Uncodemy · Docs v2.0

Turn any API spec into battle-tested test suites.

A single-page, browser-only tool that reads Swagger / OpenAPI and instantly generates ready-to-run tests in Jest, Pytest, or Postman. No backend and no build step.

Get started → <> View the code

03	06	100%
Export formats	Test categories	Browser-only

01 GETTING STARTED

Complete Beginner's Guide

This document explains everything — what the project does, how it's built, how each part of the code works, and how you can customize it. No backend, no database, no build tools.

A single-page, browser-only tool that reads a Swagger / OpenAPI specification and instantly generates ready-to-run API test code. Open the HTML file in a browser and you're ready to go.

100% Browser-Only · Privacy-First · JSON / YAML Supported · One HTML File

> 💡 **Who is this for?**
>
> Beginners learning API testing, QA engineers who want a quick starting point, and developers who want to turn a spec into test scaffolding without writing boilerplate.

---

02 SETUP

Prerequisites & How to Run

You do not need Node.js, npm, Python, or any server to run this project. It is a single HTML file.

What you need

- A modern browser (Chrome, Edge, Firefox, Safari).
- The project file — for example `index.html` — saved on your computer.
- An internet connection only for the CDN-loaded YAML parser.

How to run it

1. Save the HTML file anywhere on your computer.
2. Double-click it. It opens in your default browser.
3. Paste an OpenAPI spec or click **Load sample**.
4. Press **Generate tests**. Done! 🎉

> ✅ **No install needed.**
>
> This is a pure front-end project. It runs entirely inside the browser tab. Your API spec never leaves your machine.

Folder structure

The whole project is one self-contained HTML file. Internally it is organized like this:

```
api-test-generator/
└── index.html
    ├── <head>
    │   ├── meta + title
    │   ├── Google Fonts (Space Grotesk, JetBrains Mono)
    │   ├── js-yaml CDN (for YAML parsing)
    │   └── <style>  → all design tokens + CSS
    ├── <body>
    │   ├── Header (logo + theme toggle)
    │   ├── Step ribbon (Paste → Generate → Export)
    │   ├── Spec input card + side panel
    │   ├── Results grid (test list + export panel)
    │   └── Footer
    └── <script>
        ├── SAMPLE spec object
        ├── State (spec, tests, filter, fmt)
        ├── Spec helpers (parse, res, val)
        ├── Test generation (generate)
        ├── Code export (code)
        ├── Rendering (render)
        └── Event listeners (click, input, change)
```

---

03 HIGHLIGHTS

What Makes This Project Unique

It's not just a code generator — it's a small, complete, offline tool with a polished UI. Here's what it packs:

Two Input Modes

Paste JSON/YAML directly or upload a file from disk.

Smart Test Generation

Positive, missing-required, wrong-type, boundary, unauthorized, and not-found tests — all generated automatically.

Three Export Formats

Jest, Pytest, and Postman collection — ready to download.

Filterable Test List

Chip-based filters let you focus on positive or negative cases instantly.

Auto + Manual Dark Mode

Follows your OS by default, with a toggle to override.

Fully Offline Logic

All parsing, generation, and export happens in the browser.

Tech stack

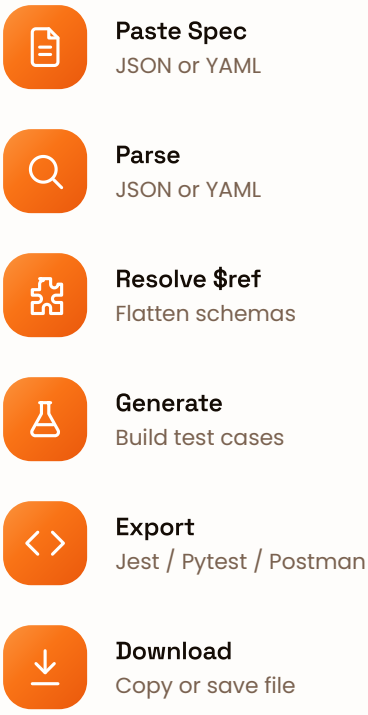
LAYER	TECHNOLOGY	WHY
Markup	HTML5	Semantic, accessible structure.
Styling	CSS (custom properties)	Theme tokens for light/dark; no framework needed.
Logic	Vanilla JavaScript (ES6+)	Zero framework, zero build step.
YAML parsing	`js-yaml` (CDN)	Converts YAML specs into JS objects.

