Test Cases Specification

JSONPlaceholder API Testing Project

Date: August 9, 2025

Author: Victor Murashev

1. Document Overview

1.1 Purpose

This document contains detailed test case specifications for the JSONPlaceholder API testing project. Each test case includes step-by-step instructions, test data, and expected results.

1.2 Test Case Template

```
Test Case ID: TC-XXX
Test Case Name: [Descriptive name]
Requirement ID: [Linked requirement]
Priority: High/Medium/Low
Test Type: Functional/Validation/Performance/etc.
Preconditions: [Setup requirements]
Test Steps: [Numbered steps]
Test Data: [Input data]
Expected Results: [Expected outcomes]
Postconditions: [Cleanup requirements]
```

2. Posts API Test Cases

2.1 Posts Retrieval Tests

TC-001: Get All Posts - Smoke Test

- **Requirement ID:** FR-001.1
- **Priority:** High
- **Test Type:** Smoke/Functional
- **Preconditions:** API service is available
- **Test Steps:**
  - Send GET request to `/posts`
  - Verify response status code
  - Verify response contains data
  - Verify response time
- **Test Data:** None required
- **Expected Results:**
  - Status code: 200
  - Response body contains array of posts
  - Response time < 2 seconds
  - Each post has id, title, body, userId fields
- **Postconditions:** None

TC-002: Get All Posts - Data Validation

- **Requirement ID:** FR-001.1
- **Priority:** High
- **Test Type:** Functional
- **Preconditions:** API service is available
- **Test Steps:**
  - Send GET request to `/posts`
  - Verify response structure
  - Validate data types
  - Check data completeness
- **Test Data:** None required
- **Expected Results:**
  - Response is valid JSON array

    - Array contains 100 posts
    - Each post has: id (number), title (string), body (string), userId (number)
    - No null or empty required fields
- **Postconditions:** None

TC-003: Get Single Post - Valid ID

- **Requirement ID:** FR-001.2
- **Priority:** High
- **Test Type:** Functional
- **Preconditions:** API service is available
- **Test Steps:**
    - Send GET request to `/posts/1`
    - Verify response status code
    - Verify response contains single post
    - Validate post data structure
- **Test Data:** Post ID = 1
- **Expected Results:**
    - Status code: 200
    - Response body contains single post object
    - Post has correct structure: id, title, body, userId
    - Post ID matches requested ID (1)
- **Postconditions:** None

TC-004: Get Single Post - Boundary Values

- **Requirement ID:** FR-001.2
- **Priority:** Medium
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
    - Send GET request to `/posts/1` (minimum valid ID)
    - Send GET request to `/posts/100` (maximum valid ID)
    - Verify both responses
- **Test Data:** Post IDs = 1, 100
- **Expected Results:**
    - Both requests return status code 200

    - Both responses contain valid post objects
    - IDs match requested values
- **Postconditions:** None

TC-005: Get Single Post - Invalid ID Format

- **Requirement ID:** FR-001.2
- **Priority:** Medium
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
    - Send GET request to `/posts/abc` (non-numeric ID)
    - Send GET request to `/posts/-1` (negative ID)
    - Send GET request to `/posts/0` (zero ID)
- **Test Data:** Post IDs = "abc", -1, 0
- **Expected Results:**
    - Status code: 404 or appropriate error status
    - Error response with meaningful message
- **Postconditions:** None

TC-006: Get Single Post - Non-existent ID

- **Requirement ID:** FR-001.3
- **Priority:** Medium
- **Test Type:** Error Handling
- **Preconditions:** API service is available
- **Test Steps:**
    - Send GET request to `/posts/999` (non-existent ID)
    - Verify error response
- **Test Data:** Post ID = 999
- **Expected Results:**
    - Status code: 404
    - Empty response body or appropriate error message
- **Postconditions:** None

2.2 Posts Creation Tests

TC-007: Create Post - Valid Data

- **Requirement ID:** FR-002.1
- **Priority:** High
- **Test Type:** Functional
- **Preconditions:** API service is available
- **Test Steps:**
  - Prepare valid post data
  - Send POST request to `/posts` with data
  - Verify response status and structure
  - Validate created post data
- **Test Data:** `{ "title": "Test Post Title", "body": "This is test post content", "userId": 1}`

- **Expected Results:**
  - Status code: 201
  - Response contains created post with assigned ID
  - All input data is preserved in response
  - Response time < 2 seconds
- **Postconditions:** None (read-only API)

