

API testing with Bruno

German perl workshop 2025

<https://usebruno.com>

What is Bruno?

- What postman used to be or should be
- Rest api client gui.
- Text based storage (use git and diff)
- Command line tool for automated testing
- Open source: <https://github.com/usebruno/bruno/>
- But some paid features

Install Bruno desktop app

- From <https://www.usebruno.com/>
- Runs local on windows, linux, macOs
- Built using Next.js and React.
Uses electron to ship a desktop version
- Tipp (for all electron apps): Ctrl-Shift-I for dev-tools (console)
- Demo data:
 - <https://dummyjson.com>
 - <https://restful-api.dev>
 - <https://reqbin.com>

Hello world

- Create collection
- Create request

- Run request

NEW REQUEST ×

Type
 HTTP GraphQL From cURL

Name
reqbin echo

URL
GET https://reqbin.com/echo

Cancel Create

→

GET <https://dummyjson.com/products/:ID?TOKEN={{TOKEN}}>

Params ² Body Headers Auth Vars Script ^{*} Assert ¹ Tests

Docs

Query

Name	Path	
TOKEN	<code>{{TOKEN}}</code>	<input checked="" type="checkbox"/> 
l p	<code>{{PRICE}}</code>	<input type="checkbox"/> 

[+ Add Param](#)

Path [?](#)

Name	Value
ID	3

GET
POST
PUT
DELETE
PATCH
OPTIONS
HEAD

Environments

The screenshot shows a user interface for managing environments, likely in a development tool or API management system. At the top, there are several icons: a clock, a person, an eye, a gear, and a globe. To the right of these is a dropdown menu set to "dev". Below this is a header bar with the title "Collection Environments" and a "ENVIRONMENTS" section.

The "ENVIRONMENTS" section contains a list of environments:

- dev** (highlighted in orange)
- dev.sample
- prod
- No Environment
- Configure

Below the environment list is a button labeled "+ Create".

To the right of the environment list is a detailed view of the "dev" environment. It shows a table of variables:

Enabled	Name	Value	Secret	
<input checked="" type="checkbox"/>	ID	1	<input type="checkbox"/>	trash
<input checked="" type="checkbox"/>	TOKEN	asdfasdf	<input type="checkbox"/>	trash

At the bottom of this view is a link "+ Add Variable".

```
1 {  
2   "id": 3,  
3   "title": "Powder Canister",  
4   "description": "The Powder Canister is a finely milled setting po  
wder designed to set makeup and control shine. With a lightweight a  
nd translucent formula, it provides a smooth and matte finish.",  
5   "category": "beauty",  
6   "price": 14.99,  
7   "discountPercentage": 9.84,  
8   "rating": 4.64,  
9   "stock": 89,  
10  "tags": [  
11    "beauty",  
12    "face powder"  
13  ],
```

[Response](#)[Headers 22](#)[Timeline](#)[Tests 1](#)[Clear Timeline](#)**200 OK GET** [2025-05-03T10:01:13.145Z]

2 minutes ago

<https://dummyjson.com/products/3?TOKEN=asdfasdf>[Request](#)[Response](#)[Network Logs](#)<https://dummyjson.com/products/3?TOKEN=asdfasdf>[▼ Headers](#)

No Headers found

[▼ Body](#)

No Body found

200 OK GET [2025-05-01T13:43:56.649Z]

2 days ago

<https://dummyjson.com/products/4?TOKEN=asdfasdf>

Create, change and delete in testing

POST ▼ <https://dummyjson.com/products/add> </> 🗑️ →

Params Body * Headers Auth * Vars

Script Assert Tests Docs

JSON ▾ Prettify

```
1 {  
2   "title": "Powder Canister",  
3   "description": "The Powder.",  
4   "category": "beauty"  
5 }
```

Response Headers 22 Timeline Tests
🔗 ⏪ 201 Created 368ms 84B

```
1 {  
2   "id": 195,  
3   "title": "Powder Canister",  
4   "description": "The Powder.",  
5   "category": "beauty"  
6 }
```

Store response data in variable

Search requests ...

