Test Summary Report

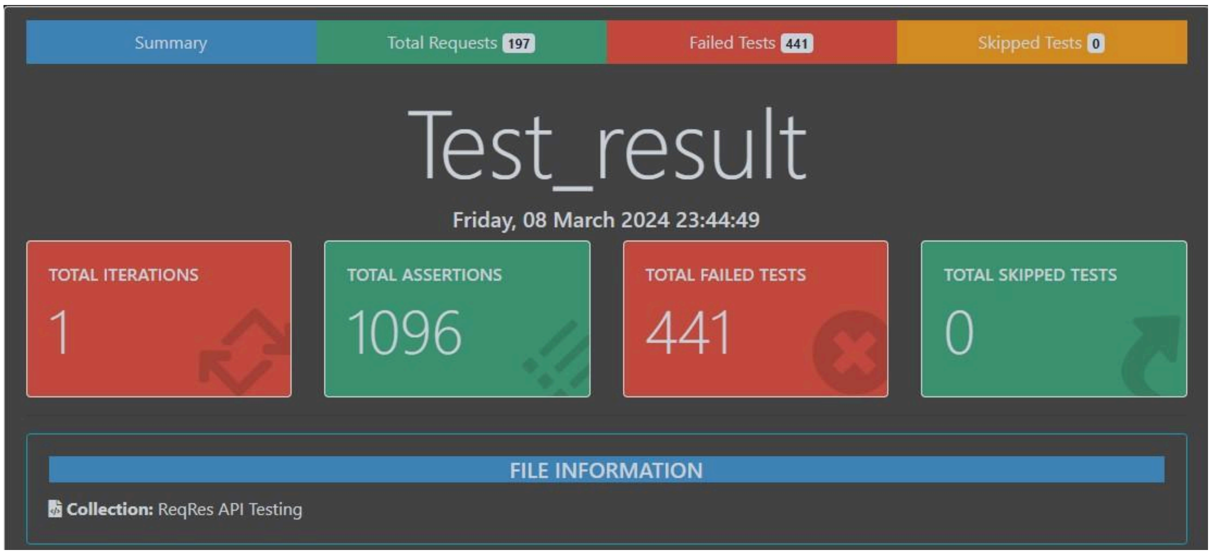
Project Name: Reqres.in API Testing

Test Date: 8.03.2024

Test Objective:

To evaluate the behavior and functionality of the Reqres.in API endpoints (/api/users, /api/unknown, /api/register, /api/login) with a focus on CRUD operations, user registration, and login functionality.

Test Result:



Summary | Total Requests 197 | Failed Tests 441 | Skipped Tests 0

Test_result

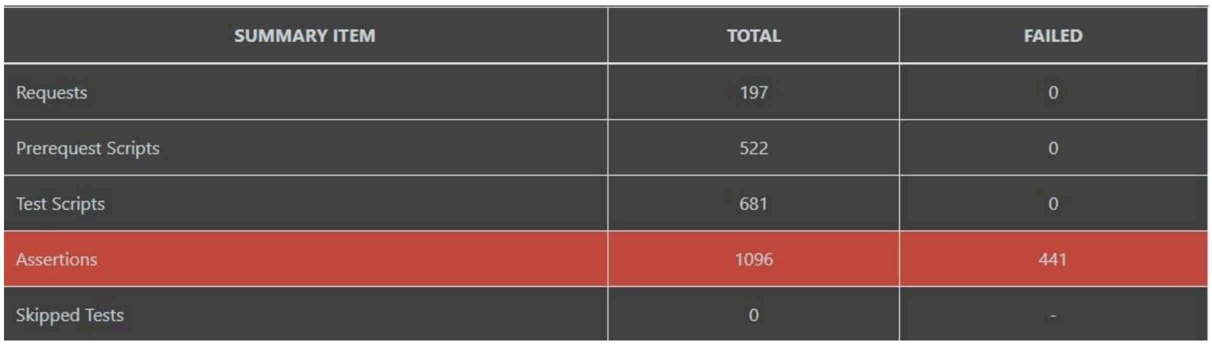
Friday, 08 March 2024 23:44:49

TOTAL ITERATIONS	TOTAL ASSERTIONS	TOTAL FAILED TESTS	TOTAL SKIPPED TESTS
1	1096	441	0

FILE INFORMATION

Collection: ReqRes API Testing

Figure: Test Report with total_assertions, total_failed tests



SUMMARY ITEM	TOTAL	FAILED
Requests	197	0
Prerequest Scripts	522	0
Test Scripts	681	0
Assertions	1096	441
Skipped Tests	0	-

