JANUARY 1, 2023

API Test Plan

Fake rest API

MD. SADIQUZZAMAN (SADIQ)
SQA ENGINEER
Dhaka, Bangladesh

Document History

Version	Date	Author	Description of Change
V1			Draft
V2			Draft- Reviewed

Prepared BY: Md. Sadiquzzaman (Sadiq)

Table of Contents

Objective:

This test plan aims to ensure the quality, functionality, and reliability of the Fake rest API hosted at https://fakerestapi.azurewebsites.net/index.html. The API is designed to handle Books, Authors, Cover Photos, Users, and Activities for a book management system.

API Description:

❖ Create (POST) Operation:

- Test the API's ability to create new authors, books, users, and others using valid input data.
- Verify that appropriate error responses are returned for invalid or missing data.
- Validate that newly created data is stored correctly in the system.

❖ Read (GET) Operation:

- Test the API's ability to retrieve data by various criteria (e.g., user ID, author mane, book name).
- Verify that the API returns the correct data in response to read requests.
- Test for correct handling of non-existent or invalid data.

❖ Update (PUT) Operation:

- Test the API's ability to update existing data with valid new data.
- Verify that the API rejects invalid update requests with appropriate error responses.
- Validate that the data is correctly modified in the system after updates.

❖ Delete (DELETE) Operation:

- Test the API's ability to delete data by providing valid criteria (book or user IDs).
- Verify that the API returns appropriate responses after successful deletion.
- Validate that the deleted data are removed from the system.

Scope:

Scope of Test Plan for FakeRESTApi:

❖ Functional Testing:

- Verify the correctness and functionality of all API endpoints as per the API documentation.
- Test various scenarios for creation, modification, and deletion.
- Validate user authentication and authorization mechanisms for protected endpoints.

❖ Data Validation Testing:

- Ensure that the API correctly validates input data, rejecting invalid requests.
- Test boundary values for input fields to check for any unexpected behavior.
- Validate the accuracy of data returned in responses.

❖ Error Handling Testing:

- Verify that appropriate error codes and messages are returned for invalid requests.
- Check error responses for sensitive information disclosure.
- Validate the API's ability to handle unexpected errors gracefully.

❖ Performance Testing:

- Assess the API's response time under normal and peak loads to identify potential bottlenecks.
- Measure the API's throughput and scalability to handle concurrent requests.

❖ Security Testing:

- Conduct security assessments to identify vulnerabilities such as SQL injection, XSS, etc.
- Validate the API's compliance with secure data transmission practices (e.g., HTTPS).
- Check for proper access controls to prevent unauthorized access to sensitive resources.

❖ Integration Testing:

- Verify interactions between different API endpoints and services.
- Test data consistency across related endpoints.

❖ Compatibility Testing:

- Test the API on different platforms, browsers, and devices to ensure cross-compatibility.

❖ Regression Testing:

- Conduct regression testing after bug fixes or updates to ensure existing functionality remains intact.

❖ Concurrency Testing:

- Assess the API's behavior when multiple users attempt to access and modify bookings simultaneously.

❖ Usability Testing:

- Evaluate the API's user-friendliness and ease of use from a developer's perspective.

❖ Performance Testing:

- Implement monitoring to track API performance in real time.

❖ Load Testing:

- Evaluate the API's behavior under high concurrent user loads to ensure stability.

❖ Rate Limit Testing:

- Check the API's adherence to rate-limiting rules to prevent abuse.

❖ Third-Party Integration Testing:

- Validate any third-party integrations for smooth functioning.

❖ Documentation Review:

- Assess the clarity, completeness, and accuracy of the API documentation.
- Verify that the API documentation is in sync with the actual API behavior.

Test Environment:

- The operating systems and versions that will be used for testing, such as Windows 10, MacOS, or Linux.
- The browsers and versions that will be used for testing, such as Google Chrome, Mozilla Firefox, or Microsoft Edge.
- The testing device types and screen sizes, such as desktop computers, laptops, tablets, and smartphones.
- The network connectivity and bandwidth that will be available for testing, such as Wi-Fi, cellular, or wired connection.
- The hardware and software requirements for running the test cases, such as specific processor, memory, or storage capacity.
- The security protocols and authentication methods that will be used to access the test environment, such as passwords, tokens, or certification.

The access permission and role of the team members who will be using the test environment, such as testers, developers, or stakeholders.

Name	Credential	Environment
Developer		
QA	Username: Password:	pre-staging
QA/Stakeholders	Username: Password:	staging
End User		

Test Data:

- Test data (valid, invalid) can be manually created by the testing team to cover specific scenarios and edge cases.
- Automation scripts can be used to generate large volumes of test data quickly and efficiently.
- Test date can be manually or automatically created based on API schemas.
- When testing API integrations, mock APIs can provide synthetic data for testing various scenarios.
- For specific use cases, third-party data providers can be used to obtain real-world data for testing.
- In some cases, sanitized data from the production environment can be used for testing to replicate real-world scenarios.

Test Strategy:

Step-1:

- The first step is to create test scenarios and test cases for the various feature in scope. While developing test cases, we will use a number of test design techniques.
  - Equivalence Class Partition
  - Boundary Value Analysis
  - Decision Table Testing
  - Static Transition Testing
  - Use Case Testing
- We also use our expertise in creating Test Cases by applying the below:
  - Error Guessing
  - Exploratory Testing
- We also prioritize the test cases.

