

Automated Testing

API development and testing



**[https://github.com/up1/
workshop-api-first](https://github.com/up1/workshop-api-first)**



Automated Testing for REST APIs



Delivery process

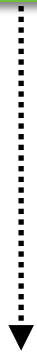
Simple process

API specification

Development

Deployment

Testing

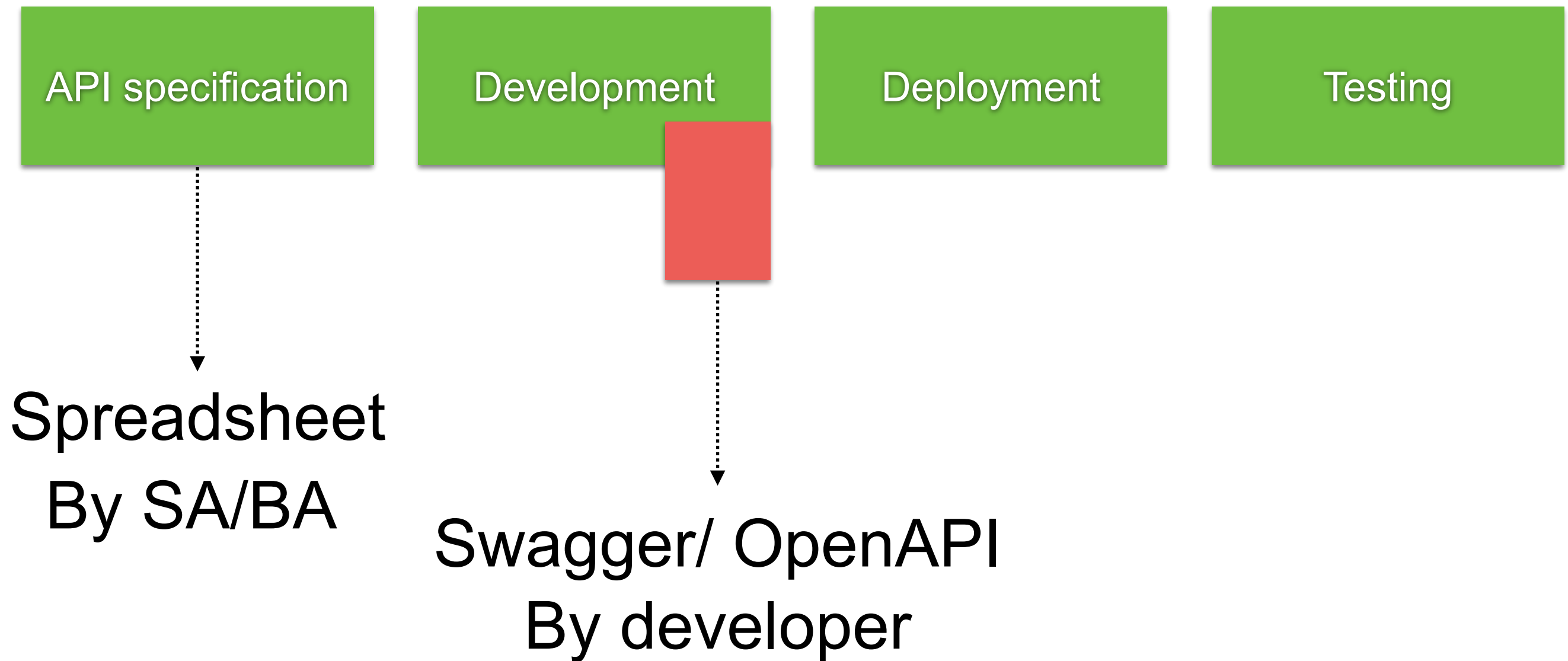


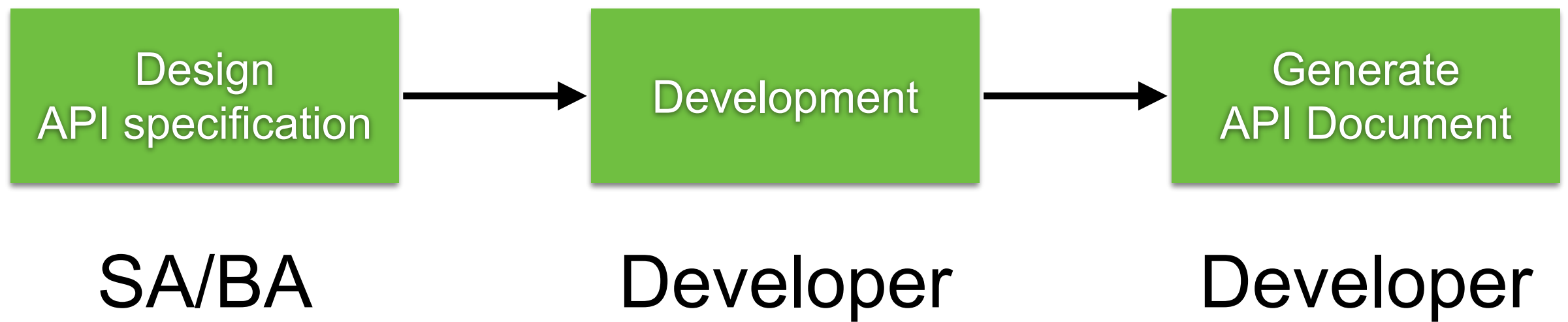
Spreadsheet
By SA/BA



Delivery process

Need API documentation

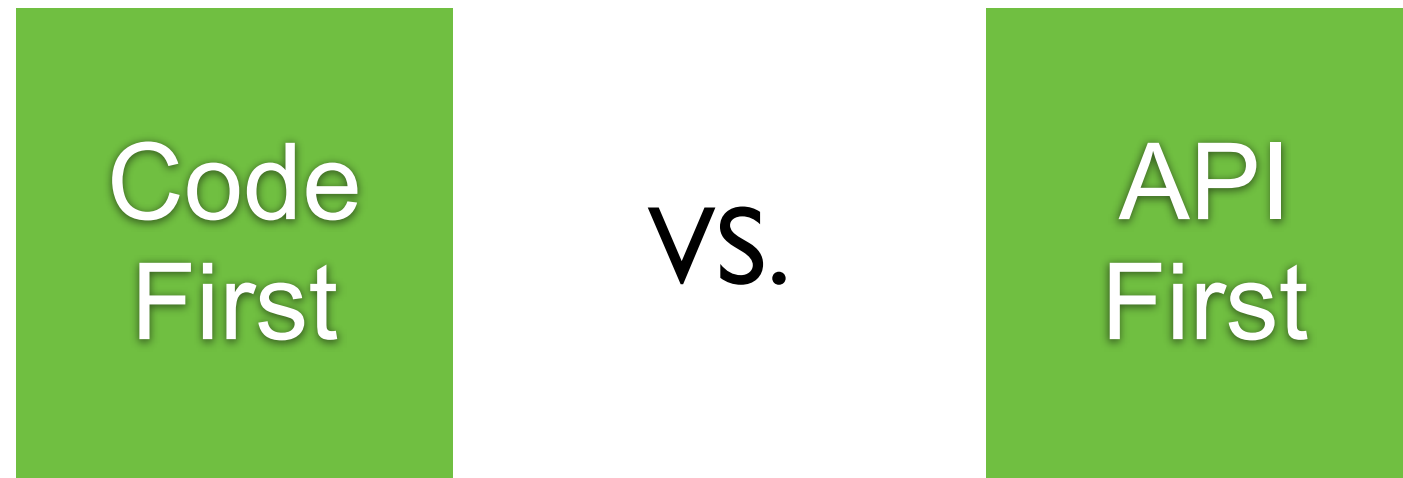




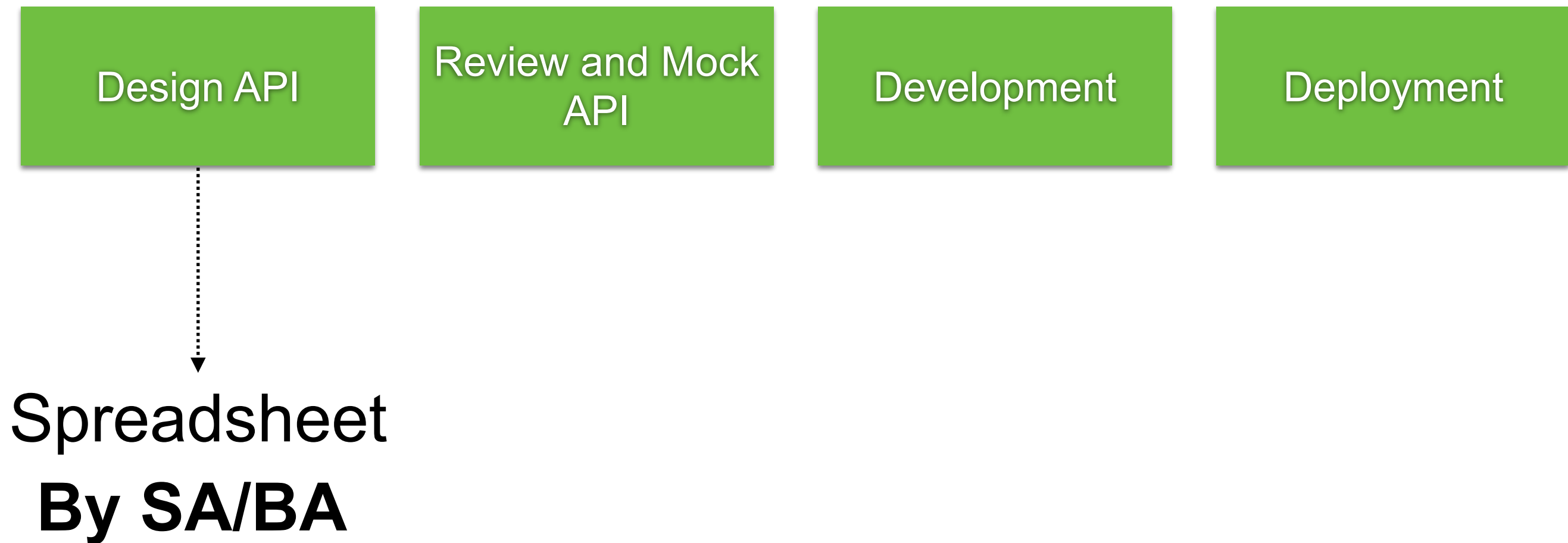
Spec vs Code Sync ?



