

Documenting APIs:

new era, same craft



by Javier Rubio



@jrubr



/javi-rubio

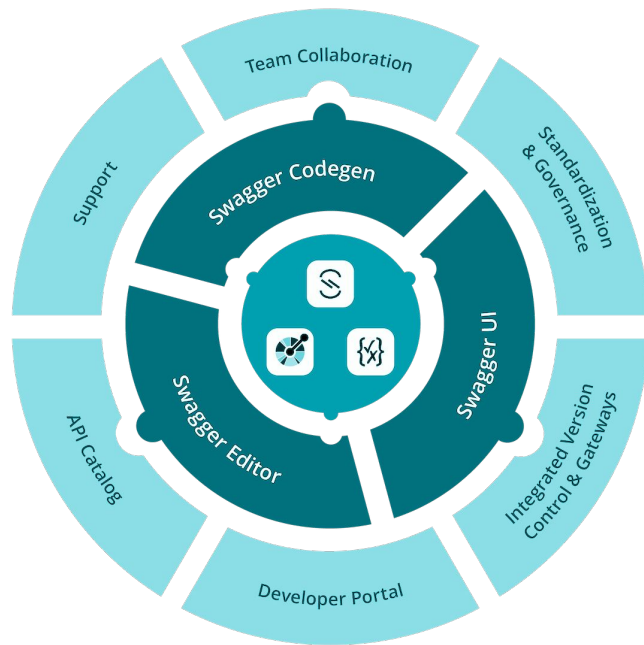


/jrub

www.javirubio.net

Documenting APIs: new era, same craft

API Docs ecosystem



Thirty years of APIs

SOAP

1998

Strictly typed

Contract: WSDL

- Verbose XML
- Hard to read
- + Client code auto-gen

REST

2000

Barely typed

No contract

- + Simple to start
- ± Flexibility
- **Requires docs**

GraphQL

2012

Strictly typed

Contract: schema

- + One endpoint
- Caching is hard
- Steep curve

gRPC

2015

Strictly typed

Contract: .proto

- + Fast
- Binary format
- ± Mostly Internal use

Insight

The contracts
are the docs



API docs history

```
1 openapi: 3.1.0
2 info:
3   title: Swagger API
4   version: 1.0.0
5 servers:
6   - url: https://api.swagger.io
7 paths:
8   /resources:
9     get:
10      tags:
11        - API Resources
12      summary: Retrieve all resources based on filter criteria
13      description: Fetches a list of all available resources.
14      responses:
15        '200':
16          description: A
17          content:
18            application/json:
19              schema:
20                type: array
21                items:
22                  $ref: '#/components/schemas/resource'
23      post:
24        tags:
25          - API Resources
26        summary: Create a new resource
27        description: Create a new resource
28        requestBody:
29          required: true
30          content:
31            application/json:
32              schema:
33                $ref: '#/components/schemas/resource'
```

Servers

https://api.swagger.io ▼

API Resources

GET /resources Retrieve all resources based on filter criteria

POST /resources Create a new resource

GET /resources/{resourceId} Retrieve a resource

PUT /resources/{resourceId} Update a resource

PATCH /resources/{resourceId} Partially update a resource

DELETE /resources/{resourceId} Delete a resource

OPTIONS /resources/{resourceId} Retrieve options

HEAD /resources/{resourceId} Check resource presence



Swagger (OpenAPI)

A standardized way to **describe REST APIs** in **JSON** or **YAML**.

Designed to fill the gaps of REST's flexibility.

Auto-generates **API reference** docs and an interactive **UI**.

Looked like the ultimate solution... but it wasn't.

Documenting *your* API conventions

Neither REST nor OpenAPI standardize any of this:

- ➡ Authentication
- ➡ Error handling
- ➡ Pagination
- ➡ Rate limiting
- ➡ Versioning

OpenAPI ecosystem

Webhooks poorly supported until recent versions.