LAYER	TECHNOLOGY	WHY
Fonts	Google Fonts	Space Grotesk + JetBrains Mono for a modern look.

04 UNDER THE HOOD

How It Works — End-to-End Flow

From the moment you paste a spec to the moment you download test code, the data passes through six clear stages.



Internal architecture

Inside the browser, the app behaves like a small pipeline. The diagram below shows which part of the code handles which stage.

INPUT

Textarea / File

User provides JSON or YAML spec.

INPUT

Base URL + Token

Side-panel fields for auth and host.

↓

CORE

parse()

Tries JSON, falls back to js-yaml.

↓

CORE

res() + val()

Resolve $refs, generate example values.

↓

CORE

generate()

Builds the tests[] array.

↓

OUTPUT

render()

Draws stats, chips, test rows.

OUTPUT

code()

Converts tests[] to source code.

↓

FINAL

Copy / Download

Clipboard or Blob download.

Key idea:
Everything is a pure function of the `spec` object and the current settings. No hidden state, no server calls, no surprises.

---

05 WALKTHROUGH

Step-by-Step Usage

Follow these steps to go from an empty page to a downloaded test file in under a minute.

01 **Open the app**
Double-click the HTML file. The page loads in your browser.

02 **Load a spec**
Click *Load sample* for a demo, or paste your own OpenAPI/Swagger JSON or YAML into the editor, or click *Upload file*.

03 **Set the Base URL**
Enter the host where your API runs (e.g. `http://localhost:3000`). The auth token field is used for secured endpoints.

04 **Press “Generate tests”**
The app parses the spec and fills the test list with positive and negative test cases.

05 **Filter the tests**
Use the chips (All, Positive, Missing required, Wrong type, Boundary, Unauthorized, Not found) to inspect specific categories.

06 **Pick an export format**

Choose Jest, Pytest, or Postman on the export panel.

07

Copy or Download

Hit *Copy* to put the code on your clipboard, or *Download* to save a properly named file.

⚠️ **Spec must contain a `paths` object.**
If the parser can't find it, the app shows a friendly error. Make sure you're pasting a real Swagger/OpenAPI document.

---

06 <> CODE WALKTHROUGH

Understanding the JavaScript, Block by Block

Below are the most important parts of the code, with annotations so you can learn how each piece works.

6.1 — State variables

The app keeps all its state in four small variables. That's the entire "database" of the app.

state.js

```
// The parsed OpenAPI object
let spec = null;

// Every generated test case lives here
let tests = [];

// Active filter chip (default: show everything)
let filter = "All";

// Active export format
let fmt = "Jest";

// Available filter categories
const TYPES = [
  "All", "Positive", "Missing required",
  "Wrong type", "Boundary",
  "Unauthorized", "Not found",
];
```

6.2 — Parsing the spec (JSON or YAML)

The app supports both JSON and YAML. It first tries `JSON.parse()`. If that fails, it falls back to `js-yaml`.

parse.js

```
function parse(t) {
  try {
    // Try JSON first — fastest path
    return JSON.parse(t);
  } catch (e) {
    // Not JSON — assume YAML
    return jsyaml.load(t);
  }
}
```

6.3 — Resolving `$ref` references

OpenAPI files use `$ref` to reuse schemas. For example:

example-spec.json

```
{
  "schema": { "$ref": "#/components/schemas/Pet" }
}
```

The `res()` function follows that pointer and returns the full schema object. The `d` parameter prevents infinite loops (max depth 6).