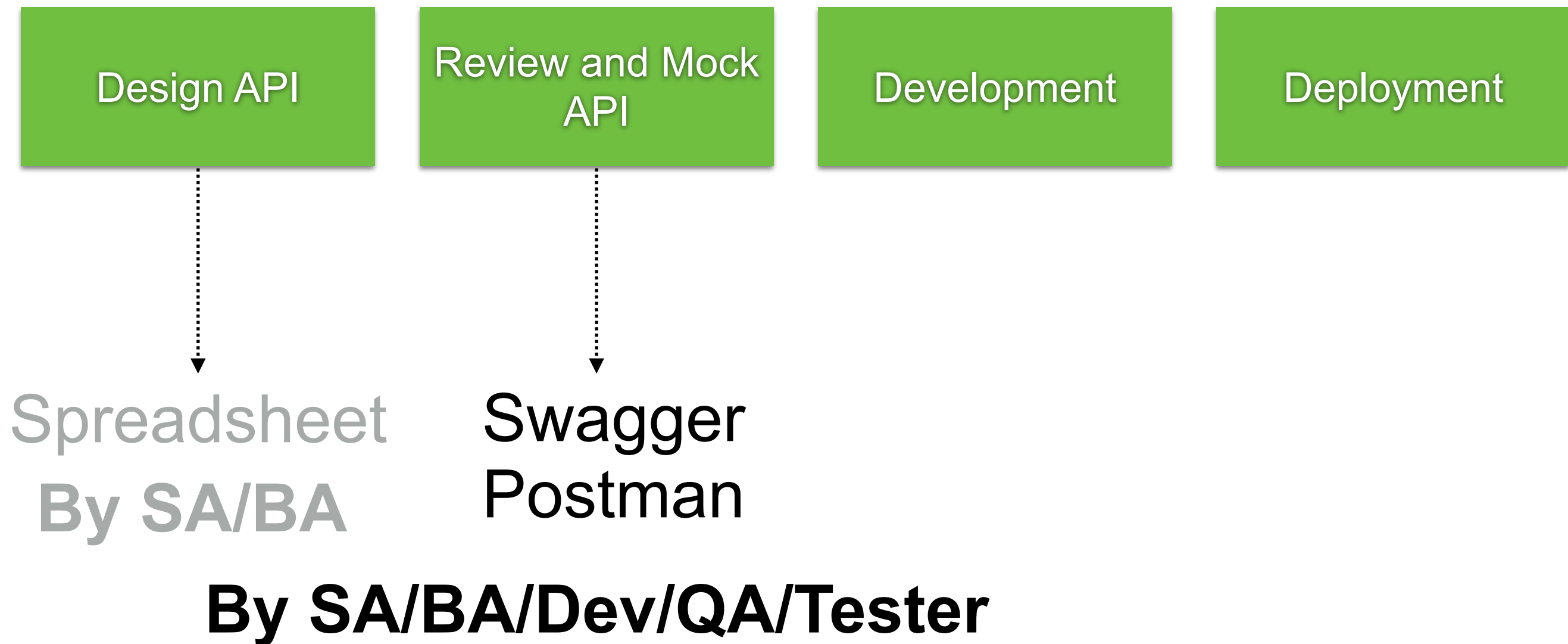
Code-first vs API-first development



API-first development



API-first development

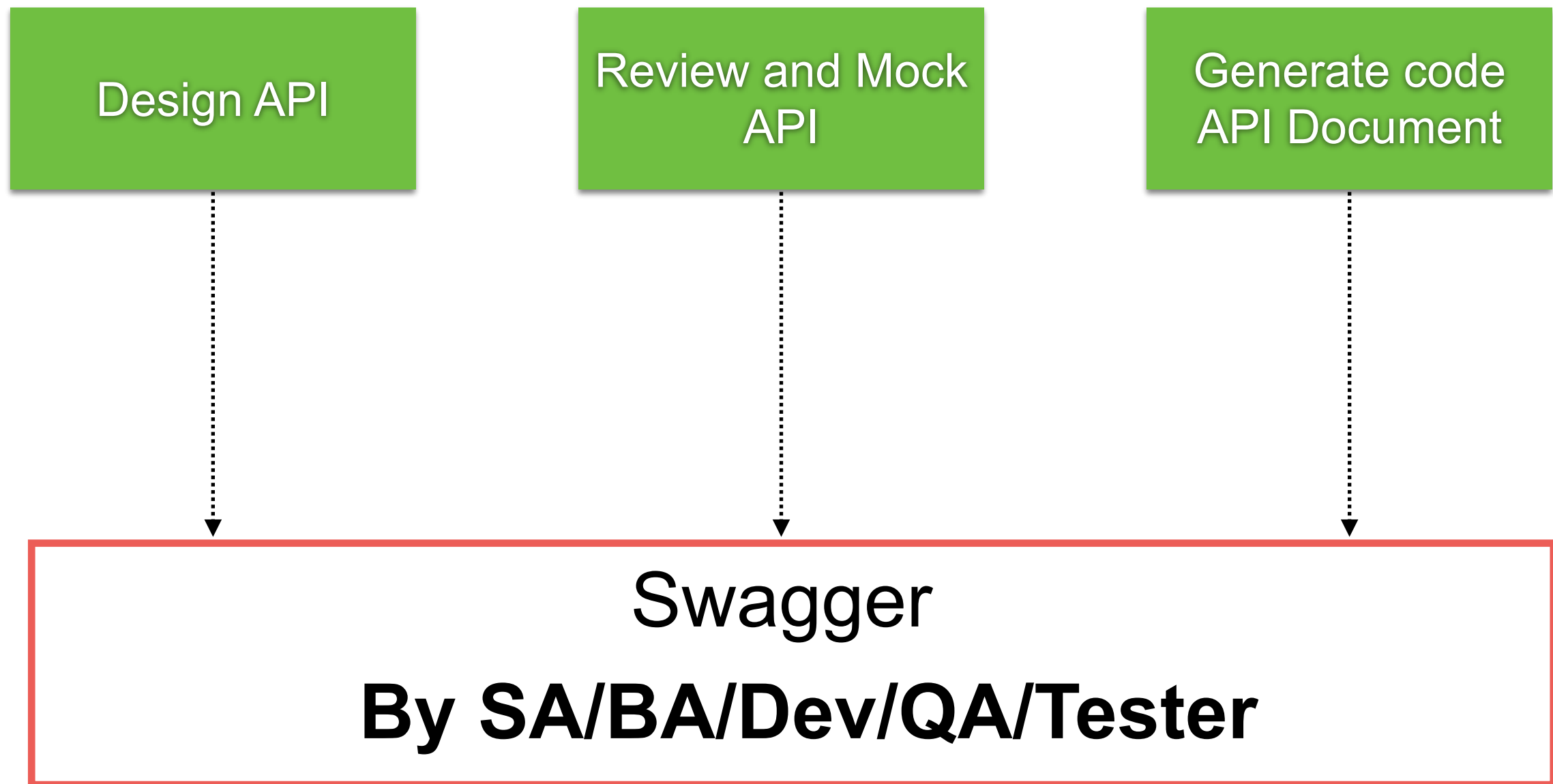


Working with Swagger

<https://swagger.io/>



Process with Swagger



What id OpenAPI ?

What Is OpenAPI?

OpenAPI Specification (formerly Swagger Specification) is an API description format for REST APIs. An OpenAPI Specification describes your entire API, including:

- Available endpoints (`/users`) and operations on each endpoint (`GET /users`, `POST /users`)
- Operation parameters Input and output for each operation
- Authentication methods
- Contact information, license, terms of use and other information.

API specifications can be written in YAML or JSON. The format is easy to learn and readable to both humans and machines. The OpenAPI Specification can be found on GitHub: [OpenAPI 3.0 Specification](#)

What Is Swagger?

Swagger is a set of open-source tools built around the OpenAPI Specification that can help you design, build, and test your APIs. The major Swagger tools include:

<https://swagger.io/docs/specification/about/>



Learn OpenAPI Specification

OpenAPI Specification

Version 3.1.0

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14 RFC2119 RFC8174](#) when, and only when, they appear in all capitals, as shown here.

This document is licensed under [The Apache License, Version 2.0](#).

Introduction

The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to HTTP APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.

An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases.

<https://swagger.io/specification/>



Step 1 :: Design API

Design API

Review and Mock
API

Generate code
API Document

Swagger Editor

<https://editor-next.swagger.io/>

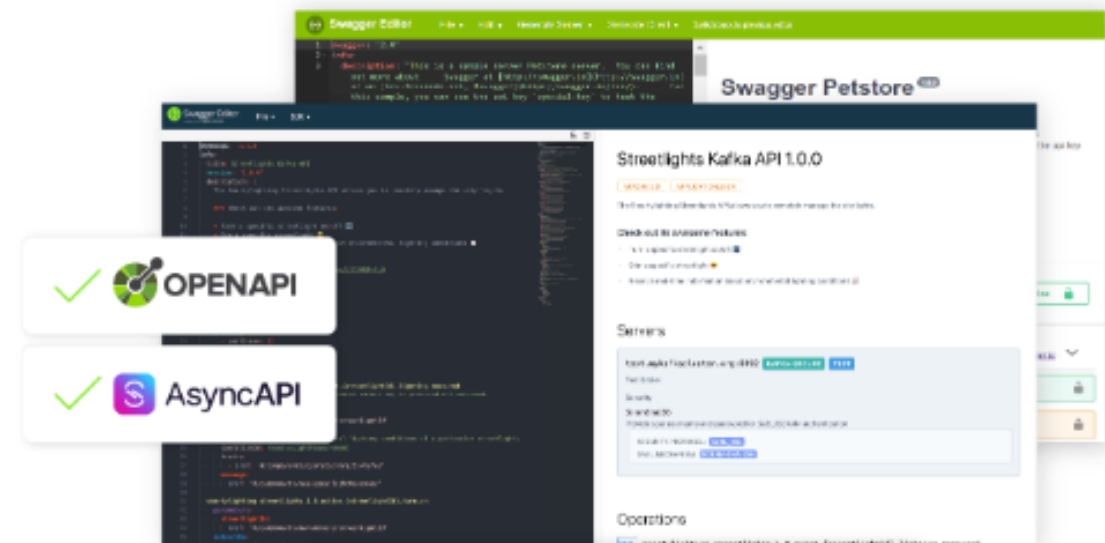


Swagger Editor

Design, describe and document API

Swagger Editor

Design, describe, and document your API on the first open source editor supporting multiple API specifications and serialization formats. The Swagger Editor offers an easy way to get started with the OpenAPI Specification (formerly known as Swagger) as well as the AsyncAPI specification, with support for Swagger 2.0, OpenAPI 3.*, and AsyncAPI 2.* versions.



<https://swagger.io/tools/swagger-editor/>



Basic Structure

Information

Server

Paths




Example

YAML or JSON

```
openapi: 3.0.3
info:
  title: Demo
  version: 1.0.0
servers:
  - url: http://localhost:3000
    variables: {}
paths:
  /ping:
    get:
      responses:
        "200":
          description: OK
```



Try in Swagger Editor

 **Swagger Editor**
Powered by SMARTBEAR

File ▾ Edit ▾ Insert ▾ Generate Server ▾ Generate Client ▾ About ▾

Try our new Editor ↗

```
1 openapi: 3.0.3
2 info:
3   title: Demo
4   version: 1.0.0
5 servers:
6   - url: http://localhost:3000
7     variables: {}
8 paths:
9   /ping:
10    get:
11      responses:
12        '200':
13          description: OK
14
15
```

Demo 1.0.0 OAS 3.0

Servers

http://localhost:3000 ▾

default ⬆

GET /ping ⬆

Parameters Try it out

No parameters

Responses

Curl

curl -X 'GET' \



Add more paths

```
paths:
  /users/{id}:
    get:
      parameters:
        - in: path
          name: id
          schema:
            type: integer
            required: true
            description: User ID of the user
      responses:
        "200":
          content:
            application/json:
              schema:
                type: object
                properties:
                  id:
                    type: integer
                  name:
                    type: string
              example:
                id: 1
                name: User 01
```

