Results and Observations Report

Executive Summary

Overview

This report details the results of comprehensive API testing conducted on the https://jsonplaceholder.typicode.com/posts API, focusing on core functionalities such as data retrieval, data manipulation, and error handling. Our objective was to assess the API's reliability and general functionality.

Methodology

The testing was carried out using Postman for manual tests and Newman for automated test scripts. We tested a total of 20 endpoints, covering a range of GET, POST, PUT, and DELETE operations. The tests included validation of response codes, response data, error handling, and boundary case scenarios.

High-Level Results

- Total Test Cases: 80
- Passed: 43
- Failed: 37
- Skipped: 0

Around the half of test cases passed, indicating a low level of API functionality and reliability. The failed cases were primarily due to wrong HTTP status code and unexpected response formats.

Key Findings

- Performance: Response times were within acceptable limits for 100% of the test cases.
- Reliability: The API showed consistent results across repeated tests, with no intermittent failures.
- Data Accuracy: Response data matched expected results in all cases with GET requests, however lack of security and additional validations is shown when performing POST, PUT, PATCH requests.
- Error Handling: API doesn't handle properly most of wrong requests returning `200 OK` HTTP code instead of `400 Bad request` or `404 Not found`. No any error message provided.

Recommendations

1. Refine HTTP Status Codes: Adjust error handling to return appropriate HTTP status codes, such as 400 Bad Request and 404 Not Found, for erroneous requests.
2. Standardize Error Responses: Implement a consistent and informative error response format across all endpoints, aiding in clearer communication of issues to API consumers.
3. Strengthen POST, PUT, PATCH Security: Enhance input validation and security checks for data-modifying operations to ensure data integrity and prevent misuse.
4. Expand Test Coverage: Conduct more thorough testing, particularly for edge cases and error scenarios, to improve reliability and identify areas needing refinement.
5. Update API Documentation: Ensure the documentation accurately reflects the API's behavior, especially regarding error handling and response formats.

Test Environment

API Details

API Endpoints Tested:

- https://jsonplaceholder.typicode.com/posts: Endpoint for retrieving and manipulating a collection of posts.
- https://jsonplaceholder.typicode.com/posts/1: Endpoint for interacting with a specific post.

API Type: Free demo APIs provided by JSONPlaceholder, designed for testing and prototyping.

Tools and Technologies

- Postman: Utilized for creating and managing API collections, as well as for designing and manually testing individual API requests.
- Newman: Command-line tool for running Postman tests. Used to execute collections in an automated fashion from the command line interface (CLI).
- Newman Reporter (htmlextra): Integrated with Newman to generate detailed and visually appealing HTML reports of the test runs. Default configuration of htmlextra was used.

Operating Systems:

- Ubuntu 20.04: Primary Linux-based environment for running Newman and analyzing test results.

Test Cases and Scenarios

Test case summary

! Note: since there is no TCMT available for this project/report a brief informationwith test case names will be provided.

Test Case	Description
1	Verify GET /POSTS request
2	Verify GET /POSTS request with unknown parameter
3	Verify GET /POSTS request with all params from same item
4	Verify GET /POSTS request with all params from different items
5	Verify GET /POSTS request ID, USERID match item, TITLE not
6	Verify GET /POSTS request ID, TITLE match item, USERID not
7	Verify GET /POSTS request USERID, TITLE match item, ID not
8	Verify GET /POSTS request with valid TITLE param
9	Verify GET /POSTS request with partial matched TITLE param
10	Verify GET /POSTS request with empty TITLE param
11	Verify GET /POSTS request with USERID=1 param
12	Verify GET /POSTS request with USERID=5 param
13	Verify GET /POSTS request with USERID=10 param
14	Verify GET /POSTS request with USERID=0 param
15	Verify GET /POSTS request with USERID=-1 param
16	Verify GET /POSTS request with USERID=test
17	Verify GET /POSTS request with USERID=11 param
18	Verify GET /POSTS request with USERID=999 param
19	Verify GET /POSTS request with empty USERID param
20	Verify GET /POSTS request with USERIDs 1, 2
21	Verify GET /POSTS request with USERIDs 5, 10
22	Verify GET /POSTS request with USERIDs 10, 999