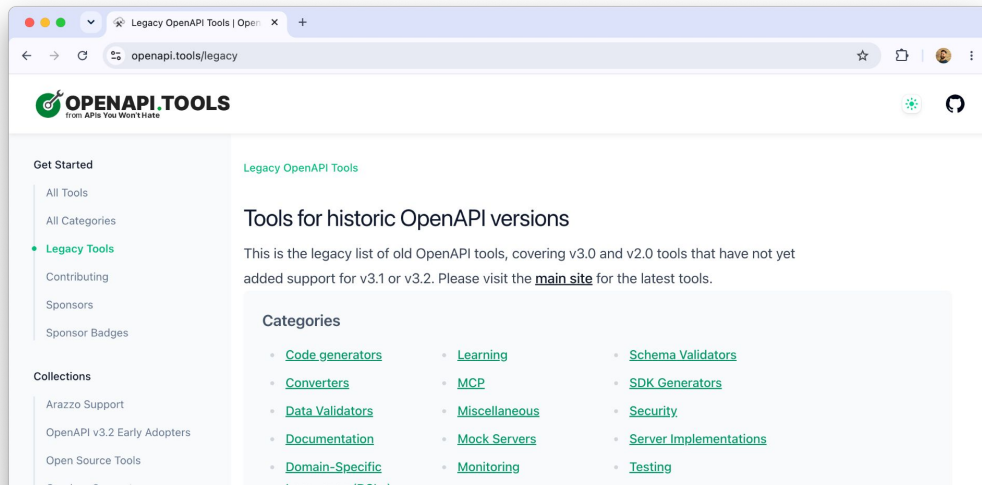
Disputes over **JSON Schema** for stricter specs.

Ecosystem fragmented with **proprietary extensions** (e.g. x-tags).

New standards keep filling the gaps: **Arazzo, AsyncAPI, Overlays**.

Conway's Law

Organizations which design systems (in the broad sense used here) are constrained to produce designs which are copies of the communication structures of these organizations.



API Docs: Tooling ecosystem





WRITE2
THE0
DOCS2
BERLIN6

There's no silver bullet

Endless tools, either laser-focused or “we-do-it-all”.

None complete (for *your needs*).

Proprietary ecosystems.

Expensive monthly subscriptions.

API Docs example: Map your needs

Platform	Free Features	Paid Features	Missing features	Pricing/month	Notes	URL
Bump.sh	auto-upload-openapi custom-domain extra-docs webhooks	auto-changelogs	endpoint-grouping try-it-editor	\$99.00	Super simple. Basic UI custom. Asked them to implement nested groups.	bump.sh
Stoplight	try-it-editor endpoint-grouping extra-docs webhooks	custom-domain auto-upload-openapi		\$79.00	*still not sure how to sync open-api Super complex Openapi spec editor. Super customizable UI (complex)	stoplight.io
Stoplight Elements (open-source)	endpoint-grouping webhooks	extra-docs		\$79.00	Stupidly enough, the markdown articles feature must be linked to their paid portal.	github.com/stoplight-elements
Gitbook	extra-docs endpoint-grouping custom-domain		auto-upload-openapi try-it-editor webhooks	\$6.70	*endpoint-groups are not collapsible	gitbook.com/
Redocly	webhooks try-it-editor auto-upload-openapi	extra-docs custom-domain		\$69.00	They do not support <code>webhooks</code> elements from the OpenAPI standard @jas per conversation with Support as why our swagger file is failing after disabling validation.	redocly.com/
Readme	try-it-editor auto-upload-openapi	extra-docs auto-changelogs custom-domain	webhooks	\$99.00		readme.com/
Mintlify	auto-upload-openapi extra-docs try-it-editor	custom-domain		\$120.00	not sure about webhooks support	mintlify.com/
SwaggerHub			webhooks try-it-editor		Basic Swagger-UI	swagger.io/features/
Template-based	extra-docs endpoint-grouping			\$99, one-time payment.	After looking at the code (a friend sent it to me), it does not provide a react/Next.js component that automatically feeds from our swagger file, so we'd have to program one :-(	tailwindui.com/templates/docs
ApiGit	extra-docs endpoint-grouping	custom-domain	try-it-editor auto-upload-openapi	\$8.00	Our swagger fails rendering here.	apigit.com
Open Documenter (open-source)			webhooks	\$0.00	No commits on 3 years, no OpenAPI 3.1 (webhooks) Next.js (Vue.js for statics)	github.com/our_mentor
Zuplo					OpenAPI import is under development :-(	zuplo.com/
RapIDoc (open-source)	custom-domain endpoint-grouping			\$0.00		github.com/rap_apiDoc
ApiTree				\$5.60		apitree.com/
DeveloperHub						developerhub.io/pricing
HelloAPI	try-it-editor endpoint-grouping custom-domain			\$0.00	Features under development: - CI auto-sync - extra-docs	helloapi.co
FUMADOCs	try-it-editor			\$0.00	Added in Nov 2024. Repo: https://github.com/tuna-nama/fumadocs	fumadocs.vercel.app
Scalar	try-it-editor			\$12.00	Added in Nov 2024. Has open-source version: https://github.com/colson/scalar	scalar.com

Choose tools that fit your workflow, not the other way around.