Path variable



Add more paths

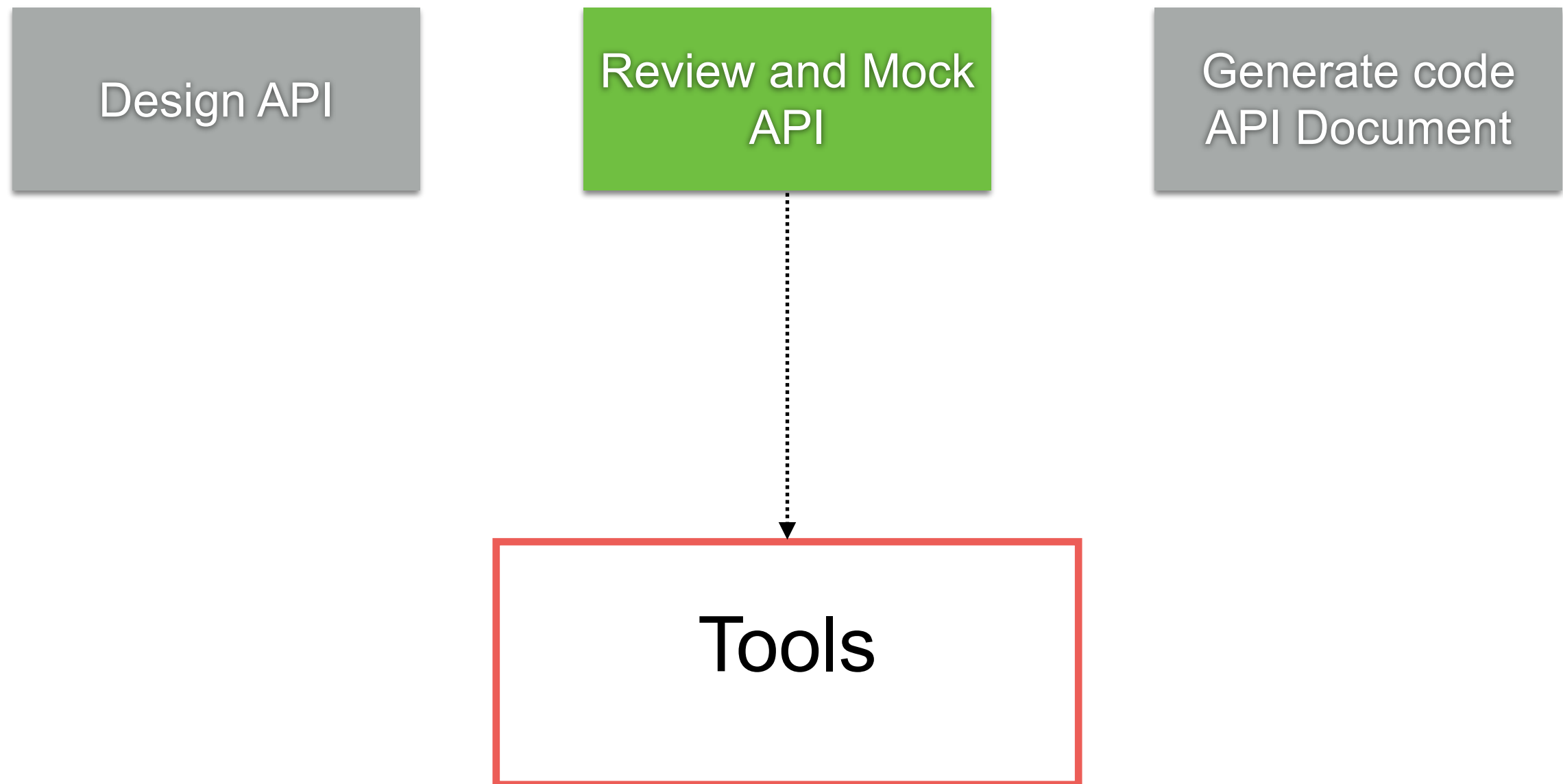
```
paths:
  /users/{id}:
    get:
      parameters:
        - in: path
          name: id
          schema:
            type: integer
            required: true
            description: User ID of the user
```

```
responses:
  "200":
    content:
      application/json:
        schema:
          type: object
          properties:
            id:
              type: integer
            name:
              type: string
          example:
            id: 1
            name: User 01
```

Response
Schema
Example

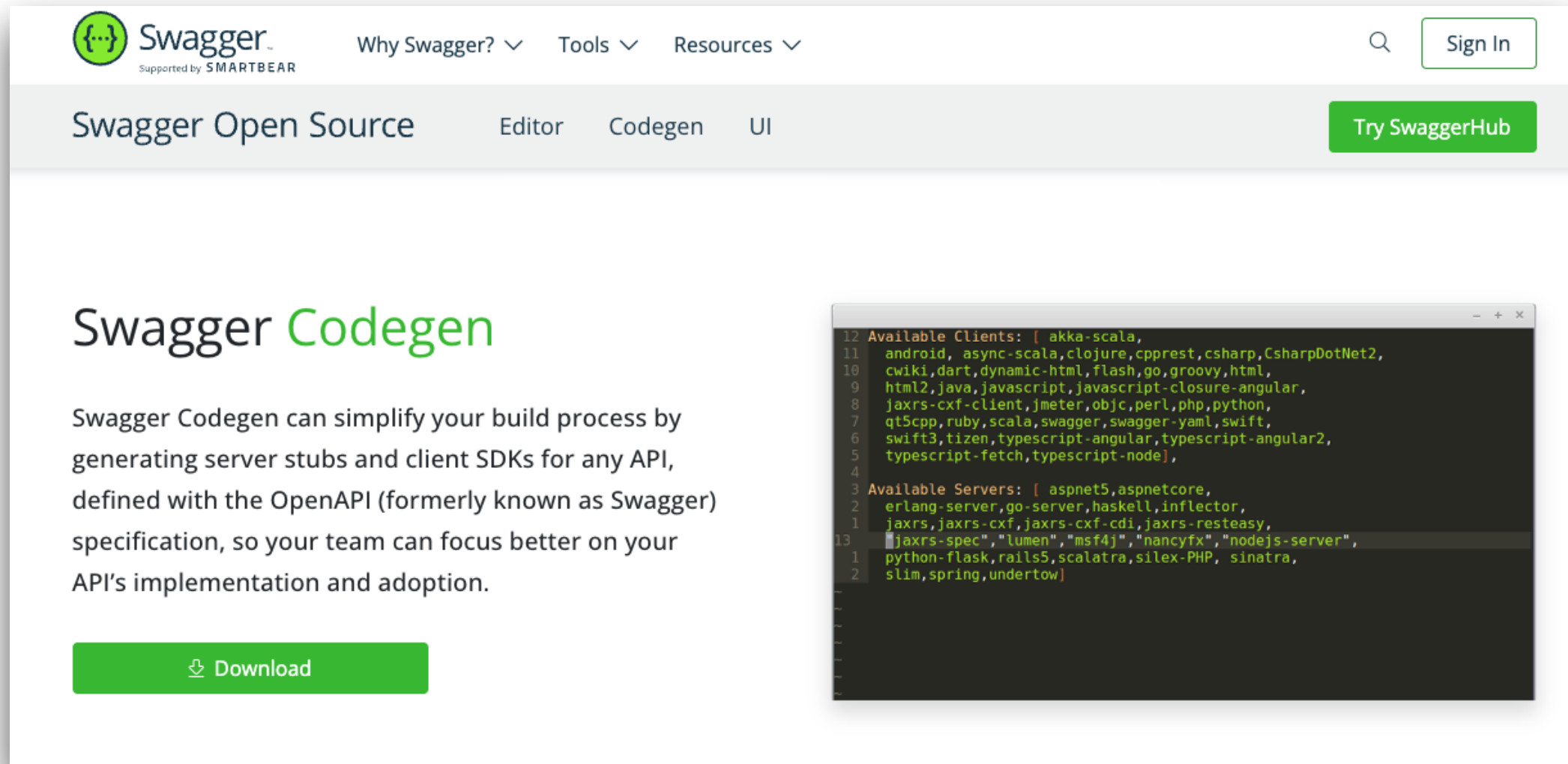


Step 2 :: Mock API



Swagger Codegen

Generate HTML, Server stub and client



The screenshot shows the Swagger Codegen website. The header includes the Swagger logo (a green circle with curly braces), the text "Swagger" and "Supported by SMARTBEAR", and navigation links: "Why Swagger?", "Tools", and "Resources". There is a search icon and a "Sign In" button. Below the header is a navigation bar with "Swagger Open Source", "Editor", "Codegen", and "UI". A green button labeled "Try SwaggerHub" is on the right. The main content area has the heading "Swagger Codegen" in green. Below it, a paragraph states: "Swagger Codegen can simplify your build process by generating server stubs and client SDKs for any API, defined with the OpenAPI (formerly known as Swagger) specification, so your team can focus better on your API's implementation and adoption." A green "Download" button is below the text. To the right, a code editor window displays two lists of available clients and servers. The "Available Clients" list includes akka-scala, android, async-scala, clojure, cpprest, csharp, CsharpDotNet2, cwiki, dart, dynamic-html, flash, go, groovy, html, html2, java, javascript, javascript-closure-angular, jaxrs-cxf-client, jmeter, objc, perl, php, python, qt5cpp, ruby, scala, swagger, swagger-yaml, swift, swift3, tizen, typescript-angular, typescript-angular2, typescript-fetch, and typescript-node. The "Available Servers" list includes aspnet5, aspnetcore, erlang-server, go-server, haskell, inflector, jaxrs, jaxrs-cxf, jaxrs-cxf-cdi, jaxrs-resteasy, jaxrs-spec, lumen, msf4j, nancyfx, nodejs-server, python-flask, rails5, scalatra, silex-PHP, sinatra, slim, spring, and undertow.

Swagger Codegen

Swagger Codegen can simplify your build process by generating server stubs and client SDKs for any API, defined with the OpenAPI (formerly known as Swagger) specification, so your team can focus better on your API's implementation and adoption.

Download

```
12 Available Clients: [ akka-scala,
11 android, async-scala, clojure, cpprest, csharp, CsharpDotNet2,
10 cwiki, dart, dynamic-html, flash, go, groovy, html,
9 html2, java, javascript, javascript-closure-angular,
8 jaxrs-cxf-client, jmeter, objc, perl, php, python,
7 qt5cpp, ruby, scala, swagger, swagger-yaml, swift,
6 swift3, tizen, typescript-angular, typescript-angular2,
5 typescript-fetch, typescript-node],
4
3 Available Servers: [ aspnet5, aspnetcore,
2 erlang-server, go-server, haskell, inflector,
1 jaxrs, jaxrs-cxf, jaxrs-cxf-cdi, jaxrs-resteasy,
13 jaxrs-spec, "lumen", "msf4j", "nancyfx", "nodejs-server",
1 python-flask, rails5, scalatra, silex-PHP, sinatra,
2 slim, spring, undertow]
```

<https://swagger.io/tools/swagger-codegen/>



Installation and how to use !!

```
$brew install swagger-codegen
```

```
$swagger-codegen generate -i api.yml -o nodejs  
-l nodejs-server
```

<https://github.com/swagger-api/swagger-codegen#prerequisites>



OpenAPI Mocker

```
$npm i -g open-api-mocker  
$open-api-mocker -s api.yml -w
```

Access to <http://localhost:5000>

<https://github.com/jormaechea/open-api-mocker>



Step 3 :: Generate !!

Design API

Review and Mock
API

Generate code
API Document



Tools



Generate API Document

Design API

Review and Mock
API

Generate
API Document



Tools



Generate API Document

Generate HTML from Swagger



```
$npm i -g open-api-mocker  
$open-api-mocker -s api.yml -w
```

<https://github.com/Redocly/redoc#redoc-cli>



Example

The image displays two side-by-side screenshots of the Redoc API documentation interface, comparing light and dark themes.

Left Screenshot (Light Theme):

- Search Bar:** Search...
- Endpoints List:**
 - GET /ping
 - GET /users/{id}
- /ping Endpoint:**
 - Responses:**
 - 200 OK
- /users/{id} Endpoint:**
 - PATH PARAMETERS:**
 - id (integer, required): User ID of the user
 - Responses:**
 - 200 OK
 - 404 User not found
- Footer:** API docs by Redocly

Right Screenshot (Dark Theme):

