# JSON Schema Validation in Postman

In the previous articles on [Postman Tutorial](https://www.softwaretestingmaterial.com/postman-tutorial/), we have covered “[How to send JWT Token as header](https://www.softwaretestingmaterial.com/how-to-send-jwt-token-as-header/)“

In this “JSON Schema Validation in Postman” article, I will be demonstrating as to how you can implement this concept and get a tight grip over this.

### **What is JSON?**

JSON stands for JavaScript Object Notation. JSON is a lightweight format for storing and transporting data. JSON is often used when data is sent from a server to a web page. JSON is not a programming language. It is a common and open standard format for storing and exchanging data. JSON is language independent.

### **Rules – JSON:**

1. Data is in key/value pairs: Key/value pair consists of a key in double quotes followed by a colon, followed by a value. In JSON, keys must be strings in double quotes but values can be of any type such as a string, a number, an object (JSON object), an array, a Boolean, null.

**Example:**

"name": "John"

2. Data is separated by commas: Data is separated by a comma in JSON

**Example:**

});

"name": "John",  
"age": 10,  
"country": "USA"

3. Object starts and ends with curly braces {} : JSON Objects are surrounded by Curly braces {}

**Example:**

{

"name": "John",

"age": 10,

"country": "USA"

}

4. Arrays starts and ends with square brackets [] : Square brackets hold arrays.

**Example:**

{

"Students": [

{

"name": "John",

"age": 10,

"country": "USA"

},

{

"name": "Mark",

"age": 12,

"country": "USA"

}

]

}

### **What is JSON Path?**

It is the specific value or data in a JSON. We can create a JSON Path manually or by using tools like JSON PathFinder – http://jsonpathfinder.com/

We just need to copy the JSON and paste it in the tool.

**Example:**

{

"Students": [

{

"name": "John",

"age": 10,

"country": "USA"

},

{

"name": "Mark",

"age": 12,

"country": "USA"

}

]

}

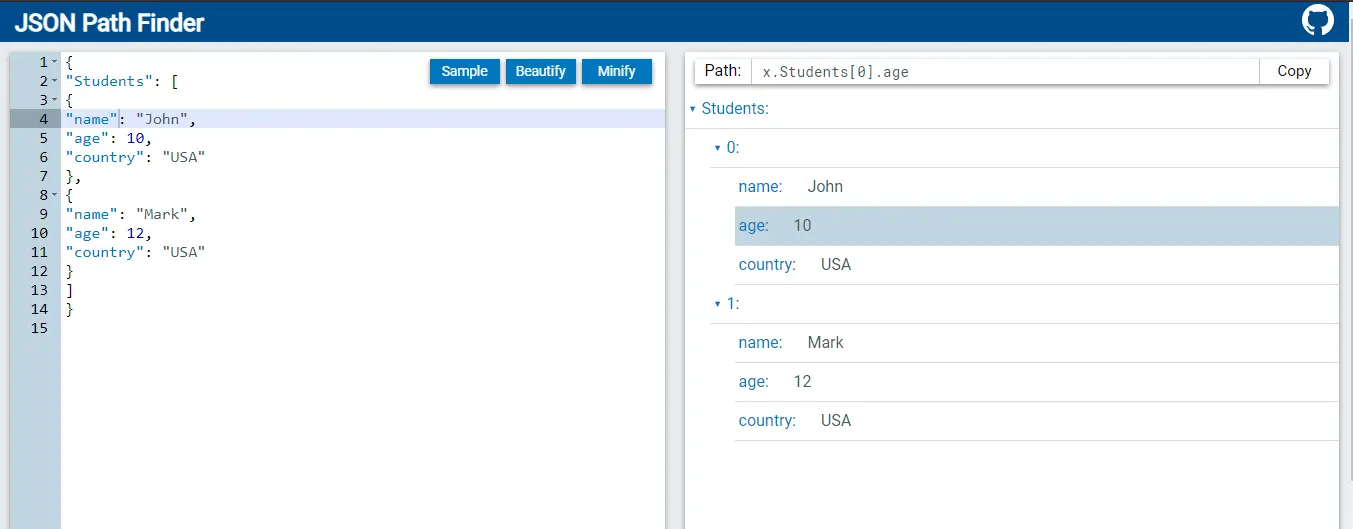
Let the complete Students array is stored in a variable say x.

* JSON path of age of John is: x.Students[0].age
* JSON path of name of Mark is: x.Students[1].name

**What is JSON Schema?**

JSON Schema is a contract for JSON document that defines the expected data types and format of each field in the response.

It describes the existing data format with clear, human and machine-readable documentation for complete structural validation, useful for automated testing and validating client submitted data.



**Why is JSON Schema Validation required?**

JSON Schema Validation is required because:

1. We monitor API responses and ensure that the format that we are getting is same as the expected one.
2. We get alert whenever there is any breaking change in JSON response.
3. We use JSON Schema to construct a model of API response and it makes easier to validate that API is returning the valid data.

