Test Scenarios

1. Positive Test Cases (Happy Path)

a. TC_001

Description: Verify the response body values regarding to the Schema. We should check the response body elements in detailed. Because we will Assert the response values with UI or/and Database values. That is why checking the response body's Key and Values are important for the assertion and verification.

Pre-conditions: User should be able to GET the values

Test Environment: Test

Test Data: BaseUrl, Search values

Test Steps:

1. Given Get request is working for the URL
2. When the user send Get request with query params ("q", "furry rabbits")
3. Then User should check response code (must be 200)
4. And User should check ContentType
5. Then User should verify the Arrays and Key Values regarding to the Schema rules

Expected Result: User should be able to confirm all the Object values relevant to the Schema
Actual Result: Should be checked
Test Result (Status): **Test Pass**
Created by: Emre Dincsoy
Executed by: Emre Dincsoy
Execution: MM/DD/YYYY

b. TC_002

Description: Verify the response Headers are relevant or not. We should check the Response headers. They should be relevant to the Schema and should be asserted after the requests.

Pre-conditions: User should be able to GET the values with endpoint

Test Environment: Test

Test Data: BaseUrl, Search values

Test Steps:

1. Given Get request is working for the URL
2. When the user send Get request with query params ("q", "furry rabbits")
3. Then User should check response code (must be 200)
4. Then User should verify the Key and Values of response Headers

Expected Result: User should be able to confirm all the Response Header values relevant to the structure
Actual Result: Should be checked
Test Result (Status): **Test Pass**
Created by: Emre Dincsoy
Executed by: Emre Dincsoy
Execution: MM/DD/YYYY

c. **TC_003**

Description: Verify the PUT operation if the user should be able to change the related values. The User should check the other CRUD operators (PATCH, DELETE, POST). They should work as expected. As an example I use PUT operator here. We should also check Delete request and check if the object deleted or not. We should check PATCH request if the user able to change the part of the body values or not. Lastly user should check POST request if new object should be created with 201 success code.

Pre-conditions: User should be able to send PUT request with endpoint

Test Environment: Test

Test Data: BaseUrl, Search values

Test Steps:

1. Given PUT request is working for the URL
2. When the user send PUT request with query and BODY
3. Then User should check response code (must be 200)
4. Then User should verify the Key and Values of response Headers
5. Then User should verify that the values have been changed

Expected Result: User should be able to confirm that the values have been changed
Actual Result: Should be checked
Test Result (Status): **Test Pass**
Created by: Emre Dincsoy
Executed by: Emre Dincsoy
Execution: MM/DD/YYYY

d. **TC_004**

Description: User should check if the GET request response body equals to Page Limit that has given. We should be sure if the page limit function works fine.

Pre-conditions: User should be able to send GET request with endpoint

Test Environment: Test

Test Data: BaseUrl, Search values

Test Steps:

1. Given Get request is working for the URL
2. When the user send Get request with query params (“q”, “furry rabbits”,” limit=5”)
3. Then User should check response code (must be 200)
4. Then User verifies Response body have 5 results

Expected Result: User should be able to confirm that the Response body have 5 results
Actual Result: Should be checked
Test Result (Status): **Test Pass**
Created by: Emre Dincsoy
Executed by: Emre Dincsoy
Execution: MM/DD/YYYY

Bonus (Optional)

a. To create a page via that API, you need to provide an Authorization Header. Describe how you would adapt testing/automation?

b. Assume we just created that public Wikipedia API. Describe which other aspects you would want to test and how.

a. Authorization Header

1. **Put the Appropriate Authentication Method to the framework:** There are different authentication methods that the web service or API requires. These Authorization types can be Basic Authentication, API keys, OAuth, Bearer tokens, or JWT (JSON Web Tokens).

For example, if the system uses API keys we have to add authorization header for every request to be able to get success response.

Here is an example for **API key**;

```
Response response = RestAssured.given()
        .log().all()
        .header("x-api-key","JbYpFvedNYBiDHSwmKiQ")
```

For Bearer token authentication, the header might look like this:

Authorization: Bearer your-access-token

2. **Error Handling**: Be prepared to handle authentication and authorization errors, such as 401 Unauthorized or 403 Forbidden responses. These errors indicate issues with the credentials or permissions.

3. **Refresh Tokens (if it was implemented)**: For certain authentication methods like OAuth or JWT, consider handling token expiration and renewal if the tokens have a limited lifespan. So while automation tests consider this time limit.

b. Describe which other aspects you would want to test and how.

1. **Endpoint Testing:** Test all of the CRUD operators (POST, PUT, PATCH, DELETE)

2. **Error Handling Testing:** Test negative test scenarios. Use unauthorized user credentials and check if you are getting 403,401 responses.

3. **Performance and Load Testing:** Evaluate the API's performance under different levels of load. This includes testing how it behaves under simultaneous requests and large data sets.

4. **Authentication and Authorization Testing:** Test the API's authentication mechanisms, ensuring that they work as intended. Validate that only authorized users can access restricted features. Confirm that the API enforces proper access control based on user roles and permissions.