**API Testing Assignment**

## **1. Environment & Collection Setup**

1. **Create a Postman Environment**
   * **Hint:** Click the gear ⚙️ in the top right, “Manage Environments” → “Add”. Set a variable baseUrl = https://jsonplaceholder.typicode.com.
2. **New Collection “JSONPlaceholder Tests”**
   * **Hint:** In the sidebar, “New” → “Collection”. Name it and toggle “Persist Variables” on in Settings → General.
   * **Folder Structure:**
     + **Posts**
     + **Users**

### **2. Manual Request Creation & Tests**

For each endpoint, create a request and then switch to the **Tests** tab to add assertions.

| **Endpoint** | **Method** | **Tests to Write & Hints** |
| --- | --- | --- |
| GET {{baseUrl}}/posts | GET | - **Test:** Status 200- **Hint:** pm.test("OK", () => pm.response.to.have.status(200));- **Test:** Array length = 100- **Hint:**pm.expect(pm.response.json().length).to.eql(100); |
| GET {{baseUrl}}/posts/{{id}} | GET | - **Test (valid id):** 200- **Hint:** Set id in URL params.- **Test (invalid id):** 404- **Hint:** Under Params, set id to 9999. |
| POST {{baseUrl}}/posts | POST | - **Test:** Status 201- **Hint:** In Body → raw JSON: supply { "title":"foo", "body":"bar", "userId":1 }.- **Test:** Response has id field- **Hint:**pm.expect(pm.response.json()).to.have.property('id'); |
| PUT {{baseUrl}}/posts/{{id}} | PUT | - **Test:** Status 200- **Hint:** In Body, change "title" and assert via pm.expect(pm.response.json().title).to.eql("new"); |
| DELETE {{baseUrl}}/posts/{{id}} | DELETE | - **Test:** Status 200- **Hint:** After Delete, add a second request in the folder: GET same id → assert 404. |
| GET {{baseUrl}}/users | GET | - **Test:** Status 200- **Test:** Array length ≥ 10- **Hint:** Use pm.expect(pm.response.json().length).to.be.above(9); |
| GET {{baseUrl}}/users/{{id}} | GET | - **Test (valid):** 200- **Test (invalid):** 404- **Hint:** Duplicate the request, change idparam to 0. |

* **Schema Validation Hint:** In Tests tab:

  
const schema = { /\* paste JSON schema \*/ };

pm.test('Schema valid', () => pm.response.to.have.jsonSchema(schema));



### **3. Data-Driven Testing with Collection Runner**

1. **Prepare Data File:**

****id,expectedStatus

1,200

0,404

9999,404

1. **Hint:** In Runner, select your collection, choose the CSV, and tick “Persist Variables”.
2. **In Tests Tab:**

****const expected = parseInt(pm.iterationData.get('expectedStatus'), 10);

pm.test(`Status is ${expected}`, () => pm.response.to.have.status(expected));



### **4. Pre-Request & Environment Variable Usage**

* **Hint:** In the “Pre-request Script” tab of your collection or individual request:

  
pm.environment.set('ts', new Date().toISOString());

* Then reference {{ts}} in Headers or Query Params to simulate dynamic values.

### **5. Basic Performance Smoke via Monitor**

1. **Hint:** In the collection view, click “Monitors” → “Create monitor”.
2. Schedule it (e.g., hourly) and enable “Response time” alerts.
3. **Hint:** Review metrics in Postman’s Monitor dashboard to spot regressions.

### **6. Security Smoke Test**

* **Request:** GET {{baseUrl}}/posts?userId=' OR 1=1 --
* **Tests Tab Hint:**

****pm.test('No stack trace exposed', () => {

pm.expect(pm.response.text()).not.include('Exception');

});

pm.test('Status is 400 or 200', () => {

pm.expect([200,400]).to.include(pm.response.code);

});



### **7. Deliverables**

* **Exported Collection & Environment** (via “…” → Export)
* **Collection Documentation** (in Postman: “View Documentation” → Publish or Export)
* **Runner/Monitor Report** (screenshots or HTML report)
* **Bug Reports** (use Postman Console to capture request/response snippets)

# **API Testing Assignment - Deliverables**

**Objective:**

* Test REST APIs for functionality, schema validity, performance, and security.
* Use manual, automated (data-driven), and scheduled monitoring approaches.