- > test
- > validate qs
- > präsi
- > powerfox
- ▽ präsi
 - GET product1
 - POST crud_product_1
 - GET crud_product_2
 - PATCH crud_product_3
 - DEL crud_product_4

POST ▶ <https://dummyjson.com/products/add>

Params Body * Headers Auth * Vars Script * Assert Tests

Docs

Pre Request

```
1
```

Post Response

```
1 // bru.setVar("DELID", res.body.id);
2 bru.setVar("DELID", 1);
```

Reuse response data in variable

The screenshot shows the Postman application interface. On the left, there's a sidebar with a 'Collections' section containing several items: 'test', 'validate qs', 'präsi', 'powerfox', and a expanded 'präsi' section which includes 'product1', 'crud_product_1', 'crud_product_2', 'crud_product_3', and 'crud_product_4'. The 'crud_product_4' item is currently selected, highlighted with a grey background. The main workspace shows a 'DELETE crud_p...' request. The URL is `https://dummyjson.com/products/:ID?TOKEN={{TOKEN}}`. The 'Params' tab is selected, showing a table with one row: Name 'TOKEN' and Path value '{{TOKEN}}'. There are a checked checkbox and a trash bin icon next to the row. Below this is a '+ Add Param' button. The 'Path' tab is also visible, showing a table with one row: Name 'ID' and Value '{{DELID}}'.

Collections

DELETE crud_p... × +

Search requests ...

> test

> validate qs

> präsi

> powerfox

präsi

- GET product1
- POST crud_product_1
- GET crud_product_2
- PATCH crud_product_3
- DEL crud_product_4

DELETE https://dummyjson.com/products/:ID?TOKEN={{TOKEN}}

Params 2

Name	Path
TOKEN	{{TOKEN}}

+ Add Param

Path ?

Name	Value
ID	{{DELID}}

Check result

- Simple assertions

Params ²	Body	Headers	Auth	Vars ¹	Script [*]	Assert ³
Tests Docs						
Expr	Operator	Value				
res.body.id	equals	{{TESTID}}	<input checked="" type="checkbox"/>			
res.body.title	isString		<input checked="" type="checkbox"/>			
res.body.title	contains	Powder Canister	<input checked="" type="checkbox"/>			

[+ Add Assertion](#)

- result

Response Headers ²² Timeline Tests ³ 200 OK 303ms 1.43KB

Tests (0/0), Passed: 0, Failed: 0

Assertions (3/3), Passed: 3, Failed: 0

- ✓ res.body.id: eq {{TESTID}}
- ✓ res.body.title: isString
- ✓ res.body.title: contains Powder Canister

Check result with Javascript (Tests)

Params Body Headers Auth * Vars Script * Assert

Tests * Docs

```
1 test("title should contain 'Mascara'", () => {  
2     expect(res.body.title).to.contain('Mascara');  
3 })
```

Response Headers 22 Timeline Tests 1 ⌂ ⏪ 200 OK 307ms 1.47KB

Tests (1/1), Passed: 1, Failed: 0

✓ title should contain 'Mascara'

Assertions (0/0), Passed: 0, Failed: 0

Testing in Javascript - Buildins

- atob - Turn base64-encoded ascii data back to binary.
- btoa - Turn binary data to base64-encoded ascii.
- chai - BDD/TDD assertion library for node.js and the browser
<https://www.chaijs.com/>.
- moment - Parse, validate, manipulate, and display dates and times in JavaScript.
- uuid - For the creation of RFC4122 UUIDs.
- nanoid - A tiny, secure, URL-friendly, unique string ID generator for JavaScript.
- crypto-js - JavaScript library of crypto standards.

Debugging tests with console.log()

```
1 test("brand should contain 'Mascara'", () => {  
2   console.log(res.body);  
3   expect(res.body.brand).to.contain('Mascara');  
4 })
```

Tests (1/1), Passed: 0, Failed: 1

X brand should contain 'Mascara'
expected 'Essence' to include 'Mascara'

