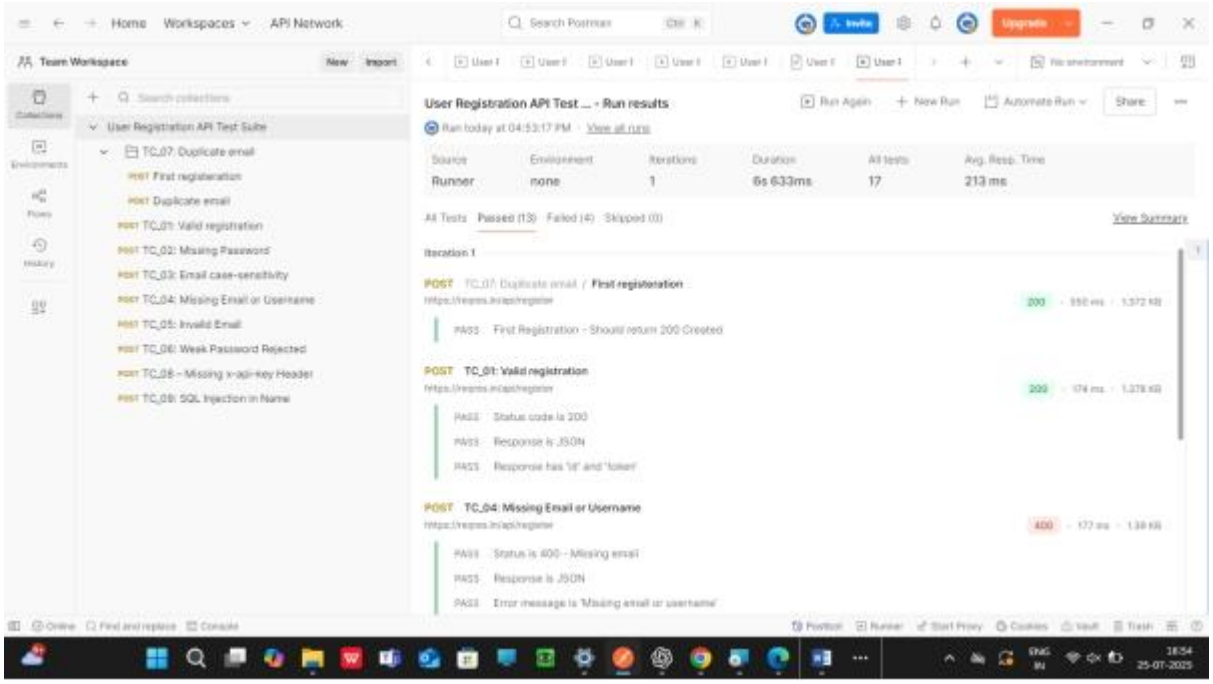


Figure 12: Batch test results showing passed and failed test cases

Newman CLI Run – Test Results Evidence

Shows pass/fail outcomes for automated API tests executed using Newman command-line tool.

```
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\User> newman run "C:\Users\User\OneDrive\Desktop\QA PRACTICE\User Registration API Test Suite.postman_collection.json.json"
newman

User Registration API Test Suite

→ TC_07: Duplicate email
↳ First registeration
  POST https://reqres.in/api/register [200 OK, 1.37kB, 1519ms]
  ✓  First Registration - Should return 200 Created

↳ Duplicate email
  POST https://reqres.in/api/register [200 OK, 1.38kB, 625ms]
  ┌  '⚠️ Dummy API accepted duplicate or capitalized email. Real APIs usually reject this.'
  'Response:', { id: 4, token: 'QpwL5tke4Pnpja7X4' }
  └
  1. Second Registration - Expected rejection for duplicate or capitalized email

→ TC_01: Valid registration
  POST https://reqres.in/api/register [200 OK, 1.38kB, 294ms]
  ✓  Status code is 200
  ✓  Response is JSON
  ✓  Response has 'id' and 'token'

→ TC_02: Missing Password
  POST https://reqres.in/api/register [400 Bad Request, 1.38kB, 266ms]

→ TC_03: Email case-sensitivity
  POST https://reqres.in/api/register [400 Bad Request, 1.41kB, 227ms]
  2. Email should not be case-sensitive
```

