

OpenAPI for Web Developers

Lorna Mitchell

About OpenAPI

Standard for describing APIs, for machines and for humans.

OpenAPI Example

```
openapi: 3.0.1
servers:
  - url: http://datasette.local
  - url: https://datasette.io
info:
  description: Execute SQL queries against a Datasette database
    and return the results as JSON
  title: Datasette API
  version: v1
paths:
  /content.json:
    get:
      description: Accepts SQLite SQL query, returns JSON. Does
        not allow PRAGMA statements.
```

Credit: <https://github.com/APIs-guru/openapi-directory>

Code-First vs Design-First

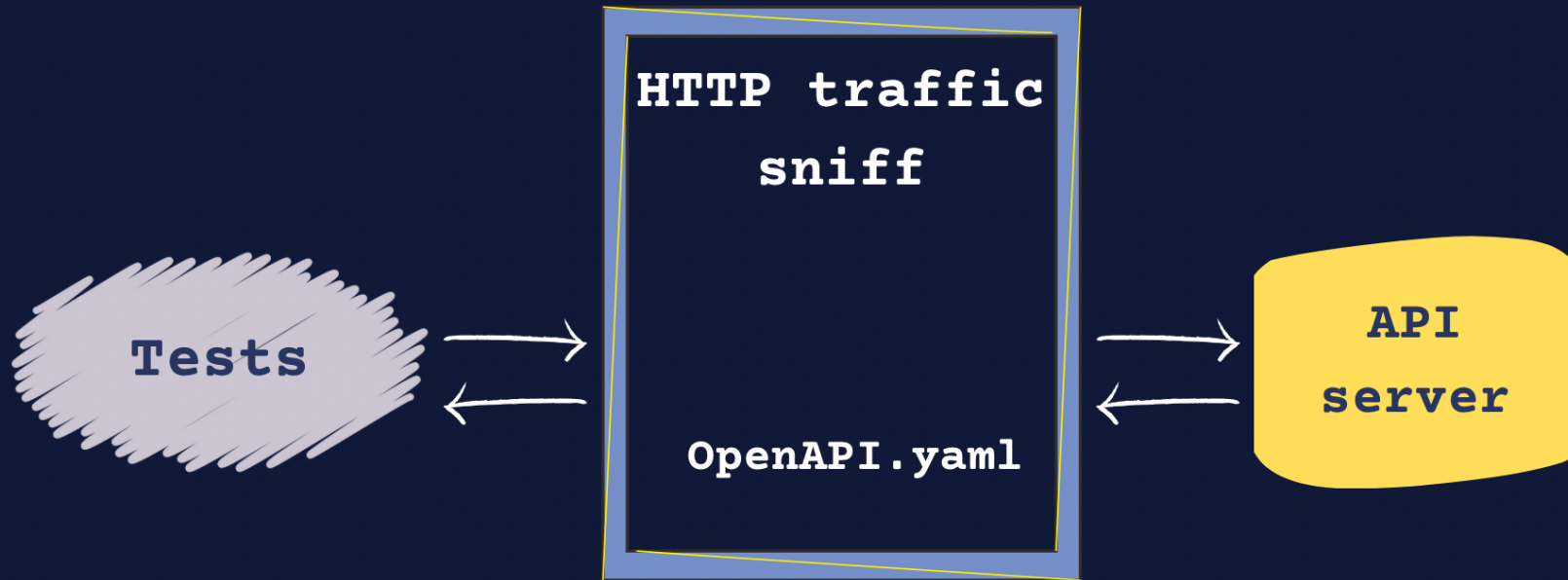
If you wrote the API code already

Get an OpenAPI file of what you have, and then start using it design-first.

Two great options:

- generate from your codebase
- use a learning/sniffing tool like Optic

If you wrote the API code already



If you have humans using your API

Make the OpenAPI human-readable with docs!

- Scalar (next-generation docs)
- Redoc (traditional docs)
- Bump.sh (short-lived hosted docs)

If you have humans using your API

The image shows the Scalar API client interface for the `/users` endpoint. On the left is a sidebar with a search bar and a list of endpoints: `/events`, `/users` (highlighted), `/users/{user}`, `/events/{event}`, and `/events/{event}/tracks`. The main area displays the endpoint `/users` with a description: "A full user list, with pagination". Below this, the "Responses" section shows a `200` response with a schema for an array of user objects. The schema includes fields like `username`, `full_name`, `biography`, `twitter_username`, `uri`, `verbose_uri`, `website_uri`, `talks_uri`, and `attended_events_uri`. On the right, a "Test Request" button is visible, and below it, a JSON response for the `200` status is shown, containing a list of user objects with various fields.