Assertions (0/0), Passed: 0, Failed: 0

The screenshot shows the Chrome Developer Tools Console tab. At the top, there are tabs for Elements, Console, Sources, and Network. Below the tabs is a toolbar with icons for search, refresh, and other developer functions. The main area displays the output of a console.log statement. The output is a JSON object representing a product. The object has properties like id, title, description, availabilityStatus, brand, category, dimensions, and discountPercentage. The brand value is shown as "Essence". The description field contains a long string of text about the product. The object is shown with expandable sections indicated by arrows.

```
index.82df5080.js:681  
{  
  id: 1, title: 'Essence Mascara Lash Princess', description: 'The E  
ssence Mascara Lash Princess is a popular mas... with this long-las  
ting and cruelty-free formula.', category: 'beauty', price: 9.99, ...}  
  i  
    availabilityStatus: "In Stock"  
    brand: "Essence"  
    category: "beauty"  
    description: "The Essence Mascara Lash Princess is a popular mas  
► dimensions: {width: 15.14, height: 13.08, depth: 22.99}  
    discountPercentage: 10.48
```

Bruno CLI

- npm install -g @usebruno/cli
- bru run request.bru
- bru run --env=dev request.bru

- npx @usebruno/cli run --env=dev
- npx @usebruno/cli run --env=dev --output results.html --format html

```
C:\Users\Public\Documents\bruno\praesi>set NODE_OPTIONS="--no-deprecation"
```

```
C:\Users\Public\Documents\bruno\praesi>npx @usebruno/cli run --env=dev  
Running Folder Recursively
```

```
product1 (200 OK) - 401 ms
  ✓ assert: res.body.id: eq {{TESTID}}
  ✓ assert: res.body.title: isString
  ✓ assert: res.body.title: contains Powder Canister
crud_product_1 (201 Created) - 102 ms
crud_product_2 (200 OK) - 104 ms
  ✓ brand should contain 'Essence'
crud_product_3 (200 OK) - 101 ms
  ✓ assert: res.body.id: eq {{ID}}
crud_product_4 (200 OK) - 104 ms
  ✓ assert: res.body.id: eq {{ID}}
```

```
Requests: 5 passed, 5 total
```

```
Tests: 1 passed, 1 total
```

```
Assertions: 5 passed, 5 total
```

```
Ran all requests - 812 ms
```

```
Requests: 5 passed, 5 total
```

```
Tests: 1 passed, 1 total
```

```
Assertions: 5 passed, 5 total
```



Summary

Requests

Iteration 1

Total requests

5

Total errors

0

Total Controls

4

Total Failed Controls

0

Total run duration

0.912 s

SUMMARY ITEM	TOTAL	PASSED	FAILED	SKIPPED	ERROR
Requests	5	5	0	0	0
Assertions	4	4	0	-	-



Summary

Requests

Iteration 1

 Show All product1 - 2/2 Passed

REQUEST INFORMATION

File

product1

Request Method

GET

RESPONSE INFORMATION

Response Code

200

Response time

375 ms

Real live (run in gitlab)

- Gitlab Pipeline schedules
- .gitlab-ci.yml
 - stages: - brunoTest
 - bruno_test:
 - script:
 - pushd tests/Bruno/
 - npm install
 - npx @usebruno/cli run --env=dev
 - popd

tests/Bruno/package.json

 package.json  131 B

```
1  {
2      "name": "bruno-validate-api-collection",
3      "version": "1.0.0",
4      "dependencies": {
5          "jsonschema": "^1.4.1"
6      }
7  }
```

- Version 1.5.0 is broken
- Install with „npm i“

Variables in

- Requests
- Collection
- Bruno Environment
- .env-file
- Environment vars
- Use: {{VAR}} or {{process.env.ENVVAR}}

Reusable code in js-module

js helper.js 2.37 KiB

```
1 const getElem = (res, name, value) => {
2     return res.body.results[0][name].find((e)=>e.validatorName === value);
3 };
4
5 const testSchema = (res) => {
6     var Validator = require('jsonschema').Validator;
7     var v = new Validator();
8     var schema = {
9         "id": "/ValidateResultV1",
10        "type": "object",
11        "properties": {
12            "requestUUID": {
13                "type": "string",
14                "pattern": /^[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}\$/,
15            },
16            "status": {
17                "type": "string",
18                "enum": [
19                    "PENDING",
20                    "VALIDATED"
21                ]
22            }
23        }
24    };
25    v.validate(res.body, schema);
26    if (v.error) {
27        console.log(v.error);
28        res.status(400).send(v.error);
29    } else {
30        res.status(200).send("Success");
31    }
32}
```