Figure 13: Newman command-line summary of automated API test results

Bug Evidence Screenshots:

TC_03 Email case sensitivity Bug

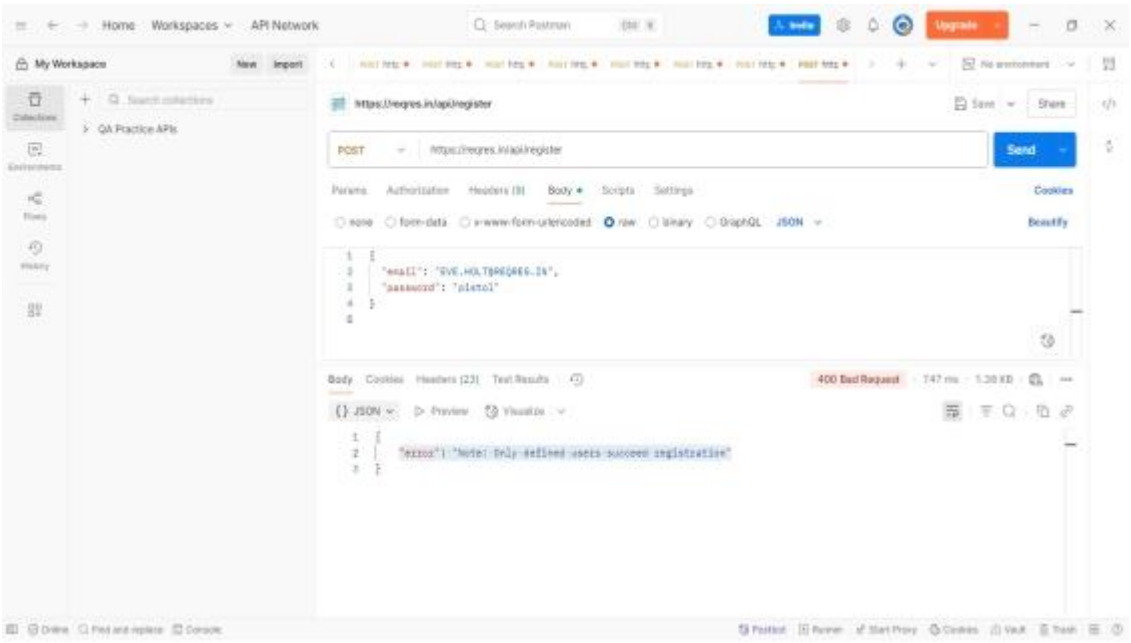