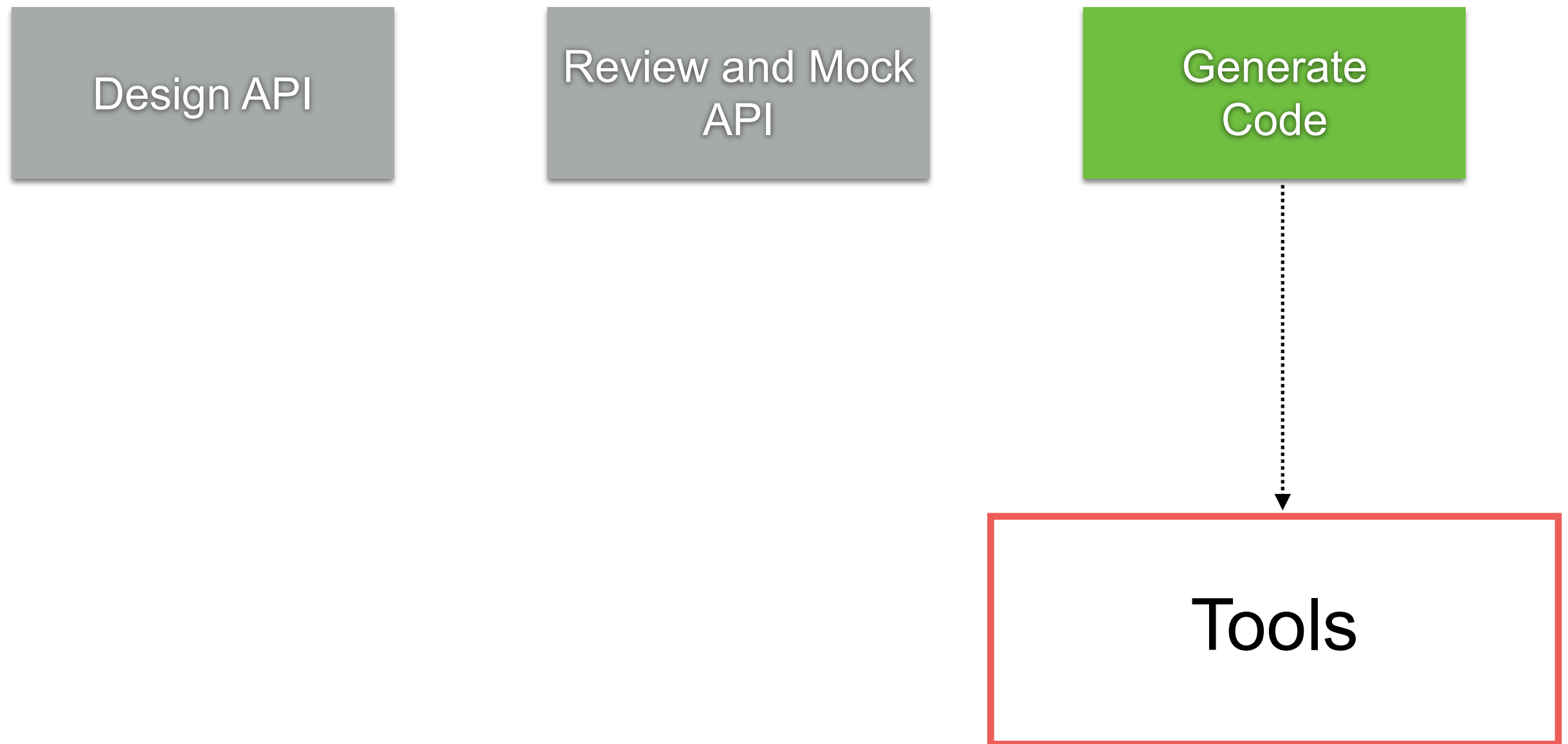
- Endpoints List:**
 - GET /ping
 - GET /users/{id}
- /users/{id} Endpoint:**
 - Response samples:**
 - 200 (selected)
 - 404
 - Content type:** application/json
 - Copy** button
 - JSON Response:**

```
{  "id": 1,  "name": "User 01"}
```

<https://github.com/Redocly/redoc#redoc-cli>



Generate Code



Generate Code from Swagger

Server-side
Client-side



<https://github.com/swagger-api/swagger-codegen>



Working with Postman API-first



<https://www.postman.com/api-first/>



Step 1 :: Design API

Design API

Review and Mock
API

Generate code
API Document

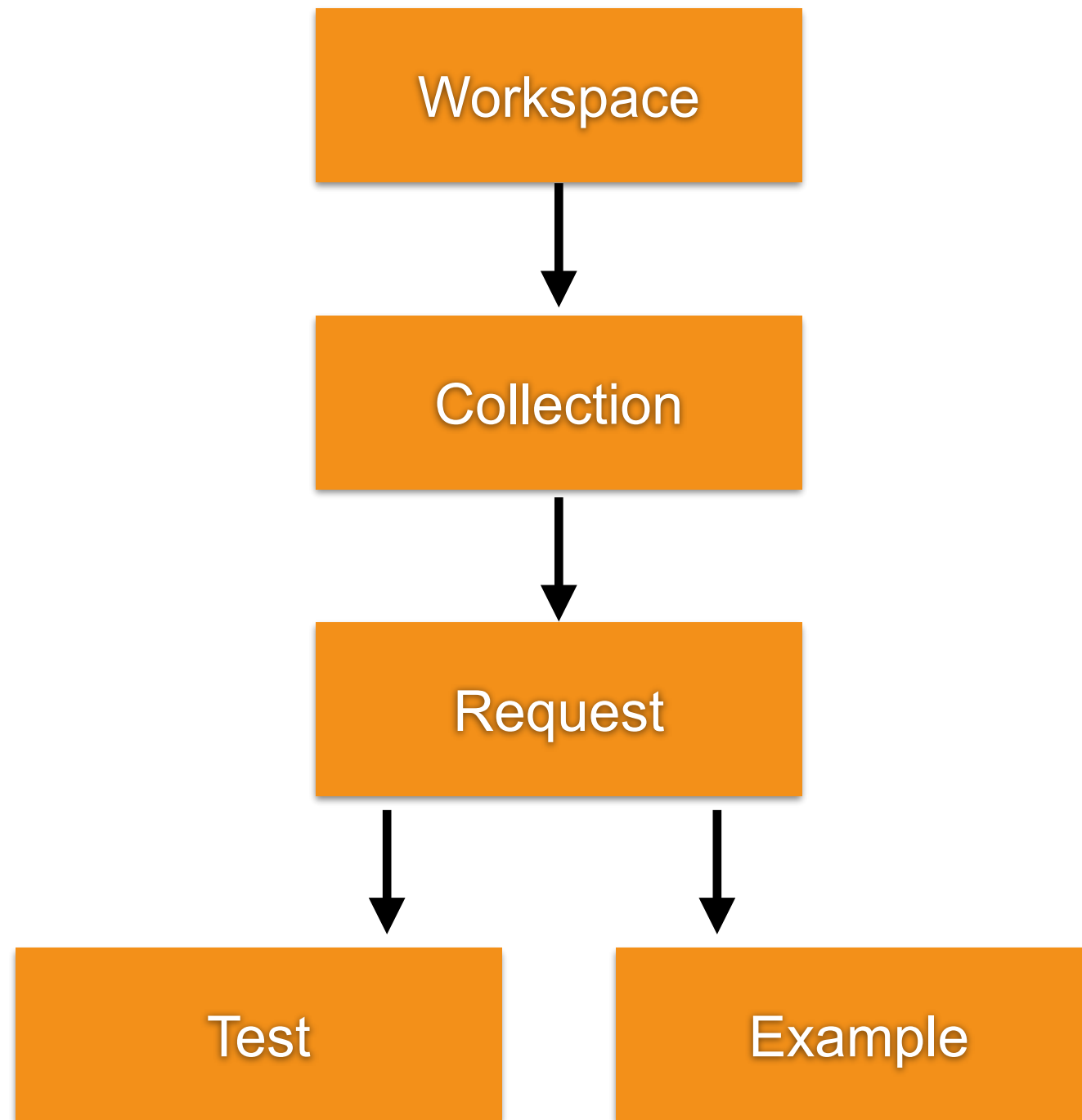


Postman

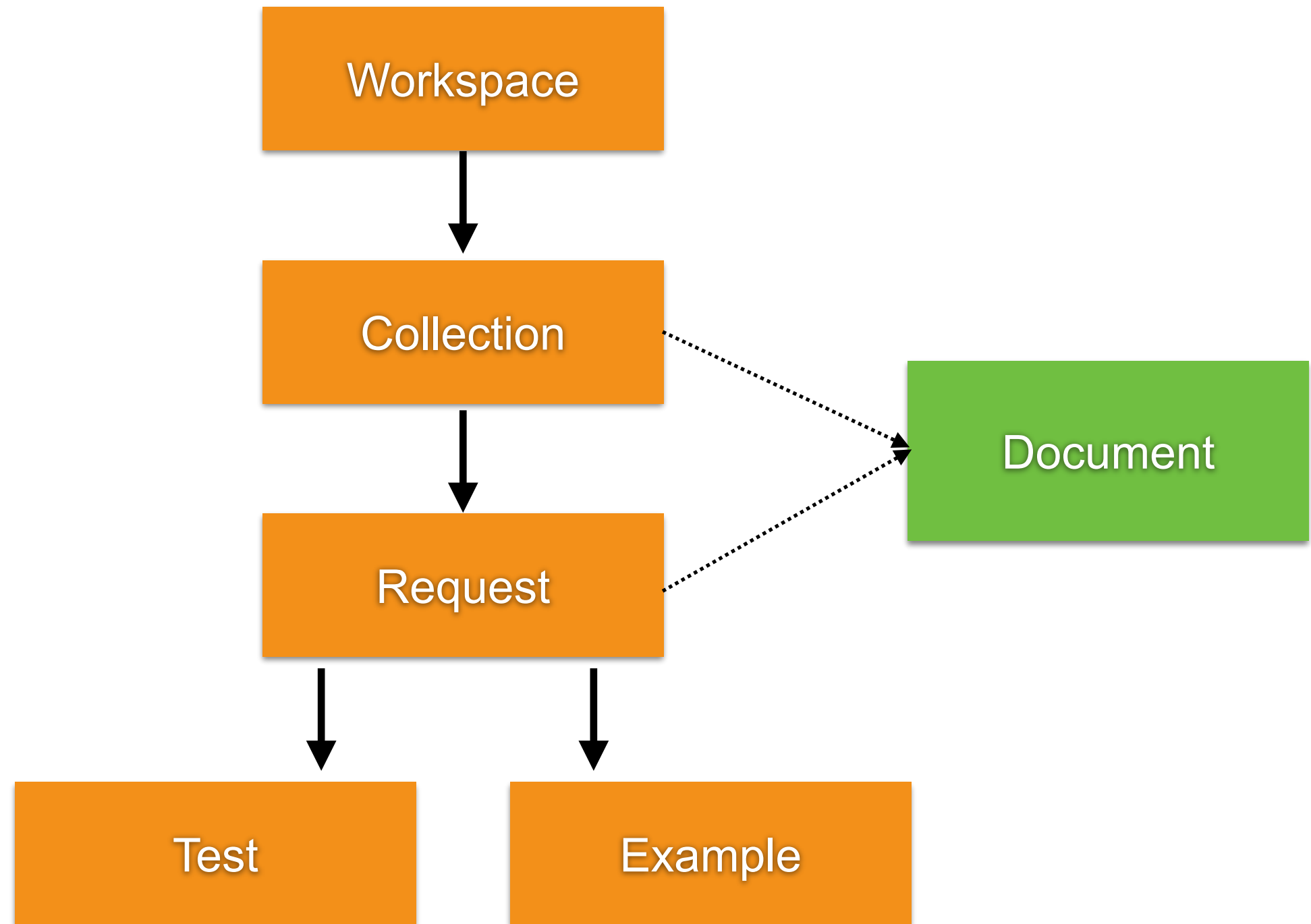
<https://www.postman.com/>



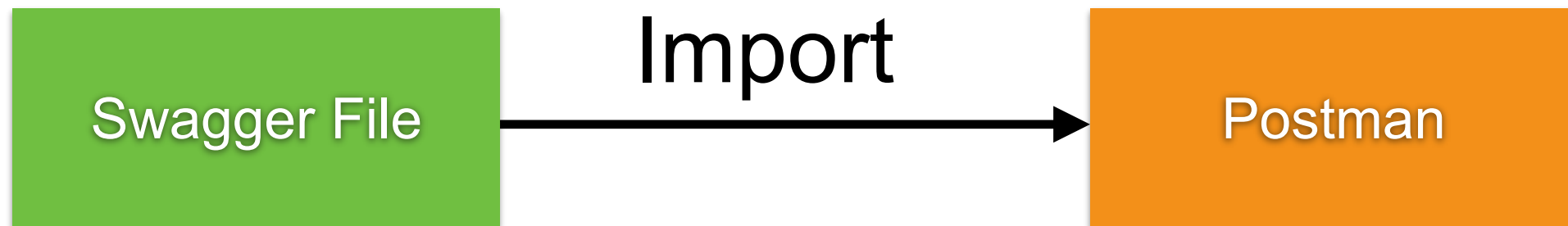
API-first in Postman



API-first in Postman




Import from Swagger/OpenAPI






Import with OpenAPI file

×



Drop anywhere to import
Or select [files](#) or [folders](#)



Other Sources ▼

[Learn more about importing data](#) ↗



Example for Design your APIs

The screenshot displays an API design tool interface. On the left, a sidebar shows a project structure under a 'Demo' folder, including 'users' and 'ping' endpoints. The 'users' folder is expanded, showing a 'GET /users/:id' endpoint. The right pane provides a detailed view of this endpoint, including its HTTP method, URL, and configuration tabs.