Step-2:

- First, we will conduct Smoke Testing to see if the various and important functionalities of the application are working.
- We reject the build, if the Smoke Testing fails and will wait for the stable build before performing in-depth testing of the functionalities.
- Once we receive a stable build, which passes Smoke Testing, we perform in-depth testing using the Test Cases created.
- Multiple Test Resources will be testing the same application on Multiple Environments simultaneously.
- We then report the Bugs in Bug Tracking tool and send Dev. management on that day.
- As part of testing, we will perform the below types of testing:
  - Smoke Testing and Sanity Testing
  - Regression Testing and Retesting
  - Usability Testing, Functionality Testing
- We repeat the Test Cycle until we ensure the quality.

Step-3:

We will follow the below best practices to make our testing better:

- **Context-Driven Testing** – We will be performing testing as per the context of the application.

- **Shift Left Testing** – We will start testing from the beginning stages of the development itself, instead of waiting for the sable built.
- **Exploratory Testing** – Using our expertise we will perform Exploratory Testing, apart from the normal execution of the Test cases.
- **End-to-End Flow Testing** – We will test the end-to-end scenario which involves multiple functionalities to simulate the end-user flows.

Defect Reporting Procedure:

- The criteria for identifying a defect, such as deviation from the requirements, user experience issues, or technical errors.
- The steps for reporting a defect, such as using a designated template, providing detailed reproduction steps, and attaching screenshots or logs.
- The process for triaging and prioritizing defects, such as assigning severity and priority levels, and assigning them to the appropriate team members for investigation and resolution.
- The tools (JIRA/Test Rail) will be used for tracking and managing defects.
- The roles and responsibilities of the team members involved in the defect reporting process, such as testers, developers, and the test lead.
- The communication channels and frequencies for updating stakeholders on the progress and status of defects.
- The metrics that will be used to measure the effectiveness of the defect reporting process, such as the number of defects found, the time taken to resolve them, and the percentage of defects that were successfully fixed.

Defect and Reporting Person:

Defect Process	POC
Front End	Mr. X
Back End	Mr. Y
Dev Ops	Mr. Z
Mobile App	Mr. M

Test Schedule:

Two sprints (One Sprint = 5 working days) will be needed to test the application.

Following is the test schedule planned for the project –

Task Time Duration

No	Task	Date	Total Days
1	Creating Test Plan		
2	Test Case Creation		
3	Test Case Execution		
4	Summary Report Submission Date		

Test Coverages:

Test coverage ensures all relevant functionalities of the API are properly tested. It measures the effectiveness and completeness of the testing effort. Coverage Criteria for API Testing:

- **Functional Coverage:**
  - This criterion focuses on testing all the functional requirements of the API as specified in its documentation and design.
  - Each endpoint and method should be tested with various valid and invalid inputs to verify its behavior and responses.

- **Data Validation Coverage:**
  - This criterion focuses on validating the correctness of data returned by the API.

- **Error Handling Coverages:**
  - Different error scenarios, such as invalid inputs, server errors, and rate-limiting responses, should be included in the testing.
  - The API should be thoroughly tested to ensure it handles errors and provides appropriate error messages with proper status codes.

- **Performance Coverages:**
  - Test cases should be created to simulate various user loads and measure the API's response times and resource usage.

- **Regression Coverages:**
  - Selected test cases from various coverage criteria are reused to validate the unchanged parts of the API.

Risk and Mitigation:

Below are some common risks and mitigation strategies to address them.

No	Risk	Mitigation
1.	Poor or outdated API documentation	Prioritize obtaining up-to-date and accurate API documentation
2.	Unclear error messages	Implement extensive negative testing to validate error-handling scenarios
3.	Non-Availability of a Resource	Backup Resource Planning
4.	APIs relying on external APIs or services can introduce uncertainties due to factors beyond the API being tested	Use mock APIs or virtualized services to simulate the behavior of external dependencies during testing.
5.	Changes or updates to the API can introduce regressions that affect existing functionalities or break compatibility with dependent applications.	Execute regression tests after each update and version release to ensure backward compatibility and identify regression issues promptly.
6.	Less time for testing	Ramp up the resources based on the Client's needs dynamically
7.	Insufficient communication and collaboration between testers, developers, and stakeholders can result in misunderstandings, missed requirements	Foster open communication channels and hold regular meetings between testing and development teams. Engage stakeholders early in the testing process to gather feedback and clarify requirements.

Testing Tools:

- Postman – API Testing
- Newman – API Automation & Test Reporting
- JMeter – Load & Performance Testing
- Burp Suite – Security Testing
- JIRA – Bug Tracking Tools
- Light short – Screenshot Tools
- Loom – Screen Recording
- MS Word & Excel - Documentation

Roles and Responsibilities:

Name	Role	Responsibilities
	(SQA) Lead	- Develop test plan - Define the scope and objectives - Assign tasks to the testing team and monitor progress - Review and approve test cases and test data - Report testing progress and results to stakeholders
	SQA Engineer	- Create, manage, and maintain test cases - Create, manage, and maintain test data and ensure that test data is relevant, accurate, and covers various scenarios - Collaborate with API testers to understand data requirements. - Provide feedback to the QA Lead regarding testing progress and challenges. - Develop and maintain automated test scripts for API testing.
	Jr SQA Engineer	- Execute test cases and record test results. - Validate API functionality, performance, and security aspects - Identify, report, and track defects in a defect-tracking system - Collaborate with developers to reproduce and verify reported defects.

Test Deliverables:

The followings are to be delivered to the client:

Deliverables	Description	Completion Date
Test Plan	Details on the scope of the project, test strategy, test schedule, and test deliverables.	
Functional Test Cases	Test cases created for the scop defined.	
Defect Report	A detailed description of the defects identified along with screenshots and step to reproduce on a daily basis.	
Summary Report	Summary report of the project	

Approvers List:

No	Name	Role	Approver Review	Date