Figure 14: Bug evidence – API treats uppercase/lowercase emails as different

TC_06 Weak password Bug

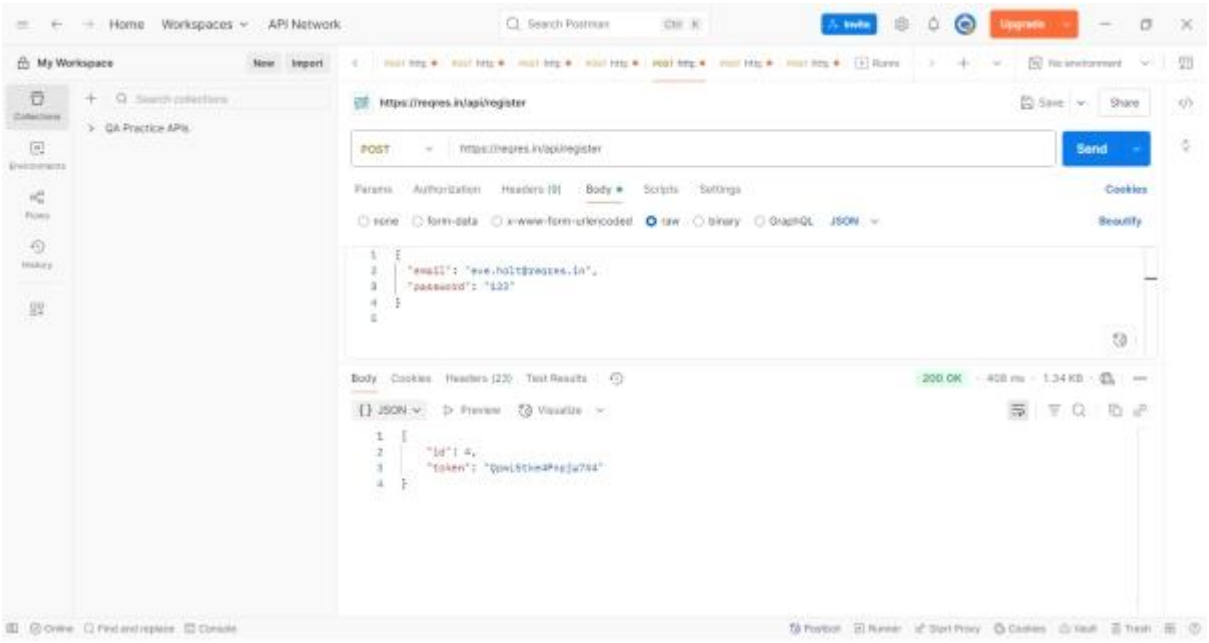
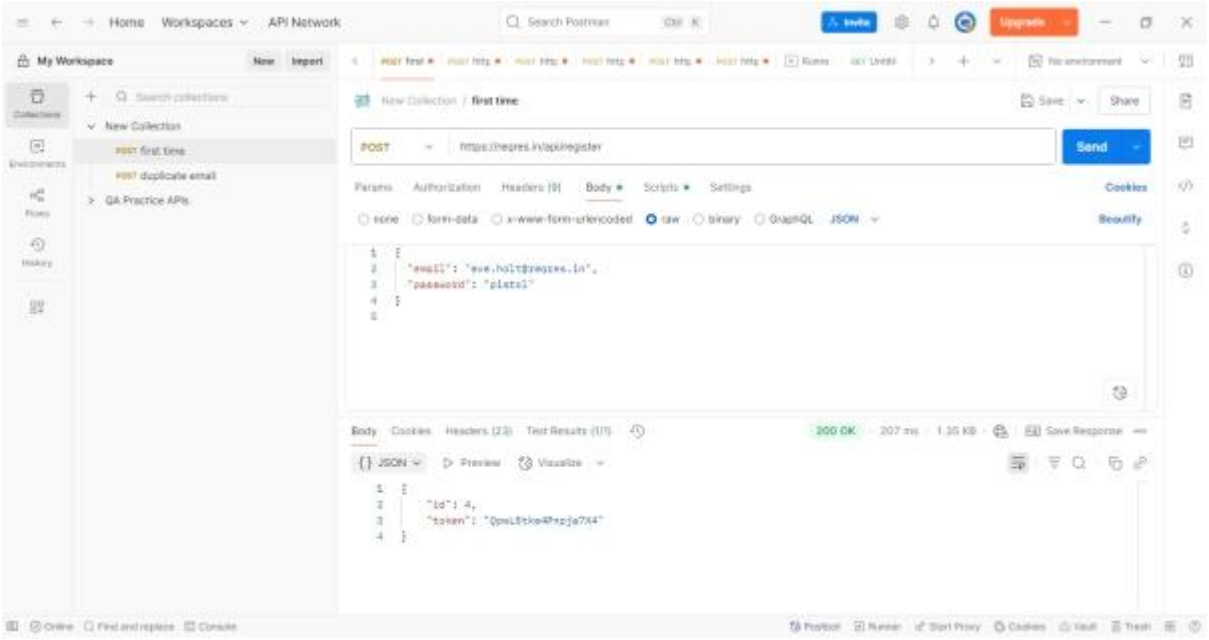