Endpoint Details:

- Method:** GET
- URL:** `{{baseUrl}}/users/:id`
- Params:** (Active tab)
- Authorization:**
- Headers (7):**
- Body:**
- Pre-request Script:**
- Tests:**
- Settings:**

Query Params:

Key	Value
Key	Value

Path Variables:

Key	Value
id	<integer>



Postman Collection to OpenAPI

Transform Postman Collections into OpenAPI

postman2openapi

Postman Collection JSON:

```
1 {
2   "info": {
3     "_postman_id": "b49384f5-dce6-49b9-8d65-b805deb3eb67",
4     "name": "Postman Echo",
5     "description": "Postman Echo is service you can use to test your REST clients and make",
6     "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
7   },
8   "item": [
9     {
10      "name": "Request Methods",
11      "item": [
12        {
13          "name": "GET Request",
14          "event": [
15            {
16              "listen": "test",
17              "script": {
18                "id": "13f30c4a-447c-4bba-930f-a023343830fa",
19                "exec": [
20                  "pm.test(\"response is ok\", function () {",
21                    "    pm.response.to.have.status(200);",
22                  "});",
23                  "pm.test(\"response body has json with request queries\", function () {",
24                    "    pm.response.to.have.jsonBody('args.foo1', 'bar1');",
25                    "    pm.response.to.have.jsonBody('args.foo2', 'bar2');",
26                  "});",
27                ],
28                "type": "text/javascript"
29              }
30            }
31          ],
32          "request": {
33            "method": "GET",
34            "header": [],
35            "url": {
36              "raw": "https://postman-echo.com/get?foo1=bar1&foo2=bar2",
37              "protocol": "https",
38              "host": [
39                "postman-echo",
40                "com"
41              ],
42            }
43          }
44        }
45      ]
46    }
47  ]
48 }
```

OpenAPI:

```
openapi: 3.0.3
info:
  title: Postman Echo
  description: >-
    Postman Echo is service you can use to test your REST clients and make
    sample API calls. It provides endpoints for `GET`, `POST`, `PUT`, various
    auth mechanisms and other utility endpoints.

    The documentation for the endpoints as well as example responses can be
    found at
    [https://postman-echo.com](https://postman-echo.com/?source=echo-collection-app-onboarding)
    version: 1.0.0
    contact: {}
  servers:
    - url: https://postman-echo.com
  paths:
    /get:
      get:
        tags:
          - Request Methods
        summary: GET Request
        description: >-
          The HTTP `GET` request method is meant to retrieve data from a server.
          The data
          is identified by a unique URI (Uniform Resource Identifier).

          A `GET` request can pass parameters to the server using "Query String
          Parameters". For example, in the following request,

          > http://example.com/hi/there?hand=wave

          The parameter "hand" has the value "wave".

          This endpoint echoes the HTTP headers, request parameters and the
          complete
```

<https://kevinswiber.github.io/postman2openapi/>



API Testing with Postman and newman






Postman

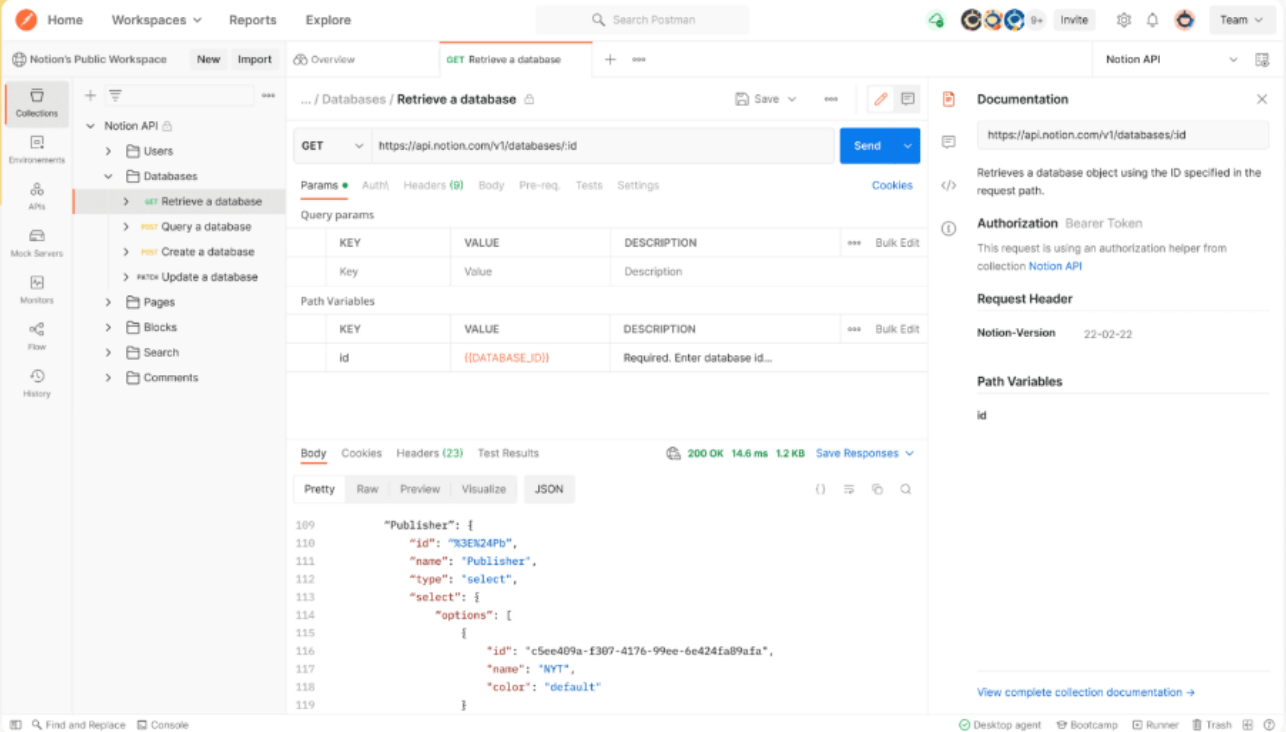
**Publi _
APIs together**

Over 25 million developers use Postman. Get started by signing up or downloading the desktop app.

[Sign Up for Free](#)


Download the desktop app for



The screenshot displays the Postman web interface. On the left, a sidebar shows a collection of API requests under 'Notion API'. The main panel shows a 'GET' request to 'https://api.notion.com/v1/databases/id'. The 'Params' tab is active, showing a query parameter 'id' with the value '[[DATABASE_ID]]'. The 'Body' tab shows a successful response with a status of '200 OK' and a JSON body containing publisher information. On the right, a 'Documentation' panel provides details about the endpoint and its authorization requirements.

What is Postman?



<https://www.postman.com/>

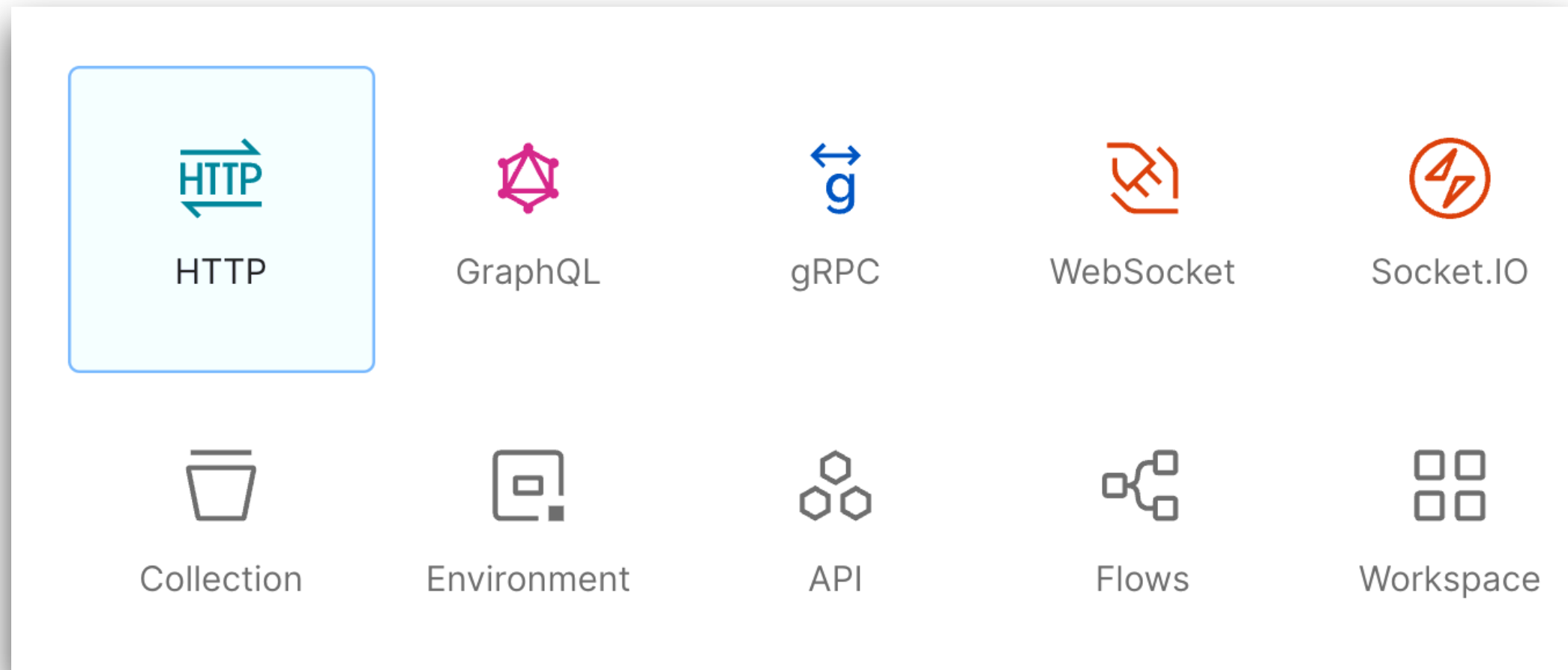


Topics with Postman

Collections
HTTP Request
Testing in HTTP Request
Import/Export collection
Mock Server



Testing with Postman



Topics with Postman

The screenshot displays the Postman application interface. On the left sidebar, the 'demo-nodejs' workspace is selected, showing a collection 'group01' with a single item 'GET Hello API'. The main panel shows the details of this request. The method is 'GET' and the URL is 'http://localhost:3000/'. The 'Tests' tab is active, containing the following JavaScript code:

```
1 pm.test("Status code is 200", function () {
2   pm.response.to.have.status(200);
3 });
4
5 pm.test("Check hello message", function () {
6   var jsonData = pm.response.json();
7   pm.expect(jsonData.message).to.eql('Hello World');
8 });
9
10 var schema = {
11   "$schema": "http://json-schema.org/draft-04/schema#",
12   "type": "object",
13   "properties": {
14     "message": {
15       "type": "string"
```

Below the code, the 'Body' tab shows the response in 'Pretty' format:

```
1 {
2   "message": "Hello World"
3 }
```

The status bar at the bottom indicates a successful response: '200 OK', '18 ms', and '260 B'. The 'Test Results' tab shows '3/3' tests passed.



