Postman Response Validations

Chai Assertion Library

Postman uses the Chai assertion library for writing tests. There are two syntaxes for defining tests:

Using Function Syntax:

```
pm.test("Test Name", function() {
  // assertion;
});
```

Using Arrow Function Syntax:

```
pm.test("Test Name", () => {
  // assertion;
});
```

1. Validating Status Code

- **Check for a specific status code:**

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
```

- **Check for one of multiple status codes:**

```
pm.test("Successful POST request", () => {
  pm.expect(pm.response.code).to.be.oneOf([201, 202]);
});
```

- **Check status code text:**

```
pm.test("Status code name has string", () => {
  pm.response.to.have.status("Created");
});
```

2. Validating Headers

- **Check if a header is present:**

```
pm.test("Content-Type header is present", () => {
  pm.response.to.have.header("Content-Type");
});
```

- **Check header value:**

```
pm.test("Content-Type header is application/json", () => {
  pm.expect(pm.response.headers.get('Content-Type')).to.eql('application/json;
charset=utf-8');
});
```

```
pm.test("Content-Type header contains application/json", () => {
  pm.expect(pm.response.headers.get('Content-
Type')).to.include('application/json');
});
```

3. Validating Cookies

- **Check if a cookie is present:**

```
pm.test("Cookie 'language' is present", () => {
  pm.expect(pm.cookies.has('language')).to.be.true;
});
```

- **Check cookie value:**

```
pm.test("Cookie language has value 'en-gb'", () => {
  pm.expect(pm.cookies.get('language')).to.eql('en-gb');
});
```

4. Validating Response Time

- **Check response time is below a threshold:**

```
pm.test("Response time is less than 200ms", () => {
  pm.expect(pm.response.responseTime).to.be.below(200);
});
```

5. Validating Response Body

Sample Response:

```
{
    "id": 1,
    "name": "John",
    "location": "india",
    "phone": "1234567890",
    "courses": [
        "Java",
        "Selenium"
    ]
}
```

- **Check value types:**

```
const jsonData = pm.response.json();
pm.test("Test data type of the response", () => {
  pm.expect(jsonData).to.be.an("object");
  pm.expect(jsonData.name).to.be.a("string");
  pm.expect(jsonData.id).to.be.a("number");
  pm.expect(jsonData.courses).to.be.an("array");
});
```

- **Check array properties:**

```
pm.test("Test array properties", () => {
  pm.expect(jsonData.courses).to.include("Java");
  pm.expect(jsonData.courses).to.have.members(["Java", "Selenium"]);
});
```

- **Check specific JSON Fields data**

```
pm.test("Validate specific fields in JSON response", () => {
    var jsonData = pm.response.json();
    pm.expect(jsonData.id).to.eql(1);
    pm.expect(jsonData.name).to.eql("John");
    pm.expect(jsonData.location).to.eql("india");
    pm.expect(jsonData.phone).to.eql("1234567890");
    pm.expect(jsonData.courses[0]).to.eql("Java");
    pm.expect(jsonData.courses[1]).to.eql("Selenium");
});
```

6. Validating JSON Schema

Example Response

```
{
    "id": 1,
    "name": "John",
    "location": "india",
    "phone": "1234567890",
    "courses": [
        "Java",
        "Selenium"
    ]

}
```

JSON Schema

```
var schema = {
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "id": {
      "type": "integer"
    },
    "name": {
      "type": "string"
    },
    "location": {
      "type": "string"
    },
    "phone": {
      "type": "string"
    },
    "courses": {
      "type": "array",
      "items": [
        { "type": "string" },
        { "type": "string" }
      ]
    }
  },
  "required": ["id", "name", "location", "phone", "courses"]
};
```

Schema Validation

```
pm.test('Schema is valid', function() {
  pm.expect(tv4.validate(jsonData, schema)).to.be.true;
});
```