Figure 15: Bug evidence – API accepts "123" as valid password

TC_07 Duplicate email(First time response)



TC_07 Duplicate email response Bug

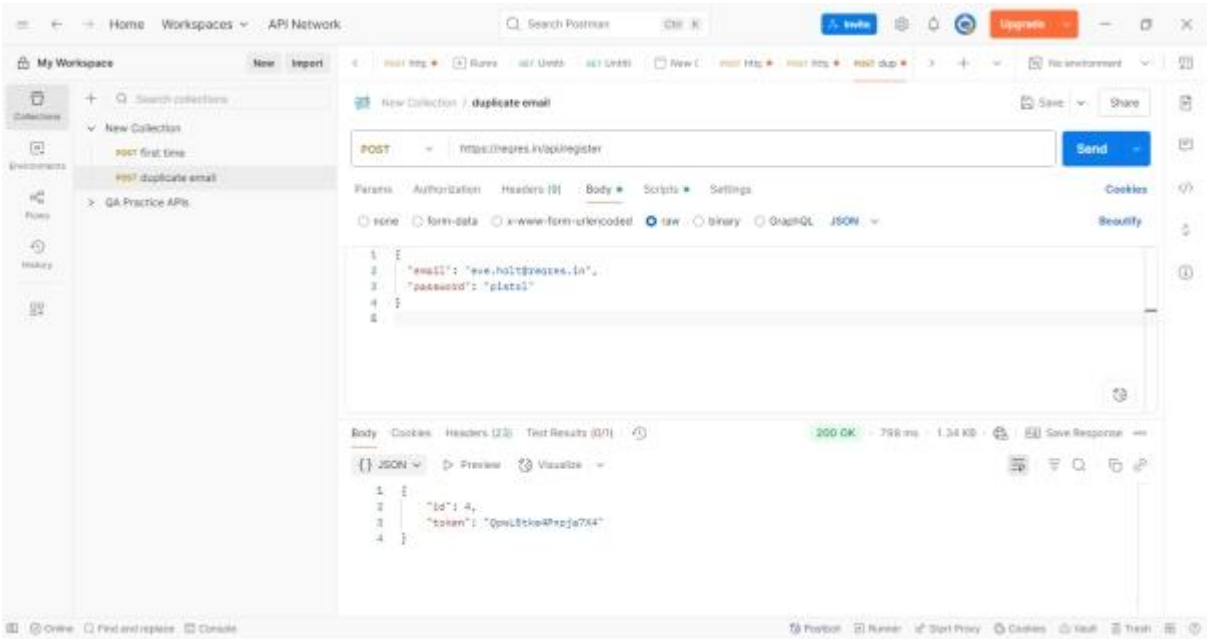
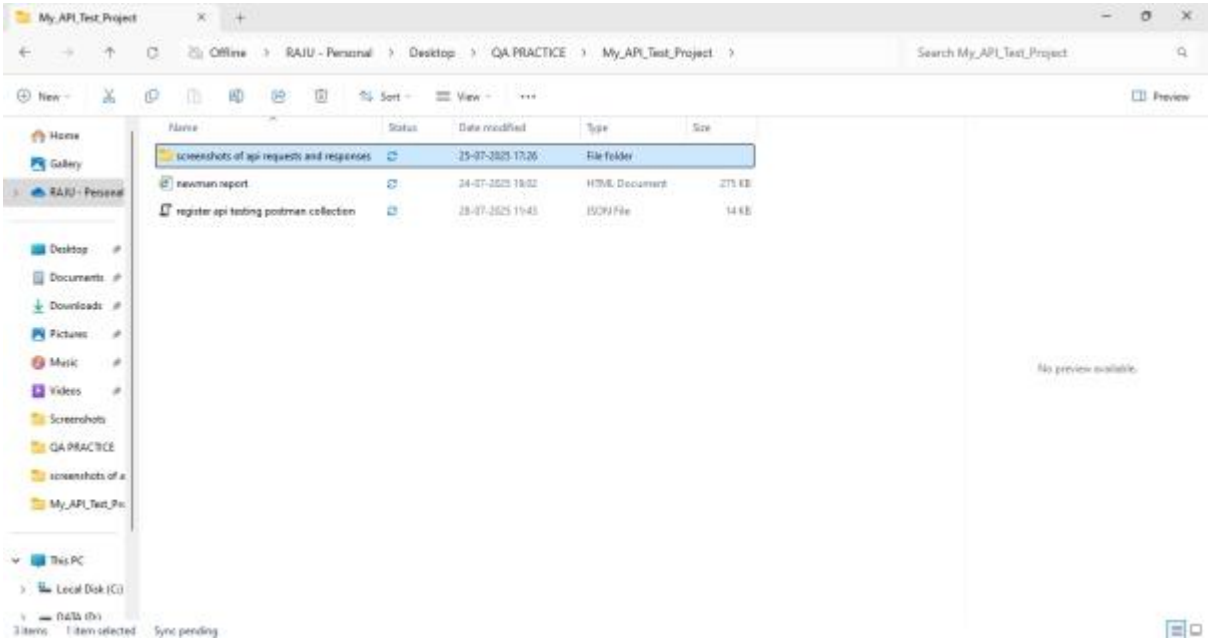