Testing in HTTP Request

HTTP Response code
HTTP Response Body
Validate JSON Schema

<https://learning.postman.com/docs/writing-scripts/test-scripts/>



Testing in HTTP Request

The screenshot shows the Postman interface for a GET request to `http://localhost:3000/`. The **Tests** tab is selected and highlighted with a red box. The test scripts are written in JavaScript:

```
1 pm.test("Status code is 200", function () {  
2   pm.response.to.have.status(200);  
3 });  
4  
5 pm.test("Check hello message", function () {  
6   var jsonData = pm.response.json();  
7   pm.expect(jsonData.message).to.eql('Hello World');  
8 });  
9  
10 var schema = {  
11   "$schema": "http://json-schema.org/draft-04/schema#",  
12   "type": "object",  
13   "properties": {  
14     "message": {  
15       "type": "string"
```

On the right side, there is a **Cookies** tab and a section for **Test scripts** with a description: "Test scripts are written in JavaScript, and are run after the response is received. Learn more about tests scripts". Below this is a **Snippets** section with several predefined test snippets:

- Response body: Contains string
- Response body: JSON value check
- Response body: Is equal to a string
- Response headers: Content-Type header check
- Response time is less than 200ms

<https://learning.postman.com/docs/writing-scripts/test-scripts/>



Newman

```
$npm i -g newman  
$newman run <json file>
```

<https://www.npmjs.com/package/newman>



Newman and report

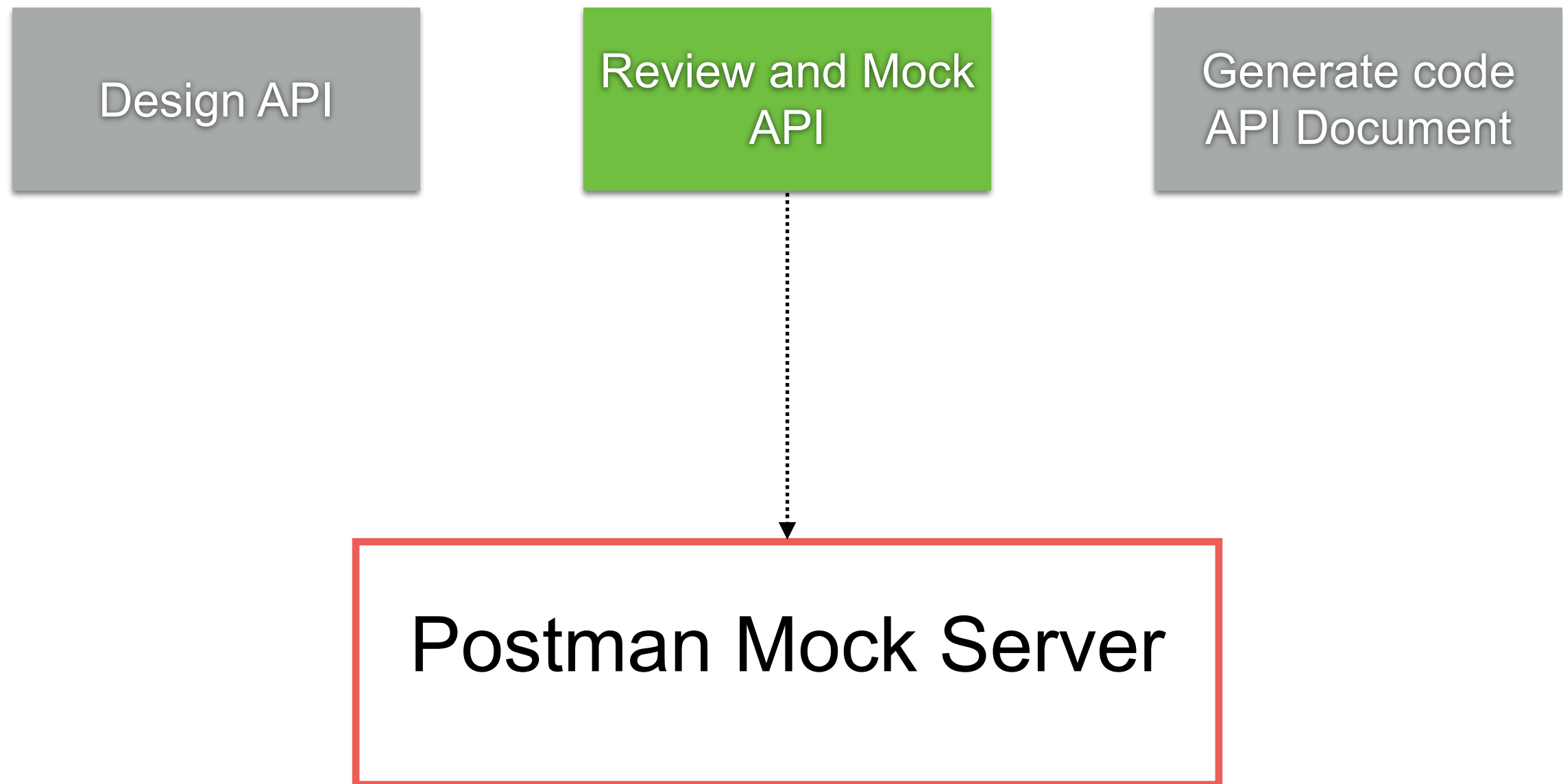
```
$npm i -g newman
```

```
$newman run <json file> -r cli,junit
```

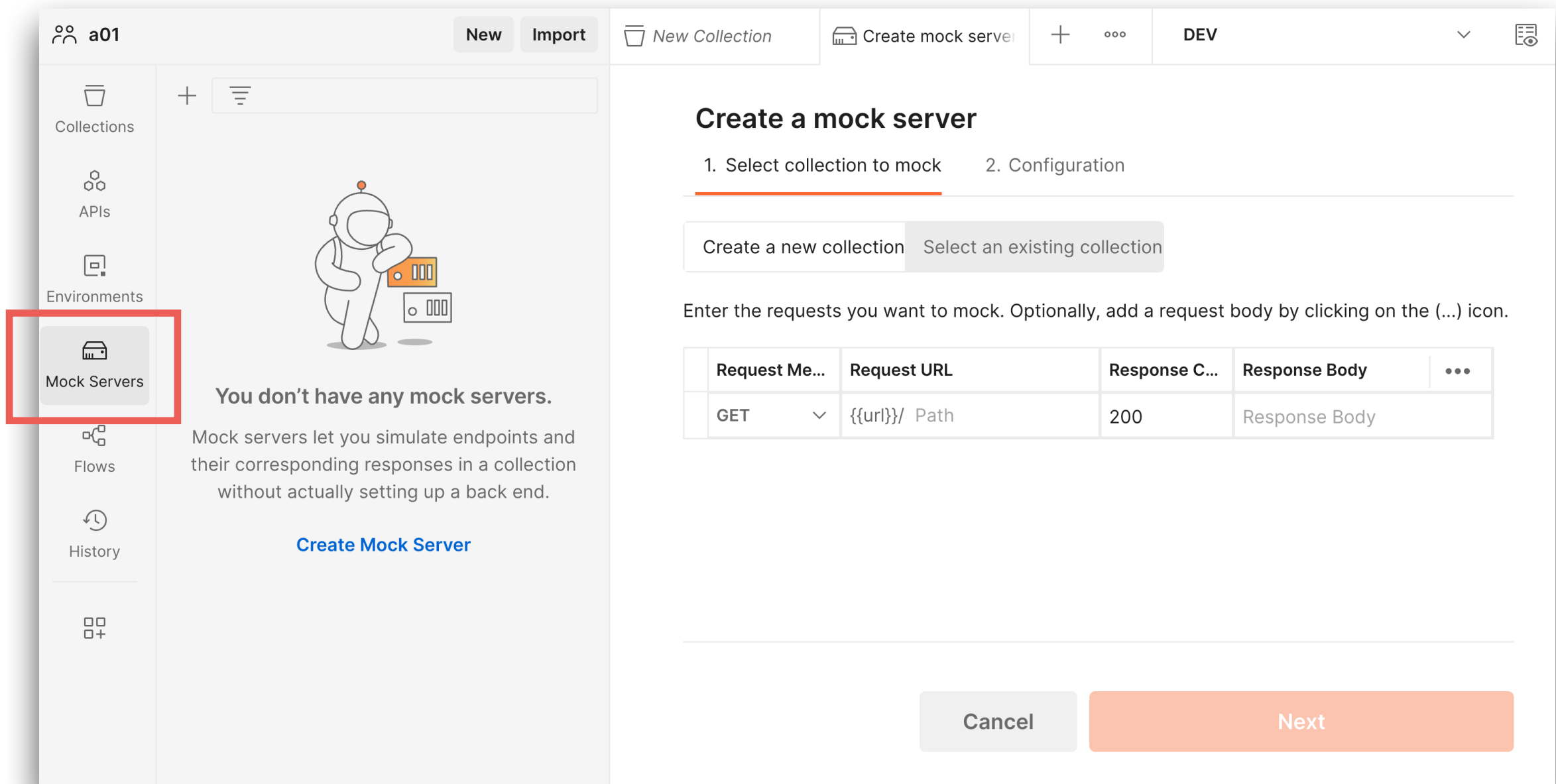
<https://www.npmjs.com/package/newman>



Step 2 :: Mock API



Create Mock Server from Collection



Create a mock server

1. Select collection to mock 2. Configuration

Create a new collection Select an existing collection

Enter the requests you want to mock. Optionally, add a request body by clicking on the (...) icon.

Request Me...	Request URL	Response C...	Response Body	...
GET ▾	{{url}}/ Path	200	Response Body	

Cancel Next

<https://learning.postman.com/docs/designing-and-developing-your-api/mocking-data/setting-up-mock/>



Step 3 :: Generate !!

Design API

Review and Mock
API

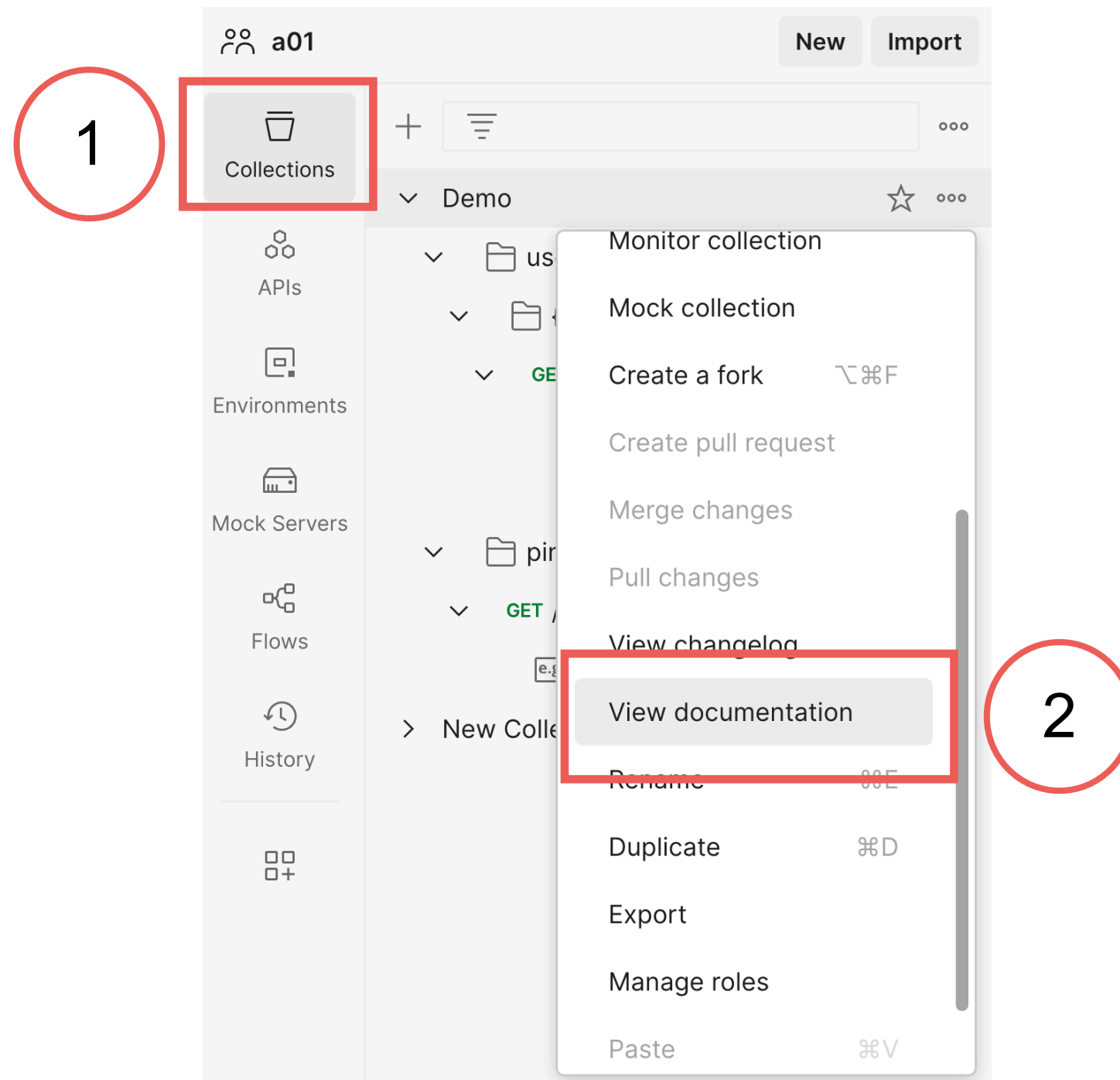
Generate code
API Document



Postman !!



