**Test Plan Document**

**1. Introduction**

**1.1 Objective**

The purpose of this test plan is to ensure the functionality, performance, and reliability of the RESTful API provided at https://fakerestapi.azurewebsites.net. The focus will be on validating all endpoints, including positive and negative test cases, at system, integration, and end-to-end levels. Performance testing will also be included to assess the API's robustness under load.

**2. Scope**

**2.1 In Scope**

* Testing of API endpoints provided in the Swagger documentation.
* Validation of all HTTP methods (GET, POST, PUT, DELETE).
* Testing both valid and invalid inputs.

**2.2 Out of Scope**

* UI Testing (if any).
* Database-level testing or back-end performance optimization.
* Performance testing using a load testing tool.

**3. Testing Levels**

**3.1 System Testing**

* Verify each API endpoint independently with positive and negative test cases.
* Ensure correct responses (status codes, headers).

**3.2 Integration Testing**

* Validate how multiple endpoints work together, e.g., relationships between Authors and Books.

**3.3 End-to-End Testing**

* Test complete user flows such as creating, updating, retrieving, and deleting a resource.

**3.4 Performance Testing**

* Measure API response times under various levels of load.
* Assess the API's scalability and stability under stress.

**4. Test Environment**

**4.1 Test Environment Details**

* **Base URL**: https://fakerestapi.azurewebsites.net
* **Tool**: Postman (for functional testing), Newman (for automated testing), JMeter (for performance testing)
* **Environment Variables**:
  + baseUrl: <https://fakerestapi.azurewebsites.net>
  + authToken: *(if required for authentication)*