Figure: Test Report with requests, test_scripts and assertions

From the provided data, it can be deduced that a total of 197 HTTP requests were made to the resreq.in API for testing purposes. These requests encompassed 1096 assertions, indicating specific criteria that were tested against the API's functionality. However, out of these assertions, 441 tests failed.

To provide a comprehensive summary:

- **Pass Rate**: The remaining 655 assertions were successful, resulting in a pass rate of approximately 59.7%.
- **Fail Rate**: Conversely, the failed tests accounted for approximately 40.3% of the total assertions tested.

In summary, while a significant portion of the assertions passed during the testing phase, there were still a considerable number of failures of high severity that need to be addressed and rectified.

Discrepancies:

1. CRUD Operations
    - Identified discrepancies related to retrieving, creating, updating, and deleting users/resources.

- Inconsistencies in error handling, data validation, and response codes were observed, also incorrect method support.

2. Registration
    - Discovered issues with user registration, including limited acceptance criteria and lack of data validation.
    - Insufficient error handling and incorrect method support.
3. Login
    - Noted discrepancies in login functionality, including unexpected behavior with incorrect methods and insufficient error handling.

CRUD Operations

1. Retrieving Users/Resources
    - Setting negative or exceedingly large values for page query params should return a "400 Bad Request" error instead of a "200 OK" response.
    - Invalid method usage (POST instead of GET) along with page and per_page parameters should return a "405 Method Not Allowed" error instead of a "201 Created" status code.
2. Retrieving User/Resource by ID
    - For a non-existent ID, the API should provide a more descriptive error message.
    - Negative ID should result in a "400 Bad Request" status code instead of a "404 Not Found" status code. Error messages should be provided instead of an empty body.
3. Creating Users/Resources
    - Inconsistent behavior in returning ID as a string in the response but returning a "404 Not Found" error upon retrieval of the created user.
    - Acceptance of duplicate creation requests without indicating that the user/resource already exists.
    - Lack of proper schema validation allowing invalid fields like "Job" and not providing proper error messages.
    - Successful creation response, but unsuccessful verification using a GET request.
    - API allows creation/update with unknown parameters, responding with a "200 OK" status instead of a "400 Bad Request" status.
    - Lack of data validation allowing invalid data formats, empty data and exceedingly large data.
4. Updating Users/Resources
    - Similar inconsistencies and lack of data validation as observed in creation operations.

    - Successful update response, but unsuccessful verification using a GET request.
5. Deleting Users/Resources
    - Successful deletion response, but unsuccessful verification using a GET request.

Registration:

- Registration only works for users defined in the user list of the api.
- Error returned when registering with username, defined user email address and password, which is not desirable behavior.
- Lack of password data validation, allowing any type of data.
- Support for incorrect method (PUT) in login, returning unexpected fields.

Login:

- Successful login for users in the list, but unsuccessful for unknown users.
- Correct error messages for missing email or password fields.
- Support for incorrect method (PUT) in login, returning unexpected fields.

Findings:

1. **Schema Validation Failure:**

    The database schema does not synchronize with the model definition, leading to schema validation failures. This discrepancy prevents the API from creating, updating, or deleting users effectively.

2. **Limited CRUD Operations:**

    The API lacks proper implementation of CRUD (Create, Read, Update, Delete) operations. Only the GET operation for retrieving a fixed list of users appears to be functional, indicating a deficiency in database interaction and data manipulation.

3. **Missing JSON Validation:**

    Server-side JSON validation is not implemented, leaving the API vulnerable to invalid data inputs and potential security risks.