resolver.js

```
function res(s, d = 0) {
  // Safety: stop deep / invalid recursion
  if (!s || typeof s !== "object" || d > 6) return s || {};

  if (s.$ref) {
    // Split "#/components/schemas/Pet" into ["components","schemas","Pet"]
    const p = s.$ref.replace(/^#\//, "").split("/");
    let n = spec;
    for (const k of p) n = n && n[k];
    return res(n || {}, d + 1);
  }

  // Recursively resolve nested object properties
  const o = { ...s };
  if (o.properties) {
    o.properties = {};
    for (const k in s.properties)
      o.properties[k] = res(s.properties[k], d + 1);
  }
  if (o.items) o.items = res(o.items, d + 1);
  return o;
}
```

6.4 — Generating sample values

`val()` creates a realistic example value for any schema. It respects `type`, `format`, `enum`, `minimum`, `maximum`, `minLength`, and `maxLength`.

values.js

```
function val(s, d = 0) {
  s = res(s);                    // resolve $ref first
  if (d > 5) return null;
  if (s.example !== undefined) return s.example;
  if (s.enum) return s.enum[0];

  switch (s.type) {
    case "integer": return s.minimum ?? 1;
    case "number":  return s.minimum ?? 1.5;
    case "boolean": return true;
    case "array":   return [val(s.items || {}, d + 1)];
    case "object":
    case undefined: {
      if (!s.properties) return s.type ? {} : "sample";
      const o = {};
      for (const k in s.properties)
        o[k] = val(s.properties[k], d + 1);
      return o;
    }
    default:
      // string formatting shortcuts
      if (s.format === "email") return "user@example.com";
      if (s.format === "uuid")  return "123e4567-e89b-12d3-a456-426614174000";
      let v = "sample";
      if (s.minLength && v.length < s.minLength)
        v = v.padEnd(s.minLength, "x");
      return v;
  }
}
```

6.5 — Generating the test cases

`generate()` is the heart of the app. It loops over every path and every HTTP method, then creates multiple test cases per endpoint.

generator.js

```
function generate() {
  tests = [];
  let eps = 0, secured = 0, n = 0;

  for (const path in spec.paths || {}) {
    const item = spec.paths[path];

    for (const method of ["get", "post", "put", "patch", "delete"]) {
      const op = item[method];
      if (!op) continue;
      eps++;

      // Gather parameters (path-level + operation-level)
      const ps = [
        ...(item.parameters || []),
        ...(op.parameters || []),
      ].map((p) => res(p));

      // Extract the request body schema
      const rb = op.requestBody &&
                 op.requestBody.content &&
                 (op.requestBody.content["application/json"] ||
                  Object.values(op.requestBody.content)[0]);
      const bs = res(rb ? rb.schema : null);

      // Does this endpoint require auth?
      const sec = (op.security ?? spec.security ?? []).length > 0;
      if (sec) secured++;

      // Find the success status (2xx)
      const ok = Object.keys(op.responses || {})
        .find((c) => /^2/.test(c)) || "200";

      // Build path + query values for the "valid" request
      const pathV = {}, query = {};
      ps.forEach((p) => {
        const s = p.schema || p;
        if (p.in === "path") pathV[p.name] = val(s);
        if (p.in === "query" && p.required) query[p.name] = val(s);
      });

      // ...then push positive, missing-required, wrong-type,
      // boundary, auth and not-found test cases here.
    }
```

```
  }
}
```

> 🎯 **Why so many tests?**
> Every endpoint gets at least one positive test. The generator then adds negative tests for the most common failure cases — missing fields, wrong types, out-of-range values, missing auth, and non-existent IDs.

6.6 — Exporting to Jest / Pytest / Postman

The `code()` function converts the internal `tests[]` array into human-readable test code. Jest output looks like this:

jest-output.js

```
// npm i -D jest supertest
const request = require("supertest");
const BASE = "http://localhost:3000";

describe("GET /pets", () => {
  test("Positive: List pets - valid request", async () => {
    const res = await request(BASE)
      .get("/pets")
      .set({ "Content-Type": "application/json" })
      .query({ limit: 1 });

    expect(res.status).toBe(200);
  });
});
```

The same `tests[]` array is transformed into a Python `requests` script or a Postman Collection v2.1 JSON.