TC-008: Create Post - Minimum Valid Data

- **Requirement ID:** FR-002.1
- **Priority:** High
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
  - Prepare minimal valid post data
  - Send POST request to `/posts`
  - Verify creation success
- **Test Data:** `{ "title": "T", "body": "B", "userId": 1}`

- **Expected Results:**
  - Status code: 201

    - Post created successfully
    - Minimal data accepted
- **Postconditions:** None

TC-009: Create Post - Missing Title

- **Requirement ID:** FR-002.2
- **Priority:** High
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
    - Prepare post data without title
    - Send POST request to `/posts`
    - Verify validation error
- **Test Data:** `{  "body": "This is test post content",  "userId": 1}`

- **Expected Results:**
    - Status code: 400 or 422
    - Error message indicating missing title
- **Postconditions:** None

TC-010: Create Post - Missing Body

- **Requirement ID:** FR-002.2
- **Priority:** High
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
    - Prepare post data without body
    - Send POST request to `/posts`
    - Verify validation error
- **Test Data:** `{  "title": "Test Post Title",  "userId": 1}`

- **Expected Results:**
    - Status code: 400 or 422
    - Error message indicating missing body
- **Postconditions:** None

TC-011: Create Post - Missing UserID

- **Requirement ID:** FR-002.2
- **Priority:** High
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
  - Prepare post data without userId
  - Send POST request to `/posts`
  - Verify validation error
- **Test Data:** `{ "title": "Test Post Title", "body": "This is test post content"}`

- **Expected Results:**
  - Status code: 400 or 422
  - Error message indicating missing userId
- **Postconditions:** None

TC-012: Create Post - Invalid UserID

- **Requirement ID:** FR-002.3
- **Priority:** Medium
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
  - Prepare post data with invalid userId
  - Send POST request to `/posts`
  - Verify handling of invalid data
- **Test Data:** `{ "title": "Test Post Title", "body": "This is test post content", "userId": "invalid"}`

- **Expected Results:**
  - Status code: 400 or accepts and converts to valid format
  - Appropriate data handling
- **Postconditions:** None

2.3 Posts Update Tests

TC-013: Update Post - Full Update (PUT)

- **Requirement ID:** FR-003.1
- **Priority:** High
- **Test Type:** Functional
- **Preconditions:** Post with ID 1 exists
- **Test Steps:**
  - Prepare complete post update data
  - Send PUT request to `/posts/1`
  - Verify update success
  - Validate updated data
- **Test Data:** `{ "id": 1, "title": "Updated Post Title", "body": "Updated post content", "userId": 1}`

- **Expected Results:**
  - Status code: 200
  - Response contains updated post data
  - All fields are updated as specified
- **Postconditions:** None

TC-014: Update Post - Valid Data Types

- **Requirement ID:** FR-003.1
- **Priority:** High
- **Test Type:** Validation
- **Preconditions:** Post with ID 1 exists
- **Test Steps:**
  - Send PUT request with proper data types
  - Verify type validation
- **Test Data:** `{ "id": 1, "title": "String Title", "body": "String Body", "userId": 10}`

- **Expected Results:**
  - Status code: 200
  - Data types preserved correctly

- **Postconditions:** None

TC-015: Update Post - Partial Update (PATCH)

- **Requirement ID:** FR-003.2
- **Priority:** Medium
- **Test Type:** Functional
- **Preconditions:** Post with ID 1 exists
- **Test Steps:**
  - Prepare partial update data
  - Send PATCH request to `/posts/1`
  - Verify partial update
- **Test Data:** `{ "title": "Partially Updated Title"}`

- **Expected Results:**
  - Status code: 200
  - Only specified fields updated
  - Other fields remain unchanged
- **Postconditions:** None

TC-016: Update Post - Non-existent ID

- **Requirement ID:** FR-003.3
- **Priority:** Medium
- **Test Type:** Error Handling
- **Preconditions:** API service is available
- **Test Steps:**
  - Prepare update data
  - Send PUT request to `/posts/999`
  - Verify error handling
- **Test Data:** `{ "title": "Updated Title", "body": "Updated content", "userId": 1}`

- **Expected Results:**
  - Status code: 404 or creates new resource
  - Appropriate error message or new resource creation
- **Postconditions:** None

2.4 Posts Deletion Tests

TC-017: Delete Post - Valid ID

- **Requirement ID:** FR-004.1
- **Priority:** High
- **Test Type:** Functional
- **Preconditions:** Post exists
- **Test Steps:**
  - Send DELETE request to `/posts/1`
  - Verify deletion response
- **Test Data:** Post ID = 1
- **Expected Results:**
  - Status code: 200 or 204
  - Successful deletion confirmation
- **Postconditions:** None

