Test Plan

A test plan for the RESTful API is to ensure that all the functional and non-functional components of the API are working properly.

Test Strategy

 **Integration Testing**: Testing the integration of the API with the databases.

 **Functional Testing**: Validate API endpoints working as expected and check the API’s response with valid/invalid input.

 **Security Testing**: Validate authorization access for a user

 **Error Handling Testing**: Validate error messages and status codes are displaying correctly for invalid inputs/access issues etc..

 **Performance Testing**: Evaluate the response time and load performance of API

 **Regression Testing**: Ensure that new API changes does not break the existing functionality

High Priority Test Cases

1. Get all users (GET/activities)

2. Get a User by ID (GET /activities/{id})

3. Create a User (POST /activities)

4. Update a User (PUT /activities/{id})

5. Delete a User (DELETE /activities/{id})

Medium Priority Test Cases

1. Validate complete API process works as expected, with correct response status codes.

2. Validate the user has specific access to check the resources.

3. Validate whether multiple users are accessing the same endpoint

Low Priority Test Cases.

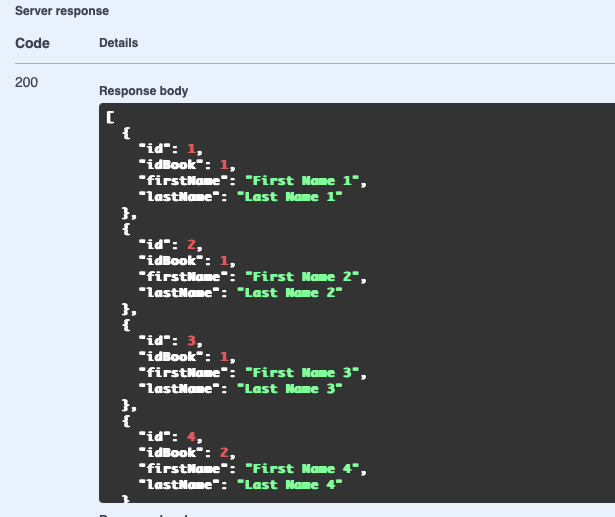
1. Validate the API endpoints by providing the invalid inputs

2. Validate the endpoints by missing fields

3. Validate the consistency of data if a user is deleted

1. Functional testing

* Positive testcase: Get all users GET /authors



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | **Test Case ID** | **Test Case Description** | **HTTP Method** | **Endpoint** | **Input data** | **Actual Outcome** | **Expected Outcome** | **Status (Pass/Fail)** | | Tc\_01 | Verify retrieval of all authors | GET | /authors | No input | 200 Success, list of authors | 200 OK, list of authors | Pass | |  |

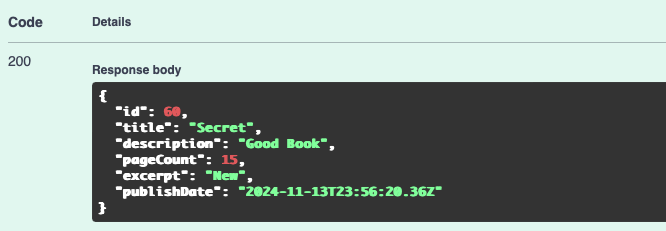
* Negative testcase: Create User with Missing Required Field POST /authors



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Description** | **HTTP Method** | **Endpoint** | **Input data** | **Actual Outcome** | **Expected Outcome** | **Status (Pass/Fail)** |
| Tc\_02 | Verify Create Operation with missing fields | POST | /authors | Missing required fields | 400 Bad Request, error message | 400 Bad Request, error message | Pass |