**JSON Object:**

[

{

"name": "John",

"age": 10

}

]

**JSON Schema Example:**

{

"type": "array",

"items":

{

"type": "object",

"Properties":

{

"name": {"type": "string"},

"age": {"type": "integer"}

}

}

}

**JSON Schema Validation**

Schema itself is an object which has different properties. One of these properties is **type**. If we declare the **type** as an object, it means if we send something that is not an object that will not work. We will be using Another JSON Schema Validator (Ajv). It’s recently built-in in postman.

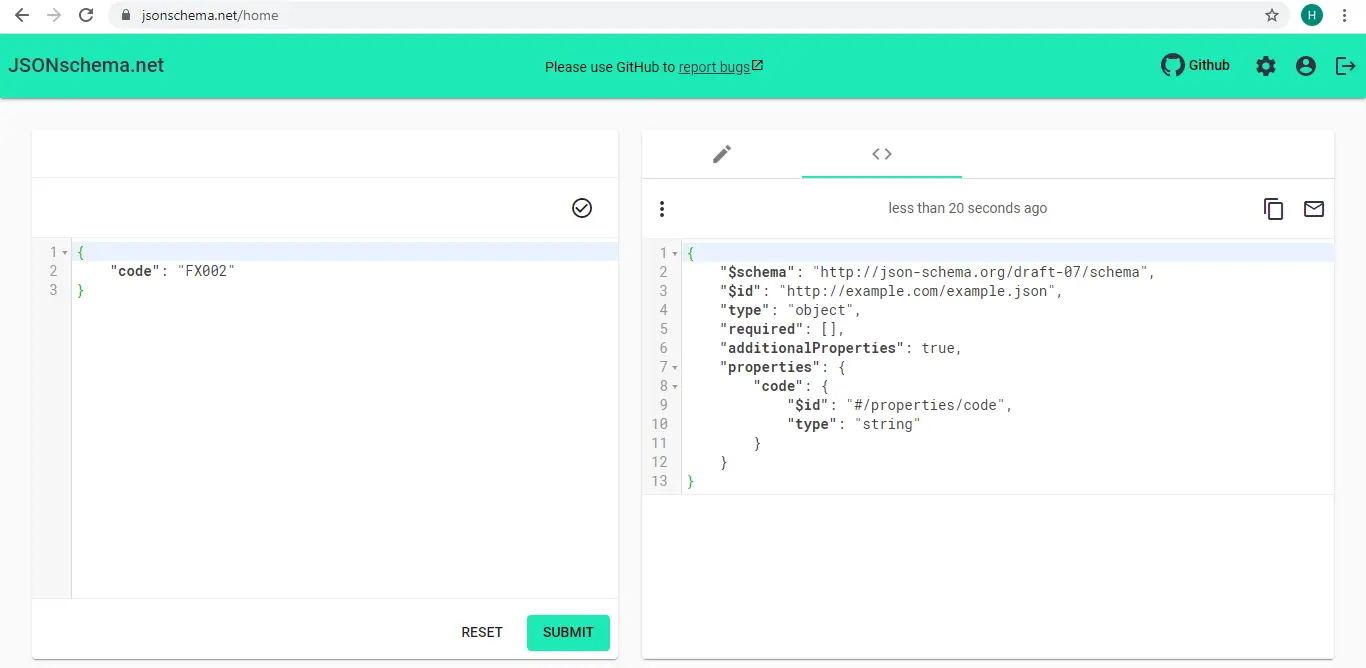
* Go to https://jsonschema.net/home
* Add JSON body on left side
* Click Submit
* It will create a schema on right side
* Copy the schema and add in Tests tab in Postman to perform schema validation
* Add this code for schema validation:

pm.test("Validate schema", () => {

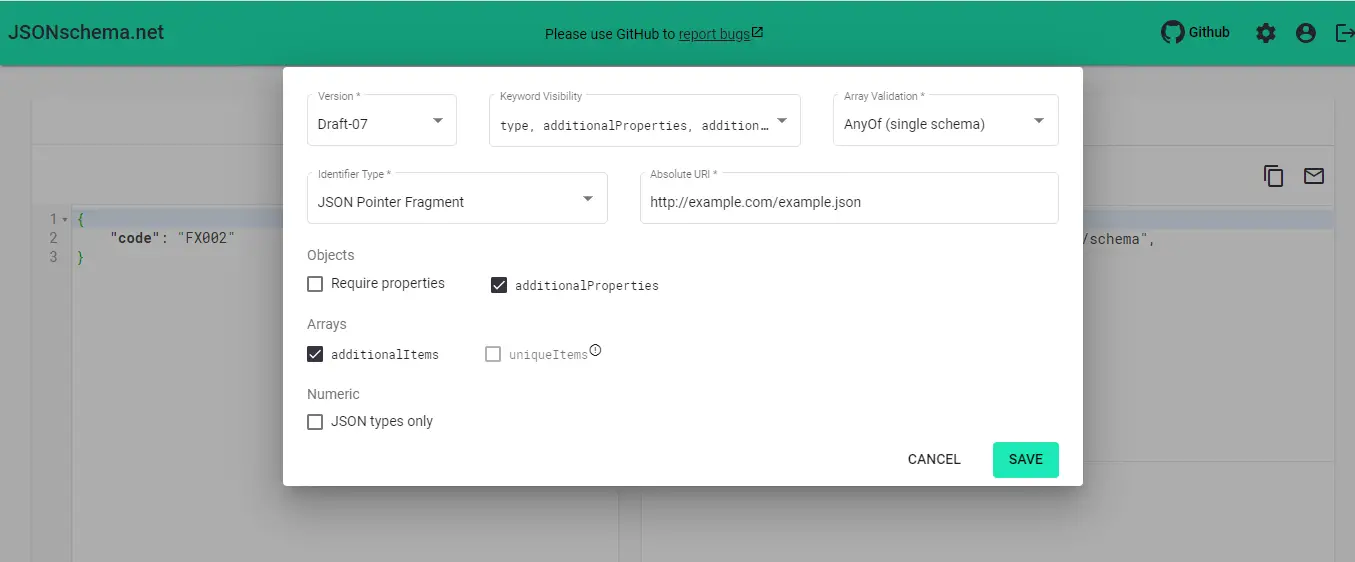
pm.response.to.have.jsonSchema(schema);