Search ^{⌘K}

- `/events` GET
- `/users` GET
- `/users/{user}` GET
- `/events/{event}` GET
- `/events/{event}/tracks` GET

[Open API Client](#)

Powered by Scalar

`/users`

A full user list, with pagination

Responses

200 200 response

object

Hide Child Attributes

users array object[] required

Hide Child Attributes

- `username` string required
- `full_name` string | null required
- `biography` ONEOF required
 - Show Child Attributes
 - Show Child Attributes
- `twitter_username` string | null required
- `uri` string required
- `verbose_uri` string required
- `website_uri` string required
- `talks_uri` string required
- `attended_events_uri` string required

GET `/users` Shell Curl ▾

1

Test Request

200 Show Schema

```
{  "users": [    {      "username": "...",      "full_name": null,      "biography": null,      "twitter_username": null,      "uri": "...",      "verbose_uri": "...",      "website_uri": "...",      "talks_uri": "...",      "attended_events_uri": "...",    }  ]}
```

200 response

If your OpenAPI needs editing

Designed OpenAPIs can be edited safely

But generated ones do have options.

- OpenAPI Overlay Specification is a standard for describing edits
- Speakeasy and Bump.sh have good Overlay tools
- Look out for other transformation tools

Repeatably edit an OpenAPI

Part 1 is to prepare the Overlay:

- Copy the the OpenAPI file, and edit the copy
- Use `speakeasy compare` creates a diff as an Overlay
- Web GUI: <https://overlay.speakeasy.com>

Part 2 is to apply it in every build, locally and in CI

- Try the `bump` or `speakeasy` CLI tools.

Repeatably edit an OpenAPI

```
YAML document using jsonpath.

Original [ ]      Original + Overlay [ ]      Overlay [ ]
24 |           name: start
25 |           required: false
26 | /users:
27 |   get:
28 |     description: A full user list
29 |     responses:
30 |       "200":
31 |         description: 200 respons
32 |         content:
33 |           application/json; char
34 |           schema:
35 |             $ref: "#/component
36 | /users/{user}:
37 |   parameters:
38 |     - in: path
39 |       name: user
40 |       required: true
41 |       schema:
42 |         type: string
43 |   get:
44 |     responses:
45 |       "200":
46 |         description: 200 respons
47 |         content:
48 |           application/json; char
49 |           schema:
25 |           required: false
26 | /users:
27 |   get:
28 |     summary: Get user list
29 |     description: A full user list, with
30 |     responses:
31 |       "200":
32 |         description: 200 response
33 |         content:
34 |           application/json; charse
35 |           schema:
36 |             $ref: "#/compon
37 | /users/{user}:
38 |   parameters:
39 |     - in: path
40 |       name: user
41 |       required: true
42 |       schema:
43 |         type: string
44 |   get:
45 |     responses:
46 |       "200":
47 |         description: 200 response
48 |         content:
49 |           application/json; charse
1 | overlay: 1.0.0
2 | info:
3 |   title: example overlay
4 |   version: 0.0.0
5 | actions:
6 |   - target: $["paths"]["/users"]
7 |     update:
8 |       summary: Get user list
9 |
```

If you have code using your API



If you like API quality and consistency

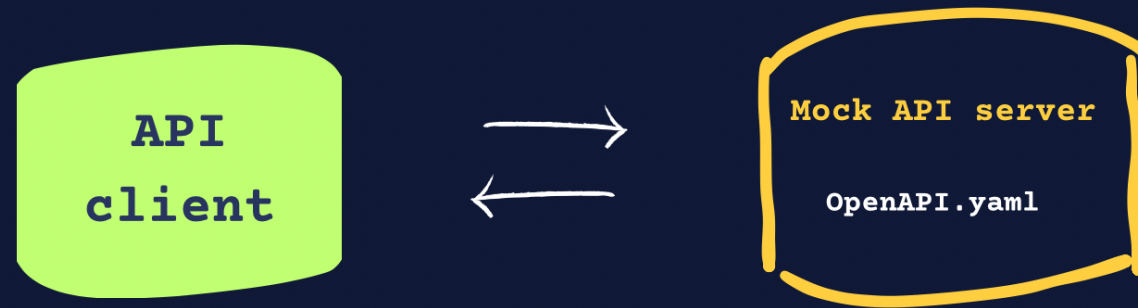
API standards are a document. Use linting to check:

- plurals and casing
- every operation has an ID
- everything named *-date uses the expected format
- errors are defined and in the expected format

Try: Spectral or Vacuum.

If an API sandbox would be useful

Use an API Mock Server tool.



Tools include: Microcks, Prism

If an API sandbox would be useful

The screenshot displays the Microcks API sandbox interface. The top navigation bar includes a menu icon, the Microcks logo, and a help icon. The left sidebar contains navigation options: Dashboard, APIs | Services (highlighted), Importers, Microcks Hub, and Administration.

The main content area shows a selected API endpoint: `GET /pastry/{name}`. It indicates the method is `GET` with a `URI_PARTS` dispatcher and 3 sample(s). Below this, a "DISPATCHING PROPERTIES" section shows the dispatcher is `URI_PARTS` (with a "Learn More" link) and the HTTP verb is `GET`. The dispatching rule is `name` and the default delay is `0 ms`.

The "MOCKS" section shows three mock configurations: `Eclair Cafe` (selected), `Millefeuille`, and `Eclair Cafe Xml`.