TC-018: Delete Post - Non-existent ID

- **Requirement ID:** FR-004.2
- **Priority:** Medium
- **Test Type:** Error Handling
- **Preconditions:** API service is available
- **Test Steps:**
  - Send DELETE request to `/posts/999`
  - Verify error handling
- **Test Data:** Post ID = 999
- **Expected Results:**
  - Status code: 404 or 200 (idempotent deletion)
  - Appropriate response message
- **Postconditions:** None

3. Users API Test Cases

3.1 Users Retrieval Tests

TC-019: Get All Users - Smoke Test

- **Requirement ID:** FR-005.1
- **Priority:** High
- **Test Type:** Smoke/Functional
- **Preconditions:** API service is available
- **Test Steps:**
  - Send GET request to `/users`
  - Verify response status code
  - Verify response contains users data
  - Verify response time
- **Test Data:** None required
- **Expected Results:**
  - Status code: 200
  - Response body contains array of users
  - Response time < 2 seconds
  - Each user has required fields
- **Postconditions:** None

TC-020: Get All Users - Data Structure Validation

- **Requirement ID:** FR-005.1
- **Priority:** High
- **Test Type:** Functional
- **Preconditions:** API service is available
- **Test Steps:**
  - Send GET request to `/users`
  - Validate user object structure
  - Check required fields presence
  - Validate nested objects (address, company)
- **Test Data:** None required
- **Expected Results:**
  - Users array contains 10 users

    - Each user has: id, name, username, email, address, phone, website, company
    - Address object has: street, suite, city, zipcode, geo
    - Company object has: name, catchPhrase, bs
    - Geo object has: lat, lng
- **Postconditions:** None

TC-021: Get Single User - Valid ID

- **Requirement ID:** FR-005.2
- **Priority:** High
- **Test Type:** Functional
- **Preconditions:** API service is available
- **Test Steps:**
    - Send GET request to `/users/1`
    - Verify response status code
    - Verify response contains single user
    - Validate user data structure
- **Test Data:** User ID = 1
- **Expected Results:**
    - Status code: 200
    - Response body contains single user object
    - User has complete data structure
    - User ID matches requested ID (1)
- **Postconditions:** None

TC-022: Get Single User - Boundary Values

- **Requirement ID:** FR-005.2
- **Priority:** Medium
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
    - Send GET request to `/users/1` (minimum valid ID)
    - Send GET request to `/users/10` (maximum valid ID)
    - Verify both responses
- **Test Data:** User IDs = 1, 10

- **Expected Results:**
  - Both requests return status code 200
  - Both responses contain valid user objects
  - IDs match requested values
- **Postconditions:** None

TC-023: Get Single User - Invalid ID Format

- **Requirement ID:** FR-005.2
- **Priority:** Medium
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
  - Send GET request to `/users/abc`
  - Send GET request to `/users/-1`
  - Send GET request to `/users/0`
- **Test Data:** User IDs = "abc", -1, 0
- **Expected Results:**
  - Status code: 404 or appropriate error status
  - Error response with meaningful message
- **Postconditions:** None

TC-024: Get Single User - Non-existent ID

- **Requirement ID:** FR-005.3
- **Priority:** Medium
- **Test Type:** Error Handling
- **Preconditions:** API service is available
- **Test Steps:**
  - Send GET request to `/users/999`
  - Verify error response
- **Test Data:** User ID = 999
- **Expected Results:**
  - Status code: 404
  - Empty response body or appropriate error message
- **Postconditions:** None

3.2 Users Creation Tests

TC-025: Create User - Valid Complete Data

- **Requirement ID:** FR-006.1
- **Priority:** High
- **Test Type:** Functional
- **Preconditions:** API service is available
- **Test Steps:**
  - Prepare complete valid user data
  - Send POST request to `/users`
  - Verify creation success
  - Validate created user data
- **Test Data:** `{ "name": "John Doe", "username": "johndoe", "email": "john@example.com", "address": { "street": "123 Main St", "suite": "Apt 1", "city": "Anytown", "zipcode": "12345", "geo": {"lat": "40.7128", "lng": "-74.0060"} }, "phone": "555-123-4567", "website": "johndoe.com", "company": { "name": "Acme Corp", "catchPhrase": "Quality first", "bs": "synergistic solutions" }}`

- **Expected Results:**
  - Status code: 201
  - Response contains created user with assigned ID
  - All input data preserved in response
- **Postconditions:** None

TC-026: Create User - Minimum Required Data

- **Requirement ID:** FR-006.1
- **Priority:** High
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
  - Prepare minimal user data (only required fields)
  - Send POST request to `/users`
  - Verify creation success