Figure 16: Bug evidence – second registration with same email accepted

Folder Structure Screenshot:



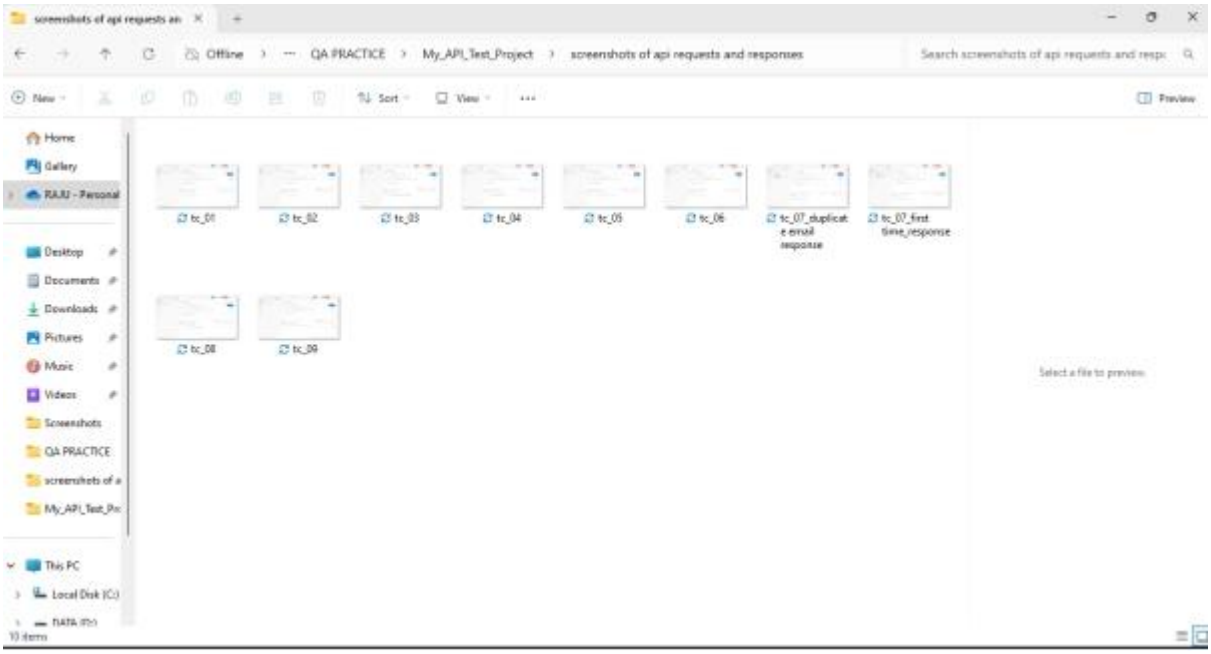


Figure 17: Local folder showing