API Docs example: Map your needs

Table +

</> Platform	📁 Free Features	💰 Paid features	⚙️ Missing features	💵 Pricing
🚫 Bump.sh	auto-upload-openapi custom-domain extra-docs webhooks	auto-changelog	endpoint-grouping try-it-editor	
🚧 Stoplight	try-it-editor endpoint-grouping extra-docs webhooks	custom-domain auto-upload-openapi		
🚧 Stoplight Elements (open-source)	endpoint-grouping webhooks	extra-docs		
🚫 Gitbook	extra-docs endpoint-grouping custom-domain		auto-upload-openapi try-it-editor webhooks	
🚫 Redocly	webhooks try-it-editor auto-upload-openapi	extra-docs custom-domain		
📖 Readme	try-it-editor auto-upload-openapi	extra-docs auto-changelog custom-domain	webhooks	💬
🚧 Mintlify	auto-upload-openapi extra-docs try-it-editor	custom-domain		

Research based on your needs 🙏

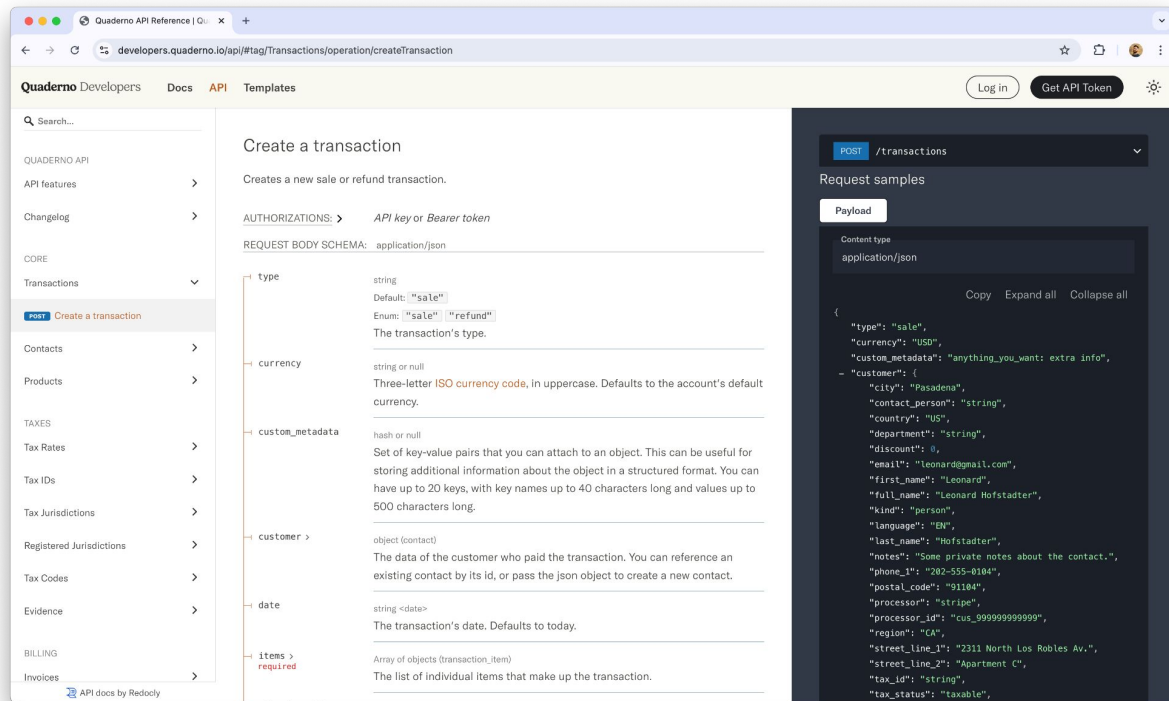
Don't let tools model your behaviour, model your tools

Traditionally, some teams preferred **static sites** from templates.

Which was flexible, but high-maintenance.

AI might help here...