6.7 — Rendering the UI

`render()` redraws the chips, stats, and test rows. It's called whenever the filter or format changes.

render.js

```
function render() {
  // 1. Filter chips
  const chipsEl = $("chips");
  chipsEl.innerHTML = "";
  TYPES.forEach((t) => {
    // ...build chip button + click handler
  });

  // 2. Test list
  const f = tests.filter((t) => filter === "All" || t.type === filter);
  // ...render each row with method badge, path, tag, description

  // 3. Code panel
  $("out").textContent = code();
}
```

6.8 — Theme toggle logic

The theme follows the OS by default. Once the user clicks the toggle, a `data-theme` attribute overrides the OS setting.

theme.js

```
$("theme").addEventListener("click", () => {
  const root = document.documentElement;
  const prefersDark =
    window.matchMedia &&
    window.matchMedia("(prefers-color-scheme: dark)").matches;

  // If a manual theme is set, use that. Otherwise use OS.
  const currentDark = root.dataset.theme
    ? root.dataset.theme === "dark"
    : prefersDark;

  root.dataset.theme = currentDark ? "light" : "dark";
});
```

07 MAKE IT YOURS

Customization Guide

All customization is done by editing CSS variables or small pieces of JavaScript. No rebuild required.

Change the accent color

Find the `:root` block inside `<style>` and change `--accent`:

style.css

```
:root {
  --accent: #0d9488;       /* change me */
  --accent-strong: #0f766e;
  --accent-soft: #ccfbf1;
}
```

Change the default Base URL

Find `<input id="base" value="http://localhost:3000">` and change the `value` attribute.

Add a new export format

1. Add the name to the `FMTS` array.
2. Add a new branch inside `code()`.
3. Add a filename to the `FILE_MAP` object at the top of the script.

Change the sample spec

The demo spec lives in the `SAMPLE` constant. Replace it with your own object to change what *Load sample* produces.

Persist the theme choice

Wrap the theme toggle logic in `localStorage` so the preference survives a page reload:

persist-theme.js

```
// On load
const saved = localStorage.getItem("theme");
if (saved) document.documentElement.dataset.theme = saved;

// Inside the click handler, after setting data-theme:
localStorage.setItem("theme", root.dataset.theme);
```

08 REFERENCE

Beginner Glossary

New to APIs? These are the terms used throughout the project.

TERM	MEANING
API	Application Programming Interface — a way for two programs to talk to each other.
Swagger / OpenAPI	A standard way to describe a REST API. Usually written as JSON or YAML.
Endpoint	A URL + HTTP method combination, e.g. `GET /pets`.
Path parameter	Part of the URL, like `/pets/{id}`.
Query parameter	Data after `?`, e.g. `/pets?limit=10`.
Request body	Data sent with POST/PUT requests, usually JSON.
Schema	A description of what data looks like (types, ranges).
$ref	A pointer to a schema defined elsewhere in the file.
Bearer token	A string sent in the `Authorization` header for protected endpoints.

TERM	MEANING
Jest	A popular JavaScript testing framework.
Pytest	The most common Python testing framework.
Postman	A GUI tool for testing APIs; collections are JSON files.

09 HELP

FAQ & Troubleshooting

The most common questions and how to fix them.

+ I clicked “Generate tests” but nothing happened.

+ YAML is not parsing correctly.

+ My spec uses OAuth2 instead of Bearer tokens.

+ Can I run the generated tests directly?

+ Is my API spec safe? Does anything get uploaded?

+ The generated tests expect 400 for missing fields — my API returns 422.

+ How do I add more test cases?

Still stuck?
Open the browser DevTools (F12) → *Console* tab. Any parsing error will be printed there with a line number.

---

10 GO FURTHER

Ideas to Extend the Project

Once you understand the code, here are some natural next steps.

- **Add GraphQL support** — parse a GraphQL schema and generate query tests.
- **Add schema validation tests** — compare the response body against the declared schema.
- **Persist your last spec** — save it to `localStorage` so it's still there after a reload.
- **Add a "Run tests" button** — actually execute the generated requests via `fetch()` and show results.
- **Export as cURL** — a one-liner per endpoint that developers can paste into a terminal.
- **Add a dark-mode toggle in the export panel** — so you can preview the generated code in both themes.