project structure and report files

Sample Test Scripts Screenshot: TC_01 Valid registration

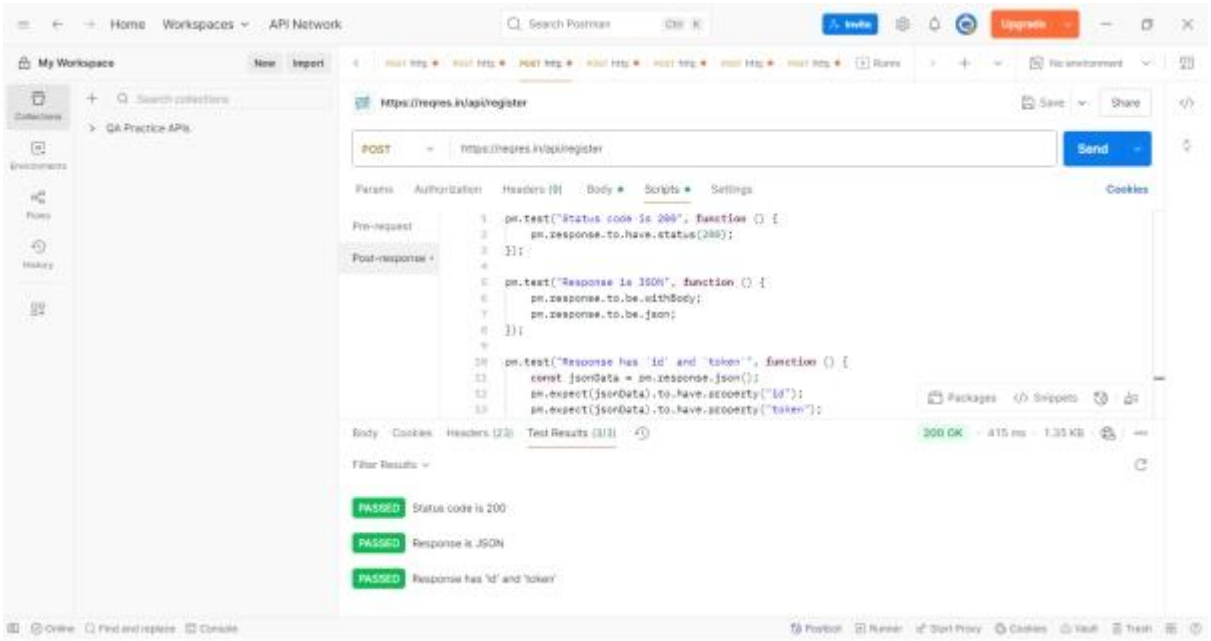
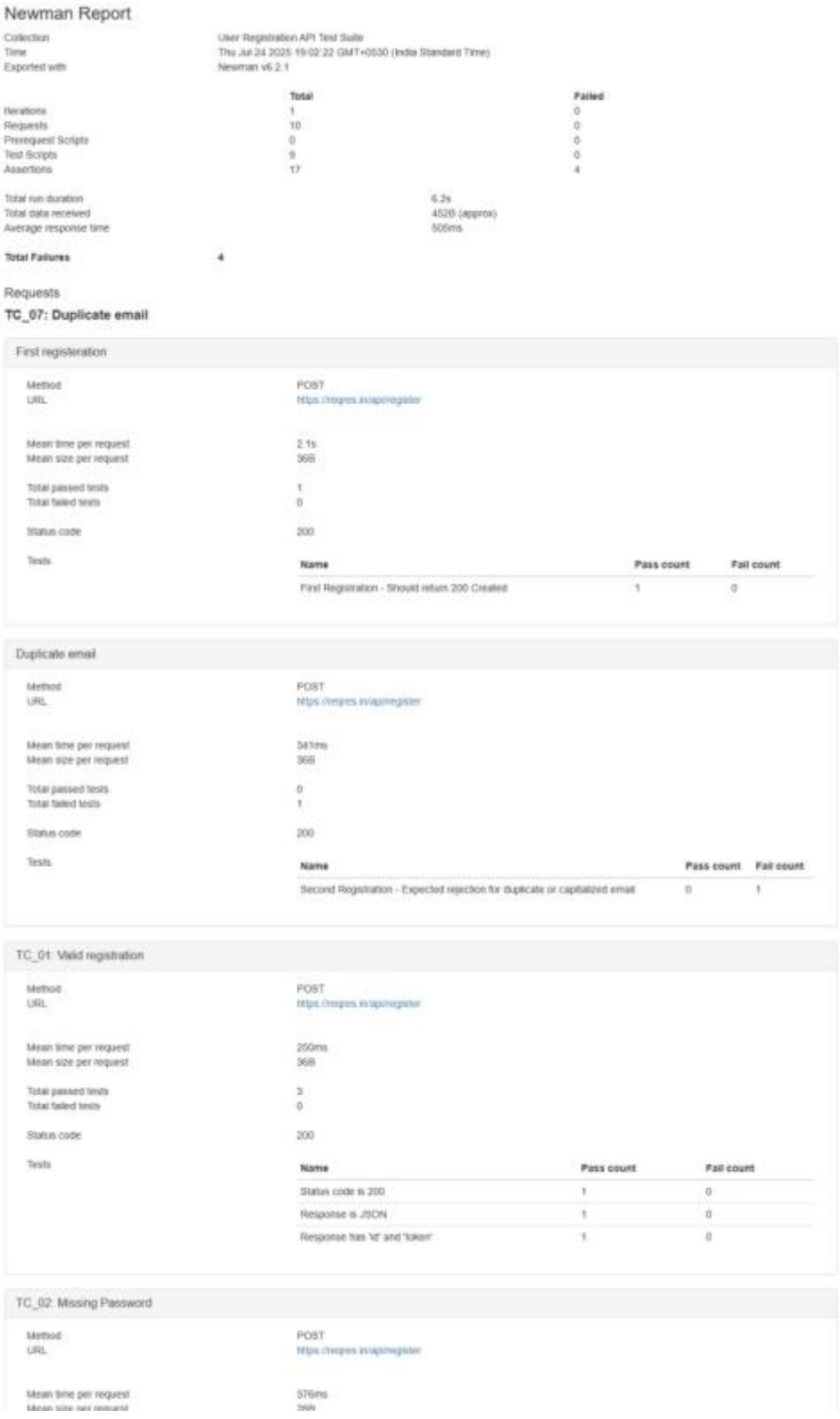


