API Session – Chapter 6

- Response Validations -

# Response Validation

## Status Code

* Indicates the overall success or failure of the request.
* Essential for determining if the API behaved as expected.
* Examples: 200 OK, 404 Not Found, 500 Internal Server Error.

## Message

* Provides additional context about the response status.
* Can offer insights into the reason for success or failure.
* Helps in debugging and understanding the API's behavior.

## Header

* Contains metadata about the response, such as content type, encoding, and cookies.
* Essential for verifying data format and handling session information.
* Can provide additional information about the response, like caching directives.

## Cookie

* Stores session information and user preferences.
* Important for validating session management and authentication.
* Can be used to track user interactions and personalize responses.

## Response Body

* Contains the actual data returned by the API.
* The core focus of most validation efforts.
* Validating data structure, content, and specific values is crucial.

## Response Time

* While not directly part of the response content, it's essential for performance testing and monitoring.

# Postman Sandbox: Your Scripting Playground

**The Postman Sandbox is essentially a JavaScript environment built into Postman.**

It's where you write scripts to automate tasks, manipulate data, and perform complex operations within your API testing workflow.

## Key Functions of the Postman Sandbox

* **Pre-request Scripts:** Code executed before a request is sent, allowing you to modify request parameters, set headers, or generate dynamic data.
* **Test Scripts:** Code executed after a response is received, enabling you to validate response data, extract values, and write assertions.
* **Accessing Request and Response Data:** You can interact with request and response objects using the pm object, allowing you to inspect and manipulate data.
* **Environment and Global Variables:** Manage and access environment and global variables for dynamic data management.
* **Custom Functions:** Create reusable functions to streamline your scripts.

**In essence, the Postman Sandbox empowers you to extend the capabilities of Postman beyond simple request-response interactions.** By writing scripts, you can automate repetitive tasks, perform complex data transformations, and create more robust and efficient API tests.

# Scripting Types

* Normal Function

**Syntax:**

**pm.test("Test Name", function () {**

**// Assertion statement;**

**});**

* Arrow Function

**Syntax:**

**pm.test("Test Name", () => {**

**// Assertion Statement;**

**});**

# Chai Library for Assertion

The Postman Sandbox uses the Chai library for assertion in scripting.

# Tests

In postman, under the Tests section we can write the validation functions.

