**API**

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

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

**});**

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

**pm.expect(pm.response.code).to.eql(200);**

**});**

**pm.test("The response has all properties", () => {**

**//parse the response json and test three properties**

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

**pm.expect(responseJson.type).to.eql('vip');**

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

**pm.expect(responseJson.id).to.have.lengthOf(1);**

**});**

**pm.test("Body contains string",() => {**

**pm.expect(pm.response.text()).to.include("customer\_id");**

**});**

**pm.test("Body is string", function () {**

**pm.response.to.have.body("whole-body-text");**

**});**

**pm.test("Person is Jane", () => {**

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

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

**pm.expect(responseJson.age).to.eql(23);**

**});**

**pm.test("Status code is 201", () => {**

**pm.response.to.have.status(201);**

**});**

**pm.test("Successful POST request", () => {**

**pm.expect(pm.response.code).to.be.oneOf([201,202]);**

**});**

**pm.test("Status code name has string", () => {**

**pm.response.to.have.status("Created");**

**});**

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

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

**});**

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

**pm.expect(pm.response.headers.get('Content-Type')).to.eql('application/json');**

**});**

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

**pm.expect(pm.cookies.has('JSESSIONID')).to.be.true;**

**});**

**pm.test("Cookie isLoggedIn has value 1", () => {**

**pm.expect(pm.cookies.get('isLoggedIn')).to.eql('1');**

**});**

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

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

**});**

**pm.test("Response property matches environment variable", function () {**

**pm.expect(pm.response.json().name).to.eql(pm.environment.get("name"));**

**});**

**/\* response has this structure:**

**{**

**"name": "Jane",**

**"age": 29,**

**"hobbies": [**

**"skating",**

**"painting"**

**],**

**"email": null**

**}**

**\*/**

**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.age).to.be.a("number");**

**pm.expect(jsonData.hobbies).to.be.an("array");**

**pm.expect(jsonData.website).to.be.undefined;**

**pm.expect(jsonData.email).to.be.null;**

**});**

**/\***

**response has this structure:**

**{**

**"errors": [],**

**"areas": [ "goods", "services" ],**

**"settings": [**

**{**

**"type": "notification",**

**"detail": [ "email", "sms" ]**

**},**

**{**

**"type": "visual",**

**"detail": [ "light", "large" ]**

**}**

**]**

**}**

**\*/**

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

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

**//errors array is empty**

**pm.expect(jsonData.errors).to.be.empty;**

**//areas includes "goods"**

**pm.expect(jsonData.areas).to.include("goods");**

**//get the notification settings object**

**const notificationSettings = jsonData.settings.find**

**(m => m.type === "notification");**

**pm.expect(notificationSettings)**

**.to.be.an("object", "Could not find the setting");**

**//detail array should include "sms"**

**pm.expect(notificationSettings.detail).to.include("sms");**

**//detail array should include all listed**

**pm.expect(notificationSettings.detail)**

**.to.have.members(["email", "sms"]);**

**});**

**pm.expect({a: 1, b: 2}).to.have.all.keys('a', 'b');**

**pm.expect({a: 1, b: 2}).to.have.any.keys('a', 'b');**

**pm.expect({a: 1, b: 2}).to.not.have.any.keys('c', 'd');**

**pm.expect({a: 1}).to.have.property('a');**

**pm.expect({a: 1, b: 2}).to.be.an('object')**

**.that.has.all.keys('a', 'b');**

**pm.test("Value is in valid list", () => {**

**pm.expect(pm.response.json().type)**

**.to.be.oneOf(["Subscriber", "Customer", "User"]);**

**});**

**/\***

**response has the following structure:**

**{**

**"id": "d8893057-3e91-4cdd-a36f-a0af460b6373",**

**"created": true,**

**"errors": []**

**}**

**\*/**

**pm.test("Object is contained", () => {**

**const expectedObject = {**

**"created": true,**

**"errors": []**

**};**

**pm.expect(pm.response.json()).to.deep.include(expectedObject);**

**});**

**pm.test("Check the active environment", () => {**

**pm.expect(pm.environment.name).to.eql("Production");**

**});**

**console.log(pm.collectionVariables.get("name"));**

**console.log(pm.response.json().name);**

**console.log(typeof pm.response.json().id);**

**if (pm.response.json().id) {**

**console.log("id was found!");**

**// do something**

**} else {**

**console.log("no id ...");**

**//do something else**

**}**

**pm.test("Test 1", () => {**

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