Test Case	**Description**
23	Verify GET /POSTS request with USERIDs 666, 999
24	Verify GET /POSTS request with IDs 666, 999
25	Verify GET /POSTS request with ID=1 param
26	Verify GET /POSTS request with ID=50 param
27	Verify GET /POSTS request with ID=100 param
28	Verify GET /POSTS request with ID=0 param
29	Verify GET /POSTS request with ID=-1 param
30	Verify GET /POSTS request with ID=test param
31	Verify GET /POSTS request with ID=101 param
32	Verify GET /POSTS request with ID=999 param
33	Verify GET /POSTS request with empty ID
34	Verify GET /POSTS request with IDs 1, 2
35	Verify GET /POSTS request with IDs 50, 100
36	Verify GET /POSTS request with IDs 10, 999
37	Verify create post option
38	Verify post can't be created with empty body
39	Verify post can't be created without userId field
40	Verify post can't be created without title field
41	Verify post can't be created without body field
42	Verify post can't be created with invalid userId: string
43	Verify post can't be created with invalid userId: number as string
44	Verify post can't be created with invalid userId: does not exist
45	Verify post can't be created with invalid userId: empty
46	Verify post can't be created with invalid userId: null
47	Verify post can't be created with invalid title: empty
48	Verify post can't be created with invalid title: wrong type
49	Verify post can't be created with invalid title: null
50	Verify post can't be created with invalid body: empty
51	Verify post can't be created with invalid body: wrong type
52	Verify post can't be created with invalid body: null

Test Case	Description
53	Verify post can't be created with unknown fields
54	Verify delete post option
55	Verify delete post option with invalid ID: does not exist
56	Verify delete post option with invalid ID: invalid
57	Verify patch post option: all fields
58	Verify patch post option: title only
59	Verify patch post option: body only
60	Verify patch post option: userId only
61	Verify resource can't be patched with empty body
62	Verify resource can't be patched with invalid title: empty
63	Verify resource can't be patched with invalid title: null
64	Verify resource can't be patched with invalid title: wrong type
65	Verify resource can't be patched with invalid body: empty
66	Verify resource can't be patched with invalid body: null
67	Verify resource can't be patched with invalid body: wrong type
68	Verify resource can't be patched with invalid field
69	Verify not existing resource PATCH status code
70	Verify update post option
71	Verify resource can't be updated with empty body
72	Verify resource can't be updated with missed field: userId
73	Verify resource can't be updated with missed field: title
74	Verify resource can't be updated with missed field: body
75	Verify resource can't be updated with wrong type field: userId
76	Verify resource can't be updated with wrong type field: title
77	Verify resource can't be updated with wrong type field: body
78	Verify DELETE /POSTS request not allowed
79	Verify PUT /POSTS request not allowed
80	Verify PATCH /POSTS request not allowed

Test Data and Scenarios Tested

Basic Functionality:

- Validating the standard operation of GET requests without parameters.
- Testing GET requests with known and unknown parameters to assess response handling.

Parameter Specific Tests:

- GET requests with specific parameters like id, userId, title, and combinations thereof.
- Scenarios where parameters partially match, mismatch, or are entirely valid/invalid.
- Examination of behavior with empty, negative, and non-numeric values in parameters.

POST Request Validations:

- Creating posts with valid and invalid data, including missing fields and incorrect data types.
- Testing the creation of posts with additional, unknown, or malformed fields.

DELETE, PATCH, and PUT Operations:

- Verifying the functionality of DELETE, PUT, and PATCH methods, including handling of invalid or non-existent resource IDs.
- Specific tests for PUT and PATCH operations targeting individual fields and combinations.

Data Validation and Security:

- Confirming the integrity and accuracy of data returned in responses.

Error Handling and Edge Cases:

- Evaluating the API's response to invalid requests and unsupported operations.
- Focused testing on edge cases and unusual combinations of request parameters.

Method Restrictions:

- Confirming the API's adherence to allowed HTTP methods, specifically testing the response to not allowed methods like DELETE, PUT, and PATCH on certain endpoints.

Test Results

! Note: We usually don't do this manually due the amount of test cases and issues - we attach links to corresponding test runs/launches from Allure TestOps or Testrail since they already contain all necessary statistics and metrics with open issues in Jira.