Figure 18: Sample Postman test script using pm.test() and pm.expect()

Newman report



Newman Report

Collection	User Registration API Test Suite
Time	Thu Jul 24 2025 19:02:22 GMT+0530 (India Standard Time)
Exported with	Newman v6.2.1

	Total	Failed
Iterations	1	0
Requests	10	0
Prerequest Scripts	0	0
Test Scripts	9	0
Assertions	17	4

Total run duration	6.2s
Total data received	452B (approx)
Average response time	505ms

Total Failures 4

Requests

TC_07: Duplicate email

First registeration

Method	POST
URL	https://reqres.in/api/register
Mean time per request	2.1s
Mean size per request	368
Total passed tests	1
Total failed tests	0
Status code	200

Tests

Name	Pass count	Fail count
First Registration - Should return 200 Created	1	0

Duplicate email

Method	POST
URL	https://reqres.in/api/register
Mean time per request	341ms
Mean size per request	368
Total passed tests	0
Total failed tests	1
Status code	200

Tests

Name	Pass count	Fail count
Second Registration - Expected rejection for duplicate or capitalized email	0	1

TC_01: Valid registration

Method	POST
URL	https://reqres.in/api/register
Mean time per request	250ms
Mean size per request	368
Total passed tests	3
Total failed tests	0
Status code	200