**5. Test Cases**

**5.1 Test Case Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Endpoint | Method | Test Description | Input Data | Expected Result | Priority |
| PT001 | /api/v1/Activities | GET | Measure response time under 1000 concurrent requests | None | Avg response time < 2s | High |
| PT002 | /api/v1/Books | POST | Ensure server can handle 50 simultaneous creations | Batch data | All requests succeed with 201 Created | High |
| TC001 | /api/v1/Activities | GET | Retrieve all activities | None | 200 OK, list of activities in JSON | High |
| TC002 | /api/v1/Activities/{id} | GET | Retrieve activity by valid ID | Valid ID (e.g., 1) | 200 OK, details of the activity in JSON | High |
| TC004 | /api/v1/Activities | POST | Create activity with valid data | { "id": 1, "title": "Activity 1" } | 201 Created, JSON response of new activity | High |
| TC005 | /api/v1/Activities | POST | Attempt to create activity with missing fields | { "title": "" } | 400 Bad Request, error message | High |
| TC006 | /api/v1/Activities/{id} | PUT | Update activity with valid data | Valid ID, { "title": "Updated" } | 200 OK, updated JSON response | High |
| TC008 | /api/v1/Activities/{id} | DELETE | Delete activity with valid ID | Valid ID (e.g., 1) | 200 OK, deletion confirmation | High |
| TC010 | /api/v1/Authors | GET | Retrieve all authors | None | 200 OK, list of authors in JSON | High |
| TC011 | /api/v1/Authors/{id} | GET | Retrieve author by valid ID | Valid ID (e.g., 1) | 200 OK, author details in JSON | High |
| TC013 | /api/v1/Authors | POST | Create author with valid data | { "id": 1, "name": "Author 1" } | 201 Created, JSON response of new author | High |
| TC014 | /api/v1/Authors | POST | Attempt to create author with missing fields | { "name": "" } | 400 Bad Request, error message | High |
| TC015 | /api/v1/Authors/{id} | PUT | Update author with valid data | Valid ID, { "name": "Updated" } | 200 OK, updated JSON response | High |
| TC016 | /api/v1/Authors/{id} | DELETE | Delete author with valid ID | Valid ID (e.g., 1) | 200 OK, deletion confirmation | High |
| TC017 | /api/v1/Books | GET | Retrieve all books | None | 200 OK, list of books in JSON | High |
| TC018 | /api/v1/Books/{id} | GET | Retrieve book by valid ID | Valid ID (e.g., 1) | 200 OK, book details in JSON | High |
| TC019 | /api/v1/Books | POST | Create book with valid data | { "id": 1, "title": "Book 1" } | 201 Created, JSON response of new book | High |
| TC020 | /api/v1/Books/{id} | PUT | Update book with valid data | Valid ID, { "title": "Updated" } | 200 OK, updated JSON response | High |
| TC021 | /api/v1/Books/{id} | DELETE | Delete book with valid ID | Valid ID (e.g., 1) | 200 OK, deletion confirmation | High |
| TC025 | /api/v1/Users | GET | Retrieve all users | None | 200 OK, list of users in JSON | High |
| TC026 | /api/v1/Users/{id} | GET | Retrieve user by valid ID | Valid ID (e.g., 1) | 200 OK, user details in JSON | High |
| TC027 | /api/v1/Users | POST | Create user with valid data | { "id": 1, "name": "User 1" } | 201 Created, JSON response of new user | High |
| TC028 | /api/v1/Users/{id} | PUT | Update user with valid data | Valid ID, { "name": "Updated" } | 200 OK, updated JSON response | High |
| TC029 | /api/v1/Users/{id} | DELETE | Delete user with valid ID | Valid ID (e.g., 1) | 200 OK, deletion confirmation | High |
| TC003 | /api/v1/Activities/{id} | GET | Attempt to retrieve activity with invalid ID | Invalid ID (e.g., 9999) | 404 Not Found, error message | Medium |
| TC007 | /api/v1/Activities/{id} | PUT | Attempt to update activity with invalid ID | Invalid ID, { "title": "Updated" } | 404 Not Found, error message | Medium |
| TC009 | /api/v1/Activities/{id} | DELETE | Attempt to delete activity with invalid ID | Invalid ID (e.g., 9999) | 404 Not Found, error message | Medium |
| TC012 | /api/v1/Authors/{id} | GET | Attempt to retrieve author with invalid ID | Invalid ID (e.g., 9999) | 404 Not Found, error message | Medium |
| TC022 | /api/v1/CoverPhotos | GET | Retrieve all cover photos | None | 200 OK, list of cover photos in JSON | Medium |
| TC023 | /api/v1/CoverPhotos/{id} | GET | Retrieve cover photo by valid ID | Valid ID (e.g., 1) | 200 OK, cover photo details in JSON | Medium |
| TC024 | /api/v1/CoverPhotos | POST | Add cover photo with valid data | { "id": 1, "url": "cover1.jpg" } | 201 Created, JSON response of new cover photo | Medium |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**6. Test Data**

**6.1 Sample Input Data**

**Activities**

* Valid Data: { "title": "New Activity" }
* Invalid Data: { "title": "" }

**Authors**

* Valid Data: { "id": 1, "name": "Author Name" }
* Invalid Data: { "id": "abc", "name": "" }

**Books**

* Valid Data: { "title": "Sample Book", "authorId": 1 }
* Invalid Data: { "title": "", "authorId": "abc" }

**7. Test Execution**

**7.1 Steps for Execution**

1. Import the Swagger file into Postman to generate requests automatically.
2. Add environment variables for baseUrl.
3. Execute each test case in the **Collection Runner**
4. Document results, highlighting Pass/Fail status.

**8. Reporting**

**8.1 Test Artifacts**

* Test Case Document (this document).
* Postman Collection Export.
* Test Execution Logs (from Postman).

**9. Assumptions & Risks**

**9.1 Assumptions**

* The API is fully functional and hosted on a stable server.
* Swagger documentation is accurate and up-to-date.

**9.2 Risks**

* Delays due to server downtime or inaccurate documentation.
* Lack of proper error messages for invalid inputs.

**10. Conclusion**

This test plan ensures comprehensive coverage of the RESTful API by validating all endpoints with positive and negative test cases. Execution will verify the API's functionality, reliability, and performance, providing actionable feedback to developers.