## Test for the response status code

**pm.test("Status code is 200", function () {**

**pm.response.to.have.status(200);**

**});**

**pm.test("Status code is 200", () => {**

**pm.response.to.have.status(200);**

**});**

## Test for the response status code to check if it’s in allowed list

**pm.test("Status code is one of 200, 201, or 404", () => {**

**pm.expect(pm.response.code).to.be.oneOf([200, 201]);**

**});**

## Test for the response status code text

**pm.test("Status code text is Created", function () {**

**pm.response.to.have.status(“Created”);**

**});**

## Test for the presence of header in response

**pm.test("Content-Type header is Present", () => {**

**pm.response.to.have.header("Content-Type");**

**});**

## Test for the response header having specific value

**pm.test("Content-Type header is correct", () => {**

**const contentType = pm.response.headers.get("Content-Type");**

**pm.expect(contentType).to.eql("application/json; chartset=utf-8");**

**});**

## Test for the presence of cookie in response

**pm.test("Cookie is present", () => {**

**const cookieName = "your\_cookie\_name";**

**pm.expect(pm.response.cookies.has(cookieName)).to.be.true;**

**});**

## Test for the response cookie having specific value

**pm.test("Cookie language has value en-gb", () => {**

**const languageCookie = pm.response.cookies.get("language");**

**pm.expect(languageCookie.value).to.eql("en-gb");**

**});**

## Test for the response time to be in range

**pm.test("Response time is less than 200ms", () => {**

**pm.expect(pm.response.responseTime).to.be.below(200);**

**});**

## Test for the response body data type check

**pm.test("Validate data types", () => {**

**const responseJson = pm.response.json();**

**pm.expect(responseJson.id).to.be.a('number');**

**pm.expect(responseJson.name).to.be.a('string');**

**pm.expect(responseJson.rollNumber).to.be.a('number');**

**pm.expect(responseJson.email).to.be.a('string');**

**});**

## Test for the response body data type check with Array data

**{**

**"id": 1,**

**"name": "John Doe",**

**"rollNumber": 1234,**

**"subjects": [ "Java", "Python"]**

**}**

**pm.test("Validate response data", () => {**

**const jsonData = pm.response.json();**

**pm.expect(jsonData.id).to.be.a('number');**

**pm.expect(jsonData.name).to.be.a('string');**

**pm.expect(jsonData.rollNumber).to.be.a('number');**

**pm.expect(jsonData.subjects).to.be.an('array');**

**pm.expect(jsonData.subjects[0]).to.be.a('string');**

**pm.expect(jsonData.subjects[1]).to.be.a('string');**

**});**

## Test for the response body fields to exists

**pm.test("Validate response fields", () => {**

**const responseJson = pm.response.json();**

**pm.expect(responseJson.id).to.exist;**

**pm.expect(responseJson.name).to.exist;**

**pm.expect(responseJson.rollNumber).to.exist;**

**pm.expect(responseJson.email).to.exist;**

**});**

## Test for the response body fields

**pm.test("Validate response fields", () => {**

**const responseJson = pm.response.json();**

**pm.expect(responseJson.id).to.eql(1);**

**pm.expect(responseJson.name).to.eql("John Doe");**

**pm.expect(responseJson.rollNumber).to.eql(1234);**

**pm.expect(responseJson.email).to.eql("johndoe@example.com");**

**});**

## Test for the response body fields in advance format

**pm.test("Validate response fields", () => {**

**const expectedFields = ["id", "name", "rollNumber", "email"];**

**const responseJson = pm.response.json();**

**// Check if all expected fields exist**

**expectedFields.forEach(field => {**

**pm.expect(responseJson.hasOwnProperty(field)).to.be.true;**

**});**

**// Validate specific values (optional)**

**pm.expect(responseJson.id).to.equal(1); // Assuming, Id should have value as 1**

**pm.expect(responseJson.name).to.equal("John Doe"); // Assuming name should be "John Doe"**

**// Add similar assertions for other fields if needed**

**});**

# JSON Schema in API Testing

**JSON Schema** is a vocabulary that allows you to annotate and validate JSON documents. In the context of API testing, it defines the expected structure, data types, and constraints of an API response.

## Resource

<https://www.liquid-technologies.com/online-json-to-schema-converter>

## JSON Schema

**{**

**"$schema": "http://json-schema.org/draft-04/schema#",**

**"type": "object",**

**"properties": {**

**"students": {**

**"type": "array",**

**"items": [**

**{**

**"type": "object",**

**"properties": {**

**"id": {**

**"type": "integer"**

**},**

**"name": {**

**"type": "string"**

**},**

**"rollNumber": {**

**"type": "integer"**

**},**

**"email": {**

**"type": "string"**

**}**

**},**

**"required": [**

**"id",**

**"name",**

**"rollNumber",**

**"email"**

**]**

**},**

**{**

**"type": "object",**

**"properties": {**

**"id": {**

**"type": "integer"**

**},**

**"name": {**

**"type": "string"**

**},**

**"rollNumber": {**

**"type": "integer"**

**},**

**"email": {**

**"type": "string"**

**}**

**},**

**"required": [**

**"id",**

**"name",**

**"rollNumber",**

**"email"**

**]**

**}**

**]**

**}**

**},**

**"required": [**

**"students"**

**]**

**}**

## Test for JSON Schema validation

**var schema = {**

**"$schema": "http://json-schema.org/draft-04/schema#",**

**"type": "object",**

**"properties": {**

**"students": {**

**"type": "array",**

**"items": [**

**{**

**"type": "object",**

**"properties": {**

**"id": {**

**"type": "integer"**

**},**

**"name": {**

**"type": "string"**

**},**

**"rollNumber": {**

**"type": "integer"**

**},**

**"email": {**

**"type": "string"**

**}**

**},**

**"required": [**

**"id",**

**"name",**

**"rollNumber",**

**"email"**

**]**

**},**

**{**

**"type": "object",**

**"properties": {**

**"id": {**

**"type": "integer"**

**},**

**"name": {**

**"type": "string"**

**},**

**"rollNumber": {**

**"type": "integer"**

**},**

**"email": {**

**"type": "string"**

**}**

**},**

**"required": [**

**"id",**

**"name",**

**"rollNumber",**

**"email"**

**]**

**}**

**]**

**}**

**},**

**"required": [**

**"students"**

**]**

**}**

**pm.test("Schema is valid", function () {**

**pm.expect(tv4.validate(jsonData, schema)).to.be.true;**

**});**