2. Integration Testing

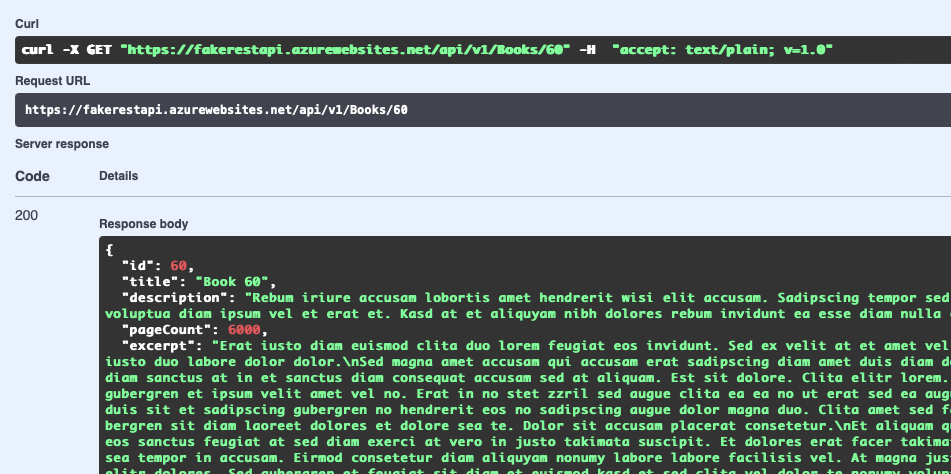
* Positive testcase: Create a user POST/books



Note: Whatever i have created it is not stored in the database

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Description** | **HTTP Method** | **Endpoint** | **Input data** | **Actual Outcome** | **Expected Outcome** | **Status (Pass/Fail)** |
| Tc\_03 | Verify Create Operation with valid data | POST | /books | Valid user data | 200 | 201 Created, valid response | Fail |

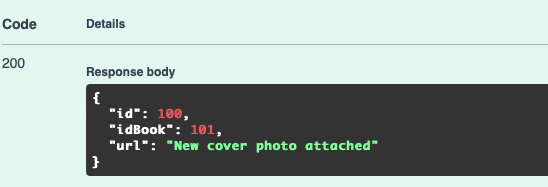
* Negative testcase: Get a user GET/books/60



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Description** | **HTTP Method** | **Endpoint** | **Input data** | **Actual Outcome** | **Expected Outcome** | **Status (Pass/Fail)** |
| Tc\_04 | Verify retrieval of a specific book by ID | GET | /books/{id} | Id=60 | 200, but the resource is not created | 200 OK, correct user data | Fail |

3. End-to-end Testing

* Testcases: Create a user, POST/coverphotos



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Description** | **HTTP Method** | **Endpoint** | **Input data** | **Actual Outcome** | **Expected Outcome** | **Status (Pass/Fail)** |
| Tc\_05 | Verify Create Operation with valid data | POST | /coverphotos | Valid user data | 200 | 201 Created, valid response | Fail |

4. Performance Testing

* Positive testcase: We can check by providing more no. of users for same endpoint and verify the system can handle the load.
* Negative testcase: Check whether the API is handling with more load

Postman Collection for CRUD:

#### GET Books

Endpoint: https://fakerestapi.azurewebsites.net/api/v1/Books

Method: GET

Description: Retrieves a list of all books.

Body(Response) (JSON Format):

Test: Check the status code is 200.

#### GET Book by ID

Endpoint: https://fakerestapi.azurewebsites.net/api/v1/Books/{id}

Method: GET

Description: Retrieves a specific book by its ID.

Body(Response) (JSON Format):

Test: Check the status code is 200.

#### POST Book

Endpoint: https://fakerestapi.azurewebsites.net/api/v1/Books

Method: POST

Description: Creates a new book.

Body(Response) (JSON Format):

Test: Check the status code is 201.

#### PUT Update Book

Endpoint: https://fakerestapi.azurewebsites.net/api/v1/Books/{id}

Method: PUT

Description: Updates an existing book's details.

Body(Response) (JSON Format):

Test: Check the status code is 200.

#### DELETE Book

Endpoint: https://fakerestapi.azurewebsites.net/api/v1/Books/{id}

Method: DELETE

Description: Deletes a specific book by its ID.

Test: Check the status code is 204.

Shift left policy:

1. With shift left policy, we can start testing as early as possible even before the development phase.

2. Start testing from the initial stages, so we can reduce the number of defects on later stages

3. Collaboration between multiple teams (developers, testers etc..) to ensure a quality product.

4. Run Automation testcases using CI/CD pipelines to catch the issues.