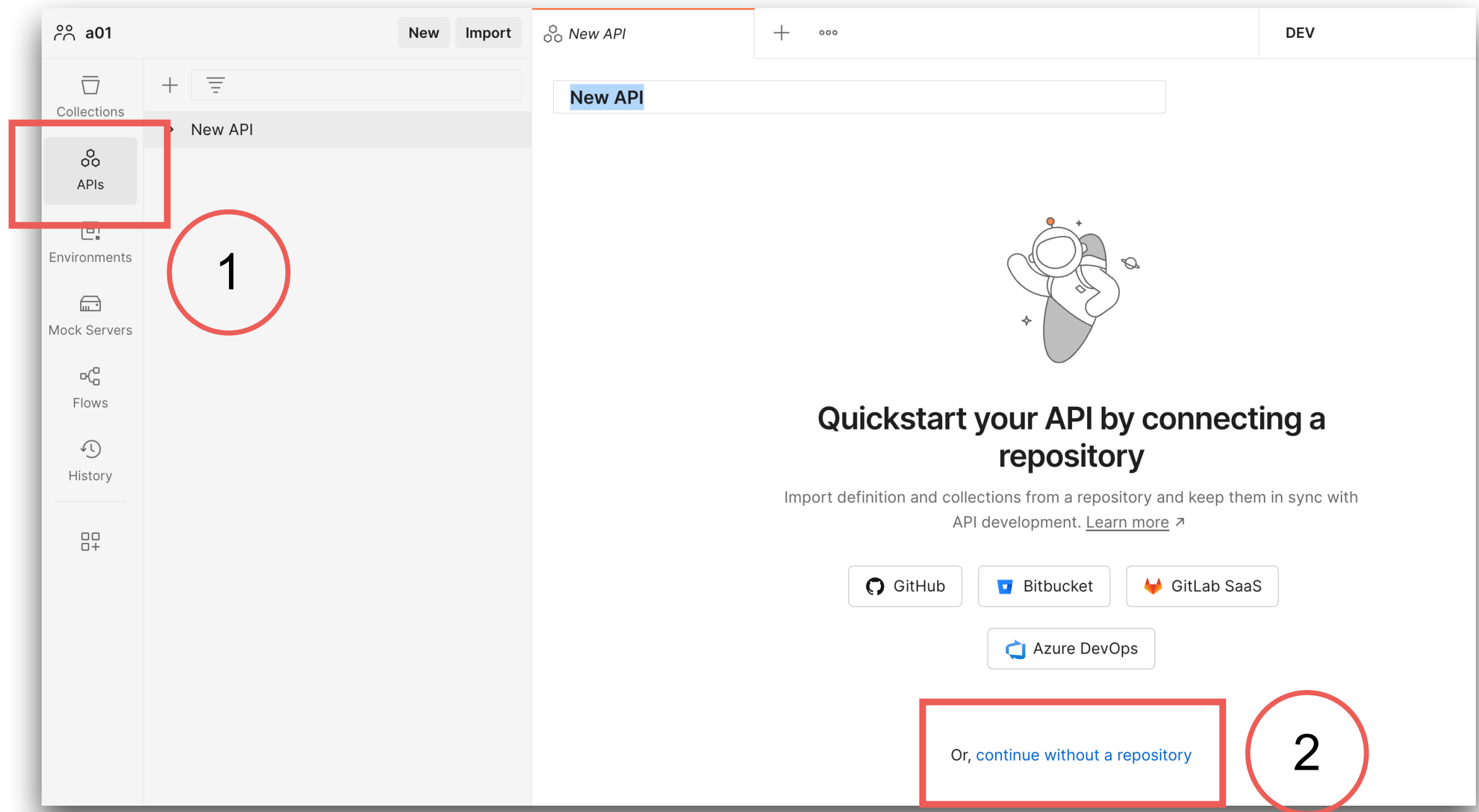
Generate API Document



<https://learning.postman.com/docs/publishing-your-api/documenting-your-api/>



Generate Code with APIs



<https://learning.postman.com/docs/designing-and-developing-your-api/the-api-workflow/>



Create APIs from Collections

The screenshot displays the Postman 'New API' interface. On the left sidebar, the 'Collections' section is expanded, showing a tree view with 'New API' as the root. Under 'New API', there are four collections: 'Demo', 'users', '{id}', and 'ping'. The 'Demo' collection is highlighted. The main panel shows the 'New API' configuration. It includes sections for 'About this API' (with 'Add summary...' and 'Add description...' fields), 'Connect repository' (with a 'Connect' button), 'Collections' (with a '+' button and a list containing 'Demo'), and 'Definition' (with a '+' button and a description field). A red circle with the number '3' is placed over the 'Collections' section. The right sidebar contains links for 'Test and Automation', 'API Performance', 'Deployments', 'Publish your API to consumers', and 'Editors'.

<https://learning.postman.com/docs/designing-and-developing-your-api/developing-an-api/generating-server-code/>



Generate code

The screenshot shows the Postman 'New API' interface. On the left, the 'New API' panel includes a 'Share' button, a view count of '0', and sections for 'About this API' with 'Add summary...' and 'Add description...' fields, and a 'Connect repository' section with a 'Connect' button. On the right, the 'Code Generation' panel (marked 'BETA') is open, showing a code icon, the text 'Generate server boilerplate from your API schema. [Learn more](#)', a 'Language and framework' dropdown menu currently set to 'Select', an unchecked checkbox for 'Only generate routes and in', and a 'Generate Code' button. A dropdown menu is open from the 'Language and framework' dropdown, listing four options: 'Go - Chi server', 'NodeJs - Express', 'Java - JAX-RS', and 'Python - Flask'. The entire interface is under a 'DEV' environment tab.

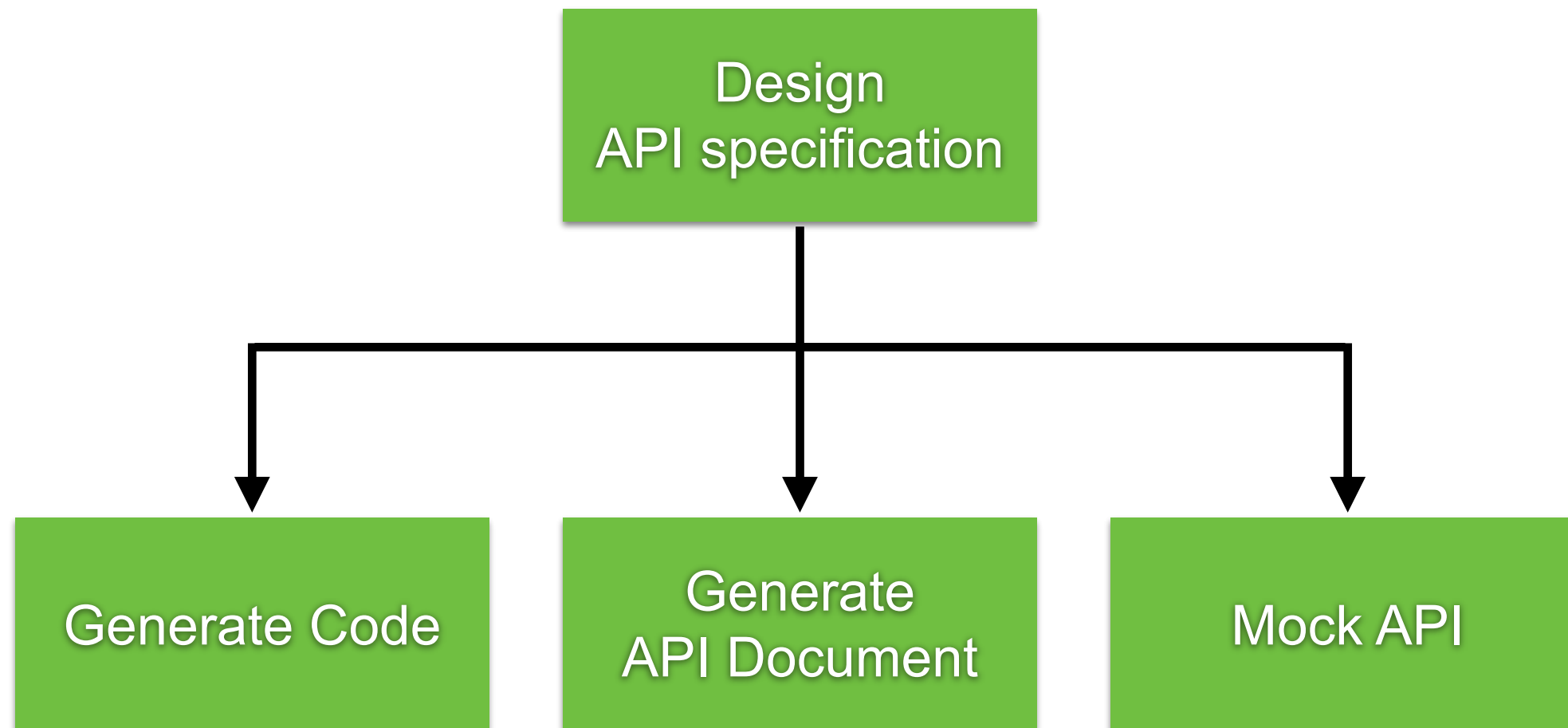
<https://learning.postman.com/docs/designing-and-developing-your-api/developing-an-api/generating-server-code/>



More Solutions ?



Idea !!



API Blueprint



[Docs](#) [Tools](#) [Developers](#) [Support](#)

API Blueprint. A powerful high-level API description language for web APIs.

API Blueprint is simple and accessible to everybody involved in the API lifecycle. Its syntax is concise yet expressive. With API Blueprint you can quickly design and prototype APIs to be created or document and test already deployed mission-critical APIs.