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

**});**

**pm.test("Test 2", () => {**

**pm.expect(jsonData.age).to.eql(29); // jsonData is not defined**

**});**

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

**//test function not properly defined - missing second parameter**

**pm.test("Not failing", function () {**

**pm.expect(true).to.eql(false);**

**});**

**You can carry out JSON schema validation with tv4.**

**const schema = {**

**"items": {**

**"type": "boolean"**

**}**

**};**

**const data1 = [true, false];**

**const data2 = [true, 123];**

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

**pm.expect(tv4.validate(data1, schema)).to.be.true;**

**pm.expect(tv4.validate(data2, schema)).to.be.true;**

**});**

**You can also validate JSON schema with ajv by default.**

**const schema = {**

**"properties": {**

**"alpha": {**

**"type": "boolean"**

**}**

**}**

**};**

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

**pm.response.to.have.jsonSchema(schema);**

**});**

**Sending an asynchronous request**

**You can send a request from your tests code and log the response.**

**pm.sendRequest("https://postman-echo.com/get", function (err, response) {**

**console.log(response.json());**

**});**

**tests["Body contains user\_id"] = responsebody.has("user\_id");**

**//set an environment variable**

**postman.setEnvironmentVariable("key", "value");**

**//set a nested object as an environment variable**

**const array = [1, 2, 3, 4];**

**postman.setEnvironmentVariable("array", JSON.stringify(array, null, 2));**

**const obj = { a: [1, 2, 3, 4], b: { c: 'val' } };**

**postman.setEnvironmentVariable("obj", JSON.stringify(obj));**

**//get an environment variable**

**postman.getEnvironmentVariable("key");**

**//get an environment variable whose value is a stringified object**

**//(wrap in a try-catch block if the data is coming from an unknown source)**

**const array = JSON.parse(postman.getEnvironmentVariable("array"));**

**const obj = JSON.parse(postman.getEnvironmentVariable("obj"));**

**//clear an environment variable**

**postman.clearEnvironmentVariable("key");**

**//set a global variable**

**postman.setGlobalVariable("key", "value");**

**//get a global variable**

**postman.getGlobalVariable("key");**

**//clear a global variable**

**postman.clearGlobalVariable("key");**

**//check if response body contains a string**

**tests["Body matches string"] = responseBody.has("string\_you\_want\_to\_search");**

**//check if response body is equal to a string**

**tests["Body is correct"] = responseBody === "response\_body\_string";**

**//check for a JSON value**

**const data = JSON.parse(responseBody);**

**tests["Your test name"] = data.value === 100;**

**//Content-Type is present (Case-insensitive checking)**

**tests["Content-Type is present"] = postman.getResponseHeader("Content-Type");**

**tests["Content-Type is present"] = postman.getResponseHeader("Content-Type");**

**//getResponseHeader() method returns the header value, if it exists**

**//Content-Type is present (Case-sensitive)**

**tests["Content-Type is present"] = responseHeaders.hasOwnProperty("Content-Type");**

**//response time is less than 200ms**

**tests["Response time is less than 200ms"] = responseTime < 200;**

**//response time is within a specific range**

**//(lower bound inclusive, upper bound exclusive)**

**tests["Response time is acceptable"] = \_.inRange(responseTime, 100, 1001);**

**//status code is 200**

**tests["Status code is 200"] = responseCode.code === 200;**

**//code name contains a string**

**tests["Status code name has string"] = responseCode.name.has("Created");**

**//successful POST request status code**

**tests["Successful POST request"] = responseCode.code === 201 || responseCode.code === 202;**