4. **Inadequate Error Management:**

    Error handling is subpar, with inconsistent error message formats and a lack of error responses for various endpoints, especially for unknown endpoints. Improved error management, including standardized JSON-formatted error messages, is necessary for better developer experience and troubleshooting.

5. **Non-compliance with Data Restrictions:**

    The API allows the creation and updating of data with non-integer IDs, violating basic data restrictions and potentially causing data integrity issues. JSON validation should enforce stricter data validation rules to prevent such anomalies.

6. **Lack of CRUD Functionality:**

    Although the API returns responses indicating successful creation or updating of data, it fails to perform these operations effectively, highlighting a fundamental lack of CRUD functionality implementation.

7. **Inconsistent Method Handling:**

    The API allows methods that are not compliant with REST standards. For instance, the endpoint "https://reqres.in/api/users?page=1&per_page=6" responds to POST requests with a status code of 201 (Created), indicating successful creation. This behavior is inconsistent with the intended usage of the API and violates the principle of method restriction.

Recommendations

CRUD Operations:

1. Sync Schema with Model Definition:
    - Ensure alignment between the database schema and model definitions to facilitate accurate schema validation and enable seamless CRUD operations.

2. Implement Full CRUD Operations:

- Develop and integrate comprehensive CRUD functionality to enable efficient management of user data, including creation, retrieval, updating, and deletion operations.

3. Enforce JSON Validation:
    - Implement server-side JSON validation mechanisms to validate incoming requests and ensure data integrity and security.

4. Enhance Error Management:
    - Implement consistent error handling strategies across all endpoints, providing meaningful error messages in JSON format for improved developer understanding and error resolution.

5. Enforce Data Restrictions:
    - Enforce strict data validation rules, including ID format restrictions, to maintain data integrity and adhere to best practices in data management.

6. Improve CRUD Functionality Implementation:
    - Enhance the implementation of CRUD operations to ensure that data creation, updating, and deletion actions are performed accurately and reliably.

Schema Design Best Practices:

1. Primary Keys and Auto-increment
    - Consider making the ID field the primary key and auto-incremented for both the user and unknown schemas. This ensures uniqueness and simplifies data management, adhering to best practices in database design.
2. Unique Fields
    - Make the email field a required and unique field for the user schema to identify unique users efficiently and prevent duplicate entries. Similarly, consider making the name field a required and unique field for the resource schema to ensure accurate identification and management of resources.

Registration

User Registration Process

- Users must register with the API by providing a unique username, valid email, and secure password. Additionally, the system must prevent existing users from registering again, returning appropriate error messages when necessary.

- Strengthen data validation mechanisms to verify the integrity of user-provided data, particularly the password field. Implement checks to ensure that passwords meet minimum security criteria and reject non-compliant inputs.
- Enhance the registration functionality to reject unsupported HTTP methods, such as PUT requests, which may lead to unintended behavior or security vulnerabilities. Ensure that the API strictly adheres to HTTP method specifications to maintain consistency and security.

Login

Enhanced Login Functionality

- Enhance the login functionality to reject unsupported HTTP methods, such as PUT requests, which may lead to unintended behavior or security vulnerabilities. Ensure that the API strictly adheres to HTTP method specifications to maintain consistency and security.
- Improve error handling for login attempts, providing clear and descriptive error messages for common failure scenarios, such as incorrect credentials or missing fields. Informative error messages help users troubleshoot login issues effectively and enhance the overall user experience.

Common

Error Handling and Validation

- Implement robust error handling mechanisms to ensure consistent responses for invalid requests, such as negative page values and unknown parameters. Responses should include informative error messages and appropriate HTTP status codes (e.g., 400 Bad Request).
- Enhance data validation procedures to enforce stricter checks on input data formats, preventing the acceptance of invalid or exceedingly large

data. Implement validation rules based on the defined schemas to maintain data integrity.

Consistency in Response

- Ensure consistency in response formats across all CRUD operations. Responses should adhere to defined API standards, providing clear and predictable outputs to consumers. Inconsistent behavior, such as returning different data types for IDs or unexpected fields, should be addressed to improve usability and clarity.

Test Conducted By: Surraiya Islam Tonni

Date of Report: 08.03.2024