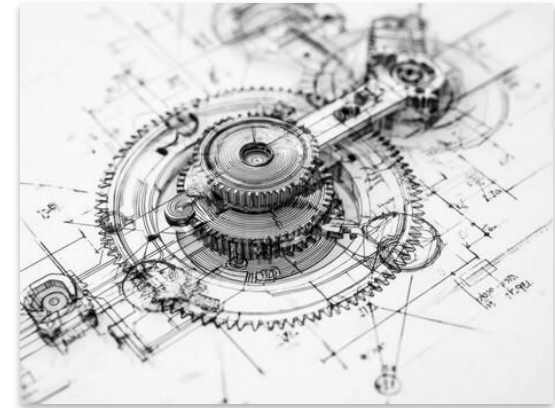
Example: Quaderno API reference



Docusaurus (guides) + **redocusaurus** plugin (API reference)

Documenting APIs: new era, same craft

Engineering Lens



Insight

The closer to the code,
the better



The code is the “source of truth”

- + Lower maintenance toll
- + Easier **shared ownership**

 **Docs should ship with the code** whenever the API changes

Where does the spec come from?

Hand-written spec

Hard to maintain. Linters might help. **Who owns it?**

Captured traffic

A proxy watches real API calls. **Who owns the pipeline?**

Annotations

Next to the code, still one more thing to update.

Automated Tests

Tests are code. Must go red when the API changes.

Framework introspection

Your **code is the spec**! If your language or framework supports it.

Where can AI help?

Hand-written spec

“Big bang risk”: generate it all, review none. For maintenance, specialized agents might help reviewing Pull requests!

Captured traffic

Low risk, low value.

Annotations

Easy for AI to write or PR review. CI will never go red.

Automated Tests

Using AI to write these tests > using AI to write the OpenAPI spec

Framework introspection

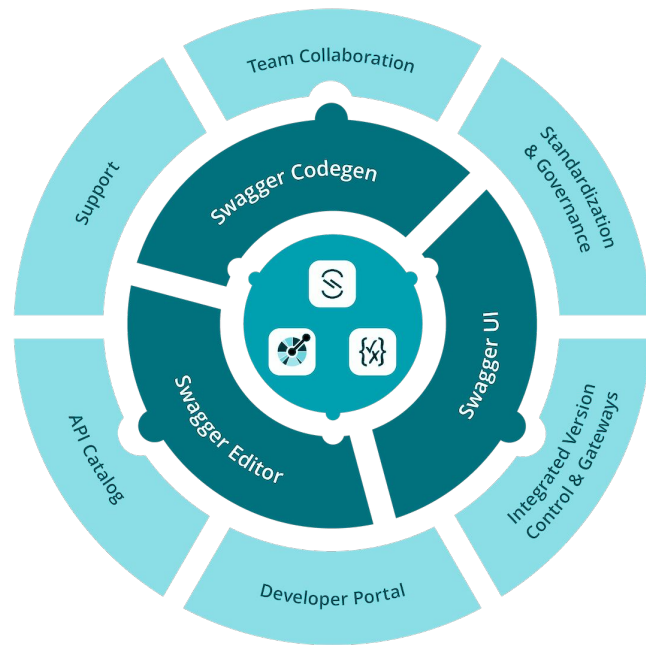
Deterministic tooling already does this well.

Famous OpenAPI libs per language

Ruby	<u>rswag</u> , <u>rspec-openapi</u>	tests
Python	<u>FastAPI</u> , <u>drf-spectacular</u>	introspection
Java, Kotlin	<u>springdoc-openapi</u> , <u>Ktor</u>	introspection + annotations
Go	<u>swaggo/swag</u>	annotations
Node	<u>@nestjs/swagger</u> , <u>swagger-jsdoc</u>	annotations
PHP	<u>swagger-php</u>	annotations
.NET	<u>built-in since .NET 9</u>	introspection

Documenting APIs: new era, same craft

Product Lens



Insight

Your API is a product
Your Docs too!



Know your users. Talk to them.

Ideally,

- ➡ Map your API user personas.
- ➡ Understand their needs and emotions when they start integrating.
- ➡ *Tip:* help them integrate your API, you will learn a lot.