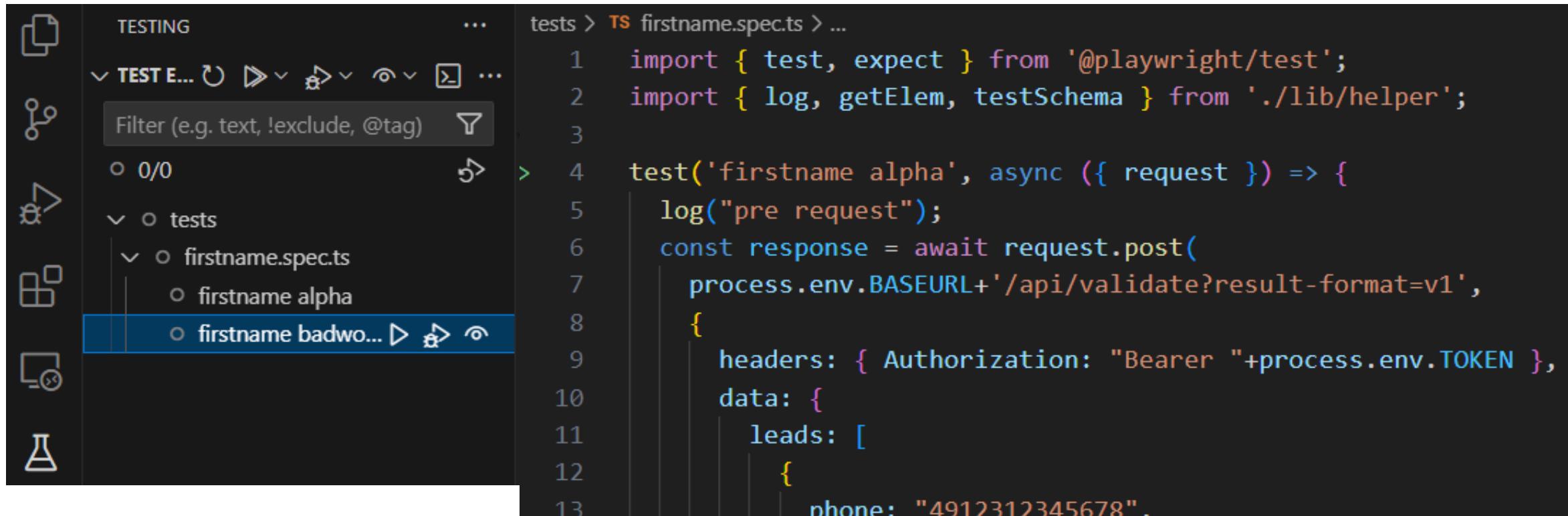
Test with helper module

```
60  const h = require("./lib/helper");
61
62  // schema
63  test("body should be valid schema. check console on failure", () => {
64      expect(h.testSchema(res).errors).to.be.empty;
65  });
66
67  // firstName
68  test("firstName should have validator badword with result false", () => {
69      expect(h.getElem(res, "firstName", "badword").result).to.be.false;
70  });
```

Debugging

- With console.log
- With node debugger, but tricky.
- If you need a debugger, maybe Bruno is not the right tool
- If you program at lot, it is useful. But that's not what it's for

Can I do the same in Playwright?



The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with various icons and a tree view of project files. The tree view shows a folder named 'TEST E...' containing a file 'firstname.spec.ts'. Below this, under 'tests', are two files: 'firstname.spec.ts' and 'firstname alpha'. The 'firstname.spec.ts' file is currently selected and highlighted with a blue bar at the bottom.

The main editor area displays the contents of the 'firstname.spec.ts' file:

```
tests > TS firstname.spec.ts > ...
1 import { test, expect } from '@playwright/test';
2 import { log, getElem, testSchema } from './lib/helper';
3
4 test('firstname alpha', async ({ request }) => {
5   log("pre request");
6   const response = await request.post(
7     process.env.BASEURL+'/api/validate?result-format=v1',
8   {
9     headers: { Authorization: "Bearer "+process.env.TOKEN },
10    data: {
11      leads: [
12        {
13          phone: "4912312345678",

```

Compare Bruno and Playwright

Bruno

- + easy install Gui and runtime
- + simple API testing
- + GUI to develop tests

Playwright

- + debuggable (e.g. vscode plugin)
 - + complex API testing (prepare/cleanup)
 - + integration with other testing
-
- big installation