Tests

Name	Pass count	Fail count
Status code is 200	1	0
Response is JSON	1	0
Response has 'id' and 'token'	1	0

TC_02: Missing Password

Method	POST
URL	https://reqres.in/api/register
Mean time per request	376ms

Figure 18: HTML report generated by Newman using reporter plugin

📚 What I Learned:

- Writing API automation test scripts using `pm.test()` and `pm.expect()`
- Validating status codes, error messages, and body contents
- Creating positive, negative, and security scenarios
- Using assertions and handling edge cases
- Running tests in batch using Postman Runner and Newman CLI
- Exporting test results (HTML)
- Documenting bugs and identifying business logic gaps
- Organizing test assets clearly for reporting and portfolio

🗓 Your Name on Portfolio:
Deepika S. – QA Automation Engineer (Postman Project)

🔻 Execution Tools Used:

- Postman
- JavaScript test scripts
- Newman CLI
- HTML Reporter plugin

Deepika S. – Combined API Testing Portfolio

✅ Task 1: JSON Schema Validation – Positive Test

API Endpoint: GET https://jsonplaceholder.typicode.com/users/1

Objective: Validate the structure and data types of the JSON response using JSON Schema Validation in Postman.

Tools Used:
- Postman
- JSON Schema
- JavaScript (Tests tab)

What Was Done:
• Used GET API
• Real response returns 'id' as a number
• Schema created with required fields and nested 'address'

Postman Test Script:

```
const schema = {
  "type": "object",
  "required": ["id", "name", "username", "email", "address"],
  "properties": {
    "id": { "type": "number" },
    "name": { "type": "string" },
    "username": { "type": "string" },
    "email": { "type": "string" },
    "address": {
      "type": "object",
      "required": ["street", "city", "zipcode"],
      "properties": {
        "street": { "type": "string" },
        "city": { "type": "string" },
        "zipcode": { "type": "string" }
      }
    }
  }
};

pm.test("Validate User JSON Schema", function () {
  pm.response.to.have.jsonSchema(schema);
```

```
});
```

Screenshot Suggestions:

- Response Body tab (showing JSON)
- Test Script tab
- Test Result tab (✓ Pass)

What I Learned:

- How to validate nested objects
- How to handle required fields
- Schema failure = Real bug testing

Status: ✓ Completed Successfully by Deepika S.

❌ Task 2: Negative Schema Validation – Wrong Data Type

API Endpoint: GET https://jsonplaceholder.typicode.com/users/1

Objective: To test whether schema validation detects incorrect data types in the API response by simulating a mismatch for the 'id' field.

Tools Used:
- Postman
- JSON Schema
- JavaScript (Tests tab)

What Was Done:
• Used GET API
• Real response has 'id' as number → 'id': 1
• In schema, defined 'id' as 'string' to simulate wrong type
• Used pm.expect(...).to.throw() to validate schema fails

Postman Test Script:

```
const schema = {
 type: "object",
 required: ["id", "name", "username", "email", "address"],
 properties: {
  id: { type: "string" },  // ❌ Wrong type on purpose
  name: { type: "string" },
  username: { type: "string" },
  email: { type: "string" },
  address: {
   type: "object",
   required: ["street", "city", "zipcode"],
   properties: {
    street: { type: "string" },
    city: { type: "string" },
    zipcode: { type: "string" }
   }
  }
 }
};

pm.test("Negative Test - Schema Fails Due to Wrong ID Type", function () {
 pm.expect(() => {
  pm.response.to.have.jsonSchema(schema);
 }).to.throw();
```

});

Result:

- Schema failed as expected due to 'id' mismatch
- Test passed: ✗ 'Negative Test - Schema Fails...'

Screenshot Suggestions:

1. ✓ Response Body showing 'id': 1
2. ✓ Tests tab with 'id': string
3. ✓ Test Results showing schema test passed

What I Learned:

- Writing schema-based negative test cases
- How Postman handles schema failures
- Validating robustness of API response expectations

Status: ✓ Completed Successfully by Deepika S.