- **Test Data:** `{ "name": "Jane Doe", "username": "janedoe", "email": "jane@example.com"}`

- **Expected Results:**
  - Status code: 201
  - User created with minimal data
  - Optional fields handled appropriately
- **Postconditions:** None

TC-027: Create User - Missing Name

- **Requirement ID:** FR-006.2
- **Priority:** High
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
  - Prepare user data without name
  - Send POST request to `/users`
  - Verify validation error
- **Test Data:** `{ "username": "testuser", "email": "test@example.com"}`

- **Expected Results:**
  - Status code: 400 or 422
  - Error message indicating missing name
- **Postconditions:** None

TC-028: Create User - Missing Username

- **Requirement ID:** FR-006.2
- **Priority:** High
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
  - Prepare user data without username
  - Send POST request to `/users`
  - Verify validation error

- **Test Data:** `{ "name": "Test User", "email": "test@example.com"}`

- **Expected Results:**
  - Status code: 400 or 422
  - Error message indicating missing username
- **Postconditions:** None

TC-029: Create User - Missing Email

- **Requirement ID:** FR-006.2
- **Priority:** High
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
  - Prepare user data without email
  - Send POST request to `/users`
  - Verify validation error
- **Test Data:** `{ "name": "Test User", "username": "testuser"}`

- **Expected Results:**
  - Status code: 400 or 422
  - Error message indicating missing email
- **Postconditions:** None

TC-030: Create User - Invalid Email Format

- **Requirement ID:** FR-006.3
- **Priority:** Medium
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
  - Prepare user data with invalid email formats
  - Send POST requests to `/users`
  - Verify email validation
- **Test Data:** `[ {"name": "Test", "username": "test1", "email": "invalid-email"}, {"name": "Test", "username": "test2", "email": "@example.com"}, {"name": "Test", "username": "test3", "email":`

`"test@"}]`

- **Expected Results:**
  - Status code: 400 or 422 for invalid emails
  - Error messages indicating invalid email format
- **Postconditions:** None

4. Performance Test Cases

TC-037: Response Time - All Endpoints

- **Requirement ID:** NFR-001.1
- **Priority:** High
- **Test Type:** Performance
- **Preconditions:** API service is available
- **Test Steps:**
  - Send requests to all main endpoints
  - Measure response times
  - Calculate average response time
- **Test Data:** Various endpoint URLs
- **Expected Results:**
  - All endpoints respond within 2 seconds
  - Average response time < 1 second
- **Postconditions:** None

TC-038: Response Time - Under Load

- **Requirement ID:** NFR-001.1
- **Priority:** Medium
- **Test Type:** Performance
- **Preconditions:** API service is available
- **Test Steps:**
  - Send 10 concurrent requests to `/posts`
  - Measure response times
  - Verify performance degradation
- **Test Data:** Multiple concurrent requests

- **Expected Results:**
  - Response times remain under 2 seconds
  - No significant performance degradation
- **Postconditions:** None

5. Data Validation Test Cases

TC-040: JSON Schema Validation - Posts

- **Requirement ID:** NFR-002.1
- **Priority:** High
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
  - Get post data from `/posts/1`
  - Validate against JSON schema
  - Verify all required fields
- **Test Data:** Post schema definition
- **Expected Results:**
  - Response matches expected schema
  - All fields have correct data types
  - No missing required fields
- **Postconditions:** None

TC-041: JSON Schema Validation - Users

- **Requirement ID:** NFR-002.1
- **Priority:** High
- **Test Type:** Validation
- **Preconditions:** API service is available
- **Test Steps:**
  - Get user data from `/users/1`
  - Validate against JSON schema
  - Verify nested object structures
- **Test Data:** User schema definition
- **Expected Results:**

    - Response matches expected schema
    - Nested objects (address, company) have correct structure
    - All data types are consistent
- **Postconditions:** None

6. Test Execution Summary

6.1 Test Case Metrics

- **Total Test Cases:** 41
- **High Priority:** 24 (59%)
- **Medium Priority:** 17 (41%)
- **Low Priority:** 0 (0%)

6.2 Test Coverage by Feature

- **Posts API:** 18 test cases
- **Users API:** 18 test cases
- **Performance:** 2 test cases
- **Data Validation:** 3 test cases

6.3 Test Types Distribution

- **Functional:** 22 test cases (54%)
- **Validation:** 15 test cases (37%)
- **Performance:** 2 test cases (5%)
- **Error Handling:** 2 test cases (4%)