Test-Case Table: JSONPlaceholder

This table outlines the test cases for the JSONPlaceholder API, separated by HTTP method.

GET /posts (Retrieve all posts)

Test Case ID	Test Scenario	Steps	Expected Result
GET_001	Retrieve all posts (positive)	Send a GET request to https://jsonplaceholder.typicode.com/posts	Status code 200 OK. The response body should be a JSON array containing 100 posts. **Verify the Content-Type header is application/json.**
GET_002	Verify response structure and data types	Send a GET request to /posts	Each object in the array should have userId (integer), id (integer), title (string), and body (string) fields.
GET_003	Filter posts by valid userId	Send a GET request to /posts?userId=1	Status code 200 OK. The response should be a JSON array containing only posts belonging to userId 1.
GET_004	Filter posts by non-existent userId (negative)	Send a GET request to /posts?userId=9999	Status code 200 OK. The response should be an empty JSON array [].
GET_005	Invalid query parameter (negative)	Send a GET request to /posts?invalid_param=test	Status code 200 OK. The API should ignore the invalid parameter and return all 100 posts.
GET_006	Malformed URL (negative)	Send a GET request with a typo in the endpoint (e.g., /postss)	Status code 404 Not Found.

GET /posts/{id} (Retrieve a specific post)

Test Case ID	Test Scenario	Steps	Expected Result
GET_007	Retrieve a specific post with valid ID (positive)	Send a GET request to /posts/1	Status code 200 OK. The response body should be a JSON object representing the post with id: 1. **Verify the Content-Type header is application/json.**
GET_008	Retrieve a post with non-existent ID (negative)	Send a GET request to /posts/9999	Status code 404 Not Found. The response body should be an empty JSON object {}.
GET_009	Retrieve a post with an invalid ID format (negative)	Send a GET request to /posts/abc	Status code 404 Not Found. The response body should be an empty JSON object {}.
GET_010	Retrieve a post with a negative ID (negative)	Send a GET request to /posts/-1	Status code 404 Not Found. The response body should be an empty JSON object {}.

POST /posts (Create a new post)

Test Case ID	Test Scenario	Steps	Expected Result
POST_001	Create a new post with valid data (positive)	Send a POST request to /posts with a valid JSON body (e.g., { "title": "foo", "body": "bar", "userId": 1 })	Status code 201 Created. The response body should contain the sent data and a unique id (e.g., id: 101). **Verify the Content-Type header is application/json.**
POST_002	Missing required fields (negative)	Send a POST request with a JSON body missing title	Status code 400 Bad Request. The API should return an error indicating the missing field.

		or userId	
POST_003	Invalid data type (negative)	Send a POST request with an integer value for body field	Status code 400 Bad Request.
POST_004	Empty JSON body (negative)	Send a POST request with a JSON body of {}	Status code 400 Bad Request.
POST_005	Invalid Content-Type header (negative)	Send a POST request with Content-Type: text/xml	Status code 415 Unsupported Media Type or 400 Bad Request.

PUT /posts/{id} (Update a post)

Test Case ID	**Test Scenario**	**Steps**	**Expected Result**
PUT_001	Update an existing post with valid data (positive)	Send a PUT request to /posts/1 with a complete JSON body for the update	Status code 200 OK. The response body should reflect the updated values. **Verify the Content-Type header is application/json.**
PUT_002	Update a post with non-existent ID (negative)	Send a PUT request to /posts/9999	Status code 200 OK. The API fakes the update and returns the request body with the non-existent ID.
PUT_003	Update a post with partial data (negative)	Send a PUT request to /posts/1 with only a title field in the JSON body	Status code 200 OK. The response should contain the updated title, but other fields may not be retained. (Note: PUT is for full resource replacement, so a PATCH request would be more suitable for partial updates. This is a good test to see how the API handles the PUT method).
PUT_004	Update a post with an invalid ID format	Send a PUT request to /posts/abc	Status code 404 Not Found.

	(negative)		
PUT_005	Update a post with an empty JSON body (negative)	Send a PUT request to /posts/1 with a JSON body of {}	Status code 200 OK. The response body should contain the sent data.

DELETE /posts/{id} (Delete a post)

Test Case ID	**Test Scenario**	**Steps**	**Expected Result**
DEL_001	Delete an existing post with valid ID (positive)	Send a DELETE request to /posts/1	Status code 200 OK. The response body should be an empty JSON object {}. **Verify the Content-Type header is application/json.**
DEL_002	Delete a post with a non-existent ID (negative)	Send a DELETE request to /posts/9999	Status code 200 OK. The API fakes the deletion and returns an empty JSON object.
DEL_003	Delete a post with an invalid ID format (negative)	Send a DELETE request to /posts/abc	Status code 404 Not Found.
DEL_004	Verify idempotency of delete operation	Send a DELETE request for the same resource multiple times	The first request should return 200 OK. Subsequent requests should also return 200 OK, confirming the API's behavior for repeated delete attempts.

Concurrency and Load Testing

Test Case ID	**Test Scenario**	**Steps**	**Expected Result**
LOAD_001	High-volume GET	Send 1000 concurrent GET	The API should maintain a consistent response time and return 200 OK

	requests	/posts requests	for all requests, without any 5xx server errors.
LOAD_002	Concurrent POST requests	Send 100 concurrent POST /posts requests	All requests should return a 201 Created status code, and the API should not fail.
LOAD_003	Rate-limiting test	Send a high volume of requests over a short period (e.g., 100 requests in 1 second)	If a rate-limiting mechanism is in place, the API should return a 429 Too Many Requests status code after a certain threshold.

Data Validation and Security

Test Case ID	**Test Scenario**	**Steps**	**Expected Result**
SEC_001	Cross-Site Scripting (XSS) attempt	Send a POST request with a title or body containing an XSS payload (e.g., <script>alert('XSS')</script>)	The API should sanitize the input and not execute the script in the response. The payload should be returned as a string.
SEC_002	SQL Injection attempt	Send a POST request with a title containing a SQL injection payload (e.g., ' OR '1'='1)	The API should handle the string as literal text and not attempt to execute it as a SQL query.
SEC_003	Boundary value for userId	Send a POST or PUT request with a userId that is a very large integer	The API should handle the large integer value without crashing or returning an error.