});

* We have created a JSON schema out of our response and now we validate JSON response with our JSON schema and if anything related to the properties of this schema and JSON’s response don’t match, the test will fail.



We can edit the settings as per our need from gear icon on top right corner. We can change certain things. For example, in Object I want to have additionalProperties and don’t want every property to be required. I can even choose which keyword should be included in JSON Schema from keyword visibility option.



Let’s look into more details of JSON Schema Validation:

1. For an empty object {} in response, the Validate schema test passed.

**Under Tests tab,**

const schema = {

"type": "object"

};

pm.test("Validate schema", () => {

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

});

2. To verify whether the schema is actually valid, let’s replace the body with Foo. We will get a JSON error. So schema will fail if the body is not JSON.

3. Instead of empty body Let’s say we have a property having a string in the response body:

{

"code": "FX002"

}

The schema will be valid because it is an object.

If we have to test for this property we will need to add another property to the schema and the property is called properties and in the properties object, we can define multiple properties.

const schema = {

"type": "object",

"properties": {

"code": { "type" : "string" }

}

};

pm.test("Validate schema", () => {

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

});

When we send this, Schema will be valid.

4. If we have an integer value,

{

"code": 2

}

Schema validation will fail. The error will say code should be a string.

5. Let’s check what will happen if this will be renamed to statusCode.

{

"statusCode": 2

}

When we run this, the schema will still be valid, even though we have specified a property called code that should be a string. This is the reason, we need to test our schema in detail to make sure everything works.

In this case, we can specify that this property is actually required.

const schema = {

"type": "object",

"properties": {

"code": { "type" : "string" }

},

"required": ["code"]

};

pm.test("Validate schema", () => {

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

});

When we send the request again, we will see that validation is now correct. It checks for a property code and in the response, there is no property code. Therefore, the Validate Schema test failed. The error says “data should have required property ‘code’ “.

6. To limit the number of properties that we get from the response, we specify **additionalProperties** in our schema.

**Example:**

{

"code": "FX002",

"error": {}

}

If we want to avoid having the properties that we don’t know about and we want to be informed if something really changes in the response, then we can add an additional configuration to the schema. This is called **additionalProperties**. If we specify it false, then it will not allow additional properties.

const schema = {

"type": "object",

"properties": {

"code": { "type" : "string" }

},

"required": ["code"],

"additionalProperties": false

};

pm.test("Validate schema", () => {

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

});

Validate Schema will fail. It says data should not have additional properties. But the response has error as an additional property.

7. Let’s suppose we have a nested object in the response called error and error itself has a property called message.

**Example:**

{

"code": "FX002",

"error": {

"message": "Something went wrong"

}

}

**The schema will be :**

const schema = {

"type": "object",

"properties": {

"code": { "type" : "string" },

"error": {

"type": "object",

"properties": {

"message": { "type": "string"}

},

"required": ["message"]

}

},

"required": ["code", "error"],

"additionalProperties": false

};

pm.test("Validate schema", () => {

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

});

Validate Schema test will pass.

8. If we have arrays in our response,

**Example:**

{

"code": "FX002",

"error": {

"message": "Something went wrong"

},

"tags": ["fatal, "error"]

}

**The schema will be :**

const schema = {

"type": "object",

"properties": {

"code": { "type" : "string" },

"error": {

"type": "object",

"properties": {

"message": { "type": "string"}

},

"required": ["message"]

},

"tags" : {

"type": "array",

"minItems": 1,

"maxItems": 3,

"items": {

"type": "string"

}

}

},

"required": ["code", "error"],

"additionalProperties": false

};

pm.test("Validate schema", () => {

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

});

Validate Schema test will pass. If we want to have atleast 1 item, we will specify as **minItems** property and similarly, we can specify **maxItems**. To define the types of items in an array, we use items property.