**Scope:**

* API Endpoints tested: **Posts** and **Users** (CRUD operations)
* Tools used: **Postman** (Collection Runner, Monitor, Environment Variables)

**Test Activities Performed:**

* **Environment Setup:** Configured baseUrl and dynamic timestamp (ts) variable.
* **Manual Testing:** Created requests for GET, POST, PUT, DELETE operations.
* **Assertions:** Validated Status Codes, Response Data, and Schema structure.
* **Schema Validation:** Ensured API responses match expected JSON schema.
* **Data-Driven Testing:** Used CSV to run multiple ID scenarios through Collection Runner.
* **Security Testing:** Simulated SQL Injection to validate API resilience.
* **Performance Monitoring:** Set up Postman Monitor to track API uptime and response times.

**Postman Collection**

* <https://zoya-3668638.postman.co/workspace/zoya's-Workspace~75af44f9-5353-4682-8aba-0b528ec10e17/collection/44329802-16c16300-0278-48ed-8a3c-252dfc5b1704?action=share&creator=44329802&active-environment=44329802-4fcf0278-ba80-4f0c-9ce5-a9a8e2c5a79c>

**Steps Performed:**

1. Environment Setup

* Created a Postman environment named JSONPlaceholder Env.
* Added the environment variable:
  + baseUrl = <https://jsonplaceholder.typicode.com>

2. Collection Setup

* Created a Postman Collection named JSONPlaceholder Assignment.
* Organized requests into two folders:  
  + Posts
  + Users
* Enabled Persist Variables in the collection settings.

**3. Manual API Requests & Tests**

**3.1 Posts Folder**

* GET /posts  
  + Verified Status 200.
  + Verified array length is exactly 100.
  + Performed Schema Validation to validate the structure of each post object.
* GET /posts/{id}  
  + For valid id=1, verified Status 200.
  + For invalid id=9999, verified Status 404.
* POST /posts  
  + Created a new post with title, body, and userId.
  + Verified Status 201 and that the response contains an id field.
* PUT /posts/{id}  
  + Updated an existing post’s title.
  + Verified Status 200 and title updated correctly.
* DELETE /posts/{id}  
  + Deleted an existing post.
  + Verified Status 200 on delete.
  + Confirmed that after deletion, a GET on the same id returns Status 404.

**3.2 Users Folder**

* GET /users  
  + Verified Status 200.
  + Verified the array length is greater than or equal to 10.
* GET /users/{id}  
  + For valid id=1, verified Status 200.
  + For invalid id=0, verified Status 404.

**4. Schema Validation**

* Applied JSON Schema validation for the GET /posts response.
* Ensured each post object contains:  
  + userId (integer)
  + id (integer)
  + title (string)
  + body (string)

**5. Data-Driven Testing (Collection Runner)**

* Prepared a CSV file DataDriven\_Posts.csv containing test data:  
  + id,expectedStatus
  + 1,200
  + 0,404
  + 9999,404
* Created a dynamic request GET Post by ID - Data Driven that:  
  + Reads id and expectedStatus from the CSV during collection run.
  + Asserts the actual status matches the expected status dynamically.

**6. Security Testing**

* Performed SQL injection simulation by sending:
  + GET /posts?userId=' OR 1=1 --
* Verified:  
  + No server-side stack traces or errors exposed.
  + Status code is either 400 (Bad Request) or 200 (OK).

**Test Results:**

| **Test Type** | **Status** |
| --- | --- |
| Status Code Verification | ✅ Passed |
| Response Data Validation | ✅ Passed |
| Schema Validation | ✅ Passed |
| Data-Driven Testing | ✅ Passed |
| Pre-Request Script Execution | ✅ Passed |
| Security Testing (No Exception Exposure) | ✅ Passed |