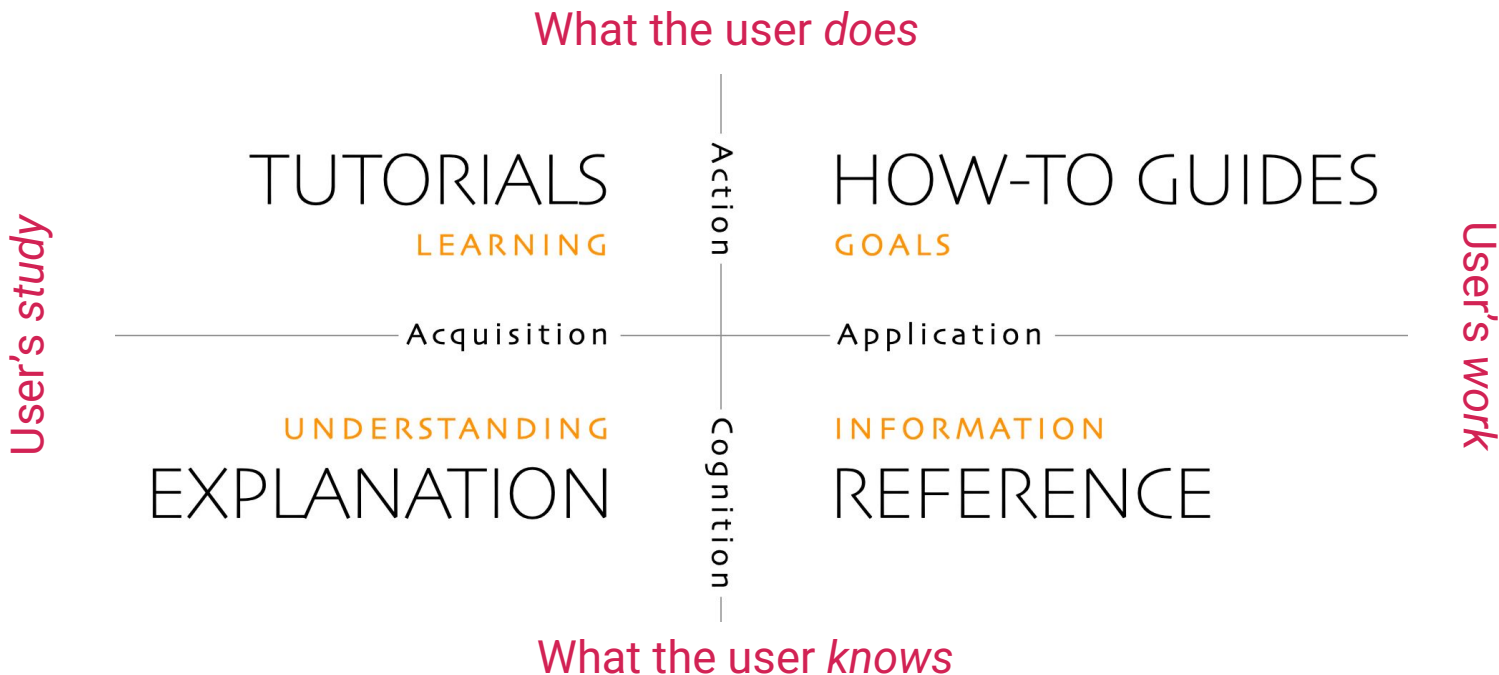
Then, **write guides that help your users achieve *their goals*.**

Developers value predictability

- ➡ Testing environments matter (**Sandbox**)
- ➡ Publish a **versioning policy**
- ➡ **Communicate breaking changes**
- ➡ Publish **migration guides**
- ➡ Keep a **Changelog**

AI too, but developers can veto buy-in!

Product lens: Diátaxis framework (Daniele Procida)



Product lens: Documentation as an organism



<https://diataxis.fr/how-to-use-diataxis/#complete-not-finished>

Arazzo helps describe use cases

Arazzo describes **workflows**, not just endpoints.

Example:

to issue a refund → find the customer → find the charge → create the refund.

Great for “How-to” guides, and to avoid AI guessing

Who is actually reading your docs?

The web overall Cloudflare, 2026

Non-human 50%+

Human

Documentation sites Mintlify, July 2026

Agents 66%

Human

AI is the new API Docs user

New use cases:

- ➡ Humans asking AI
- ➡ Fully autonomous agents

New interfaces:

- ➡ MCPs
- ➡ CLIs

lms.txt cuts the traffic overhead

404s per task · 2,400 agent runs across 20 docs sites

HTML

2.23

Markdown

1.42

Markdown + lms.txt

0.11

Wrapping-up

Treat your Docs as a Product.

Shared ownership matters more than ever.

**Your API is your product's interface
to the world.**

**Your Docs are your API's interface
to the world.**

Questions & Answers

Thanks
for your time!



@jrubr



/javi-rubio



/jrub

www.javirubio.net