The "Request" section shows the "Mock URL" as `http://localhost:8585/rest/API+Pastry+-+2.0/2.0.0/pastry/Eclair`. Below it is a table of headers:

Header name	Values
Accept	application/json

The "Response" section shows the "Response Code and Type" as `200: application/json`. The response body is a JSON object:

```
{
  "name": "Eclair Cafe",
  "description": "Delicieux Eclair au Cafe pas calorique du tout",
  "size": "M",
}
```

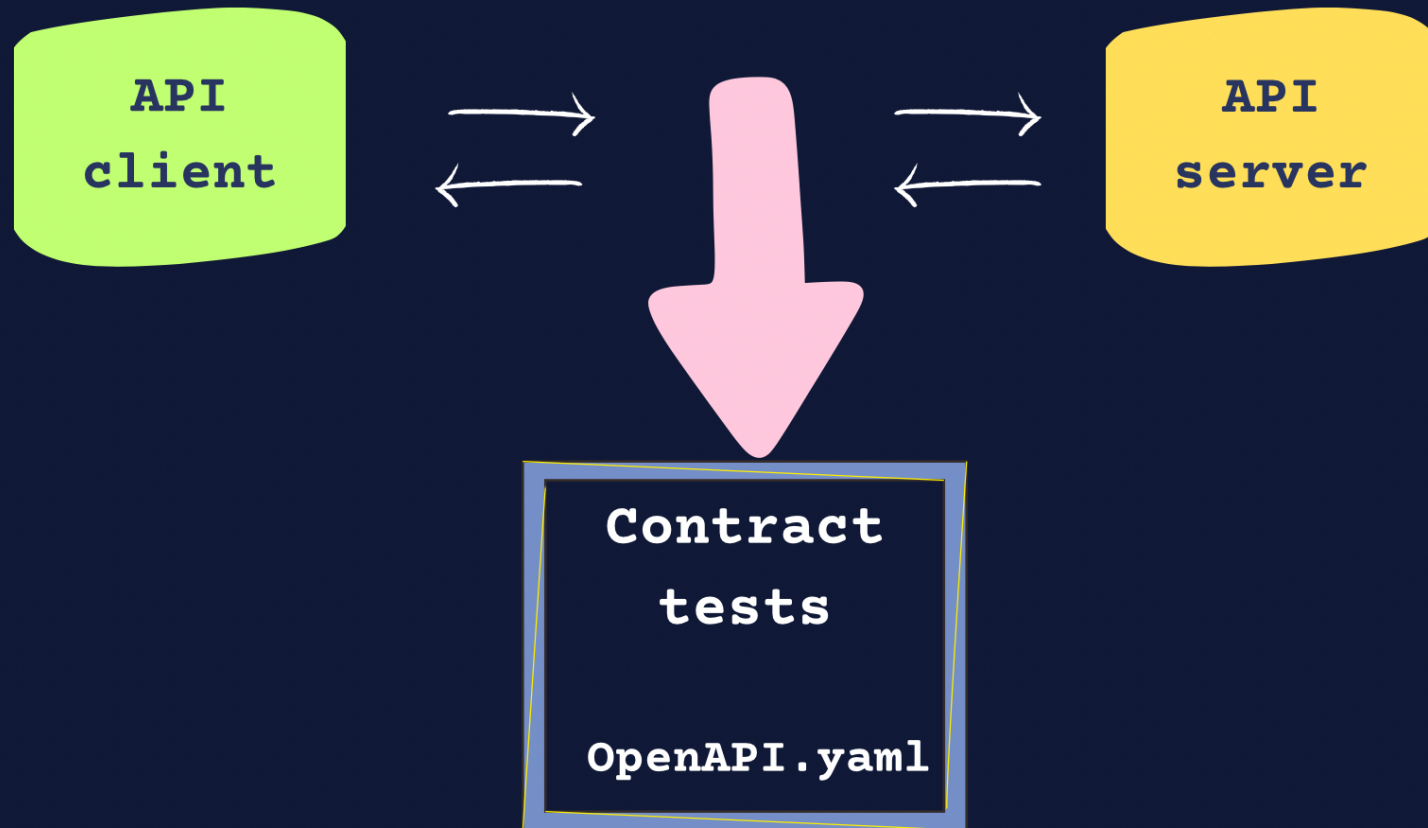
To avoid API drift

Contract testing is your friend!

Check that what the API does matches what is described.

Tools include: Microcks, WireMock, Pact

To avoid API drift



API Description Pipelines

OpenAPI isn't a static asset.

OpenAPI Community

OpenAPI Community

- Part of the Linux Foundation
- Standard is developed in the open
- <https://openapis.org>
- Public GitHub repository
- Active Slack groups
- Weekly technical meetings

OpenAPI Standards

- Overlays map repeatable amendments to an OpenAPI file (v1.0.0)
- Arazzo describe a sequence of API calls (v1.0.1)
- OpenAPI describes your API for all the tools to use
 - stable: 3.1.1
 - planned: 3.2.0
 - early days: 4.0 "Moonwalk"

OpenAPI

Open standard for API descriptions

Resources

- <https://lornajane.net>
- <https://openapis.org>
- <https://github.com/opticdev/optic>
- <https://speakeasy.com>
- <https://bump.sh>
- <https://microcks.io/>
- <https://apisyouwonthate.com/>