Test Plan: Negative Testing for GET Requests Against NASA API Services

TABLE OF CONTENTS

1. INTRODUCTION

This Test Plan outlines the approach and test cases for performing negative testing on the NASA API services that allow only GET requests. The primary goal of negative testing is to validate the robustness and error-handling capabilities of the API when faced with invalid, malformed, or unexpected inputs and scenarios.

Negative testing is a crucial aspect of API testing as it helps uncover potential vulnerabilities, edge cases, and areas for improvement in the API's design and implementation. By simulating adverse conditions and erroneous inputs, negative testing ensures that the API responds appropriately, returns meaningful error messages, and maintains its integrity and security.

The negative test cases covered in this Test Plan will be executed using Postman, a widely adopted API testing tool. Postman provides a user-friendly interface and powerful scripting capabilities, making it an ideal choice for constructing and automating various negative test scenarios.

The Test Plan will cover a range of negative test cases, including but not limited to:

- Sending requests to invalid or non-existent URLs
- Missing or invalid request parameters
- Unauthorized access attempts
- Invalid Query Parameter Values
- Boundary Value Testing
- Negative testing with headers, query strings, and character encodings
- Negative testing for requests for non-existent resources

By executing the negative test cases outlined in this Test Plan, we aim to ensure that the NASA API services respond appropriately to unexpected or erroneous inputs, providing meaningful error messages and maintaining data integrity and security. The insights gained from negative testing will contribute to enhancing the overall quality, reliability, and user experience of the API.

2. TESTING SCENARIOS

2.1 Invalid Endpoint: Use an invalid endpoint or URL.

- Use a random or non-existent endpoint.
- Modify the existing endpoint by adding extra characters or misspelling it.

The API should return a `404 Not Found` status code.

This test will help validate how the API handles invalid requests and ensure that appropriate error messages or responses are returned.

—--------------------------------

2.2 Unauthorized Access: If the API requires an API key or token for authorization, try sending a request without it or with an invalid key/token. The API should return a `401 Unauthorized` or `403 Forbidden` status code.

This test will help validate the API's access control mechanisms.

—---------------------------

2.3 Parameters.

2.3.1 Missing Required Query Parameters: If the API requires certain query parameters, try sending a request without them. The API should return a `400 Bad Request` status code.

2.3.2 Invalid Query Parameter Values: If the API expects certain values for a query parameter, try sending a request with invalid values. The API should return a `400 Bad Request` status code.

2.3.3 Extra Query Parameters: Add some extra query parameters that are not required by the API. The API should ignore these and still return a `200 OK` status code.

Example,

- if an endpoint expects a parameter called "id" with a numeric value, try sending a request with a non-numeric or negative value for the "id" parameter;
- if the API requires a `date` parameter, send a request to `https://techport.nasa.gov/some_endpoint` without the `date` parameter;
- if the API expects the `date` parameter to be in the format `YYYY-MM-DD`, send a request with an invalid date format like `YYYY-MM`.

This will help validate the API's parameter validation and error-handling mechanisms.

—----------------------------------

2.4 Resource Not Found: Send a GET request to an existing URL endpoint but with an invalid resource identifier (e.g., an ID that doesn't exist in the system).
This test will help validate how the API handles requests for non-existent resources.

—---------------------------------

2.5 Server Error: Simulate a server error by sending a request to an endpoint that causes the server to crash or return a 500 Internal Server Error.

Example:

The `date` parameter is to be in the format `YYYY-MM-DD`. Send 2024-05-00.

—--

2.6 Boundary Value Testing: If the API accepts numerical or range-based parameters, test scenarios with boundary values (e.g., minimum, maximum, or invalid ranges) to ensure the API handles these cases correctly.

Example:
- Send very large or very small values for numeric parameters.
- Send empty strings or null values for string parameters.

—---

2.7 Negative Testing with Encoding: Send requests with different character encodings or special characters to test the API's handling of such scenarios.

—--

2.8 Method Not Allowed: Try sending a different request method like POST or DELETE to the API. The API should return a `405 Method Not Allowed` status code.

—---

2.9 Network failures:
Simulate network errors (e.g., timeout, connection refused) by disconnecting from the internet or using a proxy tool.

—--

2.10 Content Negotiation: If the API supports content negotiation, try requesting a content type that the API does not support. The API should return a `406 Not Acceptable` status code.

Example:

1. Open Postman and create a new GET request.
2. Enter the URL of the API endpoint you want to test.
3. Go to the Headers tab.
4. Add a new header with \`Key\` as \`Accept\` and \`Value\` as a content type that the API does not support. For example, you could use \`application/xml\` if the API only supports \`application/json\`.
5. Send the request.