[Tutorial](#)

[Tools section](#)

<https://apiblueprint.org/>



API Blueprint

Design APIs with Simple format !!

```
FORMAT: 1A  
HOST: http://localhost:3000
```

```
# Demo API
```

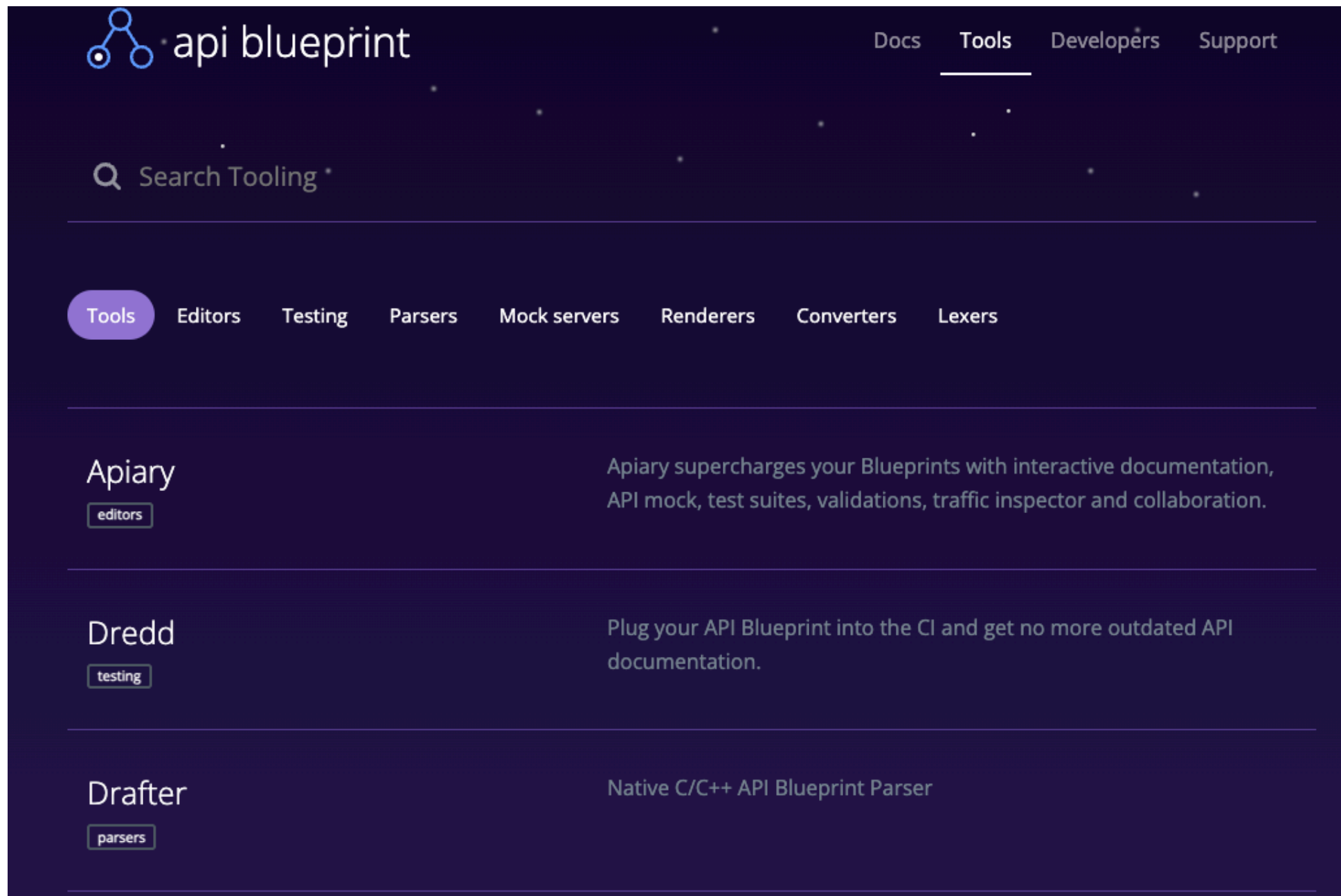
```
Demo APIs for API-first solution !!.
```

```
# Ping service [/ping]  
## Ping [GET]  
+ Response 200
```

<https://apiblueprint.org/tools.html>



API Blueprint Tools !!



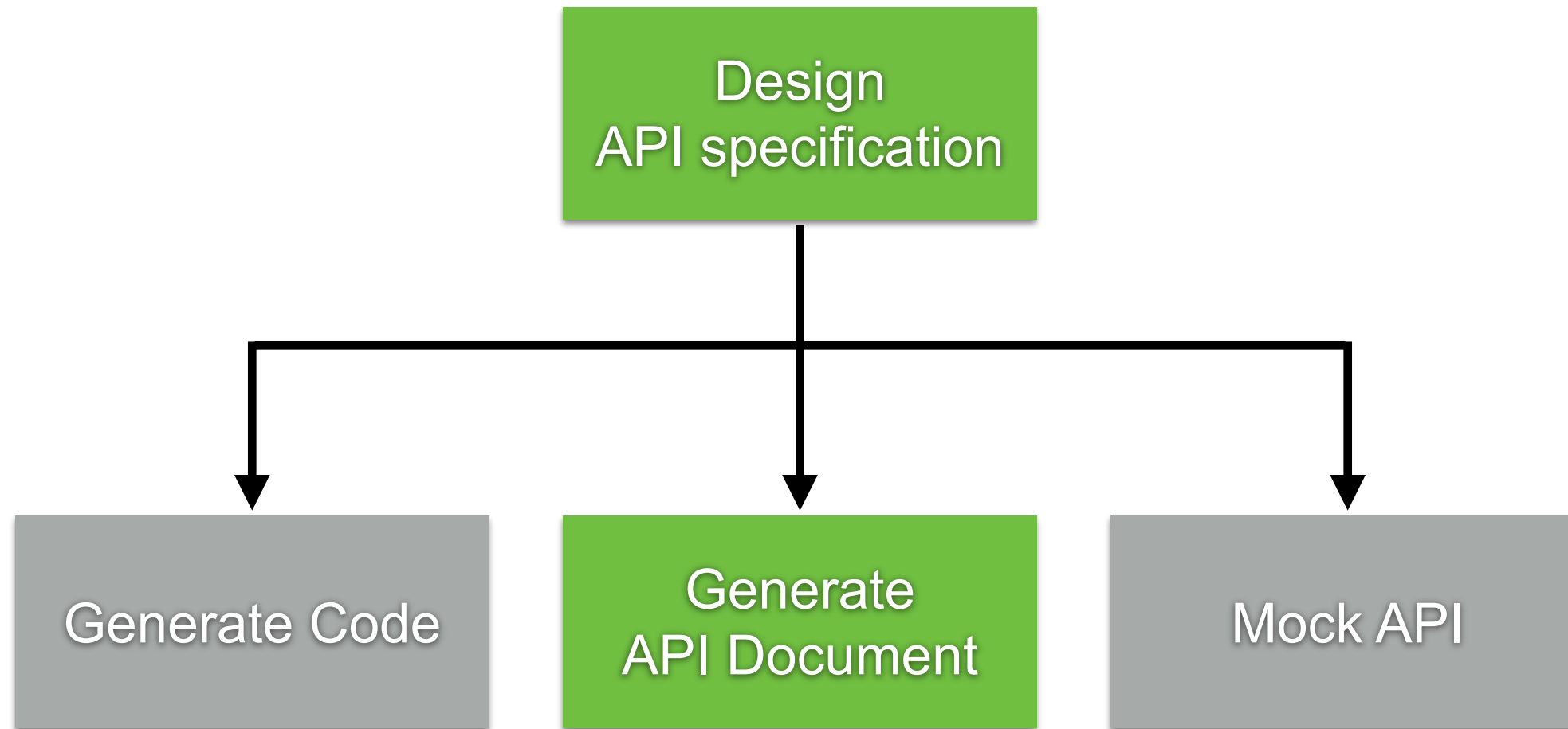
The screenshot shows the 'api blueprint' website with a dark blue background. At the top left is the logo, a blue triangle with a circle inside. To its right is the text 'api blueprint'. In the top right corner are links: 'Docs', 'Tools' (underlined), 'Developers', and 'Support'. Below the header is a search bar with a magnifying glass icon and the text 'Search Tooling'. A horizontal menu below the search bar contains the following items: 'Tools' (highlighted with a purple background), 'Editors', 'Testing', 'Parsers', 'Mock servers', 'Renderers', 'Converters', and 'Lexers'. The main content area lists three tools:

- Apiary** (category: editors): Apiary supercharges your Blueprints with interactive documentation, API mock, test suites, validations, traffic inspector and collaboration.
- Dredd** (category: testing): Plug your API Blueprint into the CI and get no more outdated API documentation.
- Drafter** (category: parsers): Native C/C++ API Blueprint Parser

<https://apiblueprint.org/>



API Blueprint



Generate API Document

```
$npm i -g aglio
```

```
$aglio -i example-api.apib -o output.html
```



<https://www.npmjs.com/package/aglio>



Generate API Document

Resource Group

Ping

Get user by id

<http://localhost:3000>

Demo API

Demo APIs for API-first solution !!.

Resource Group

PING SERVICE

GET

/ping

Ping

Example URI
[GET http://localhost:3000/ping](http://localhost:3000/ping)

Response 200

GET USER BY ID

GET

/users/{id}

Get user by id

Example URI
[GET http://localhost:3000/users/id](http://localhost:3000/users/id)

URI Parameters

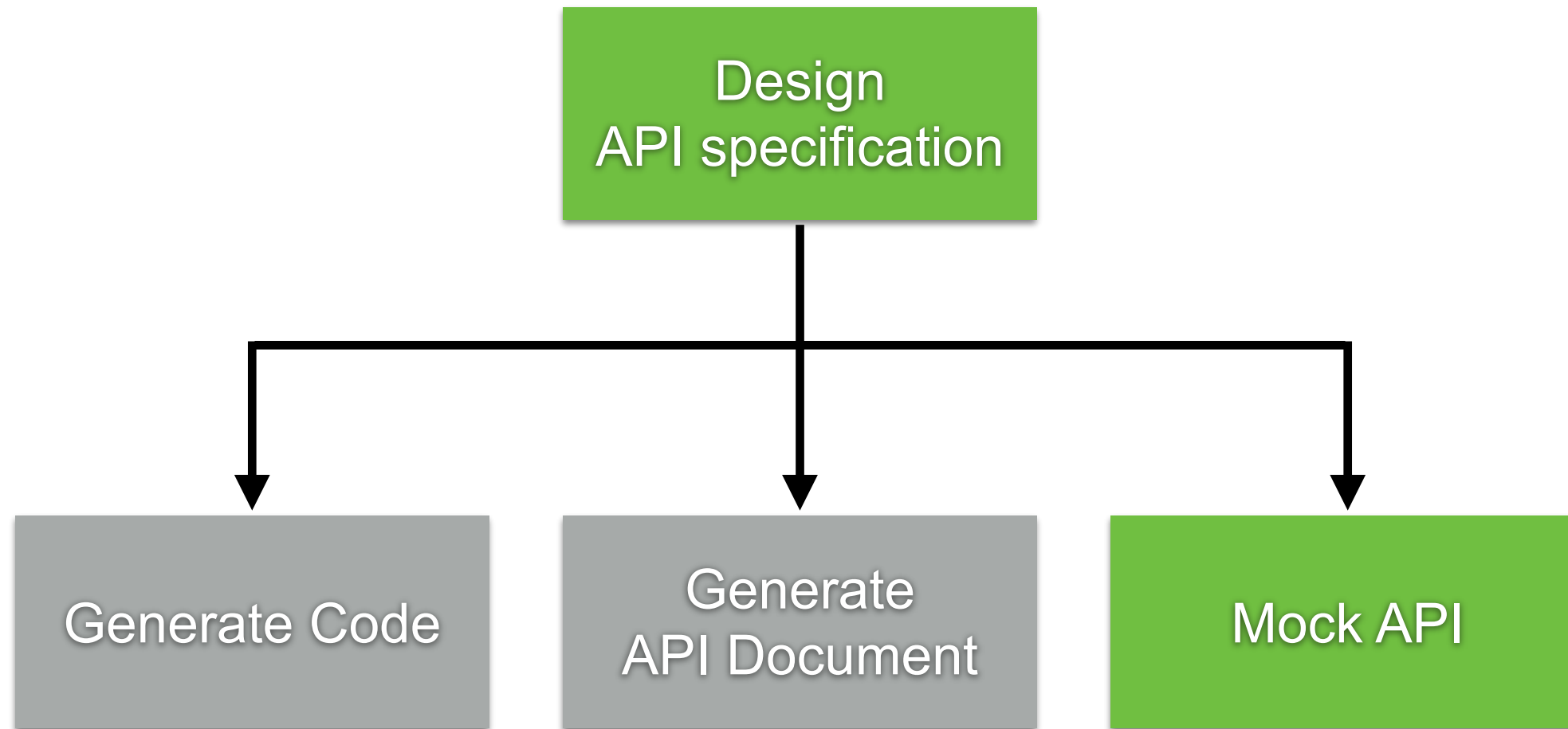
id string (required)

Response 200

<https://www.npmjs.com/package/aglio>



API Blueprint



Create Mock API

```
$npm i -g drakov
```

```
$drakov -f example-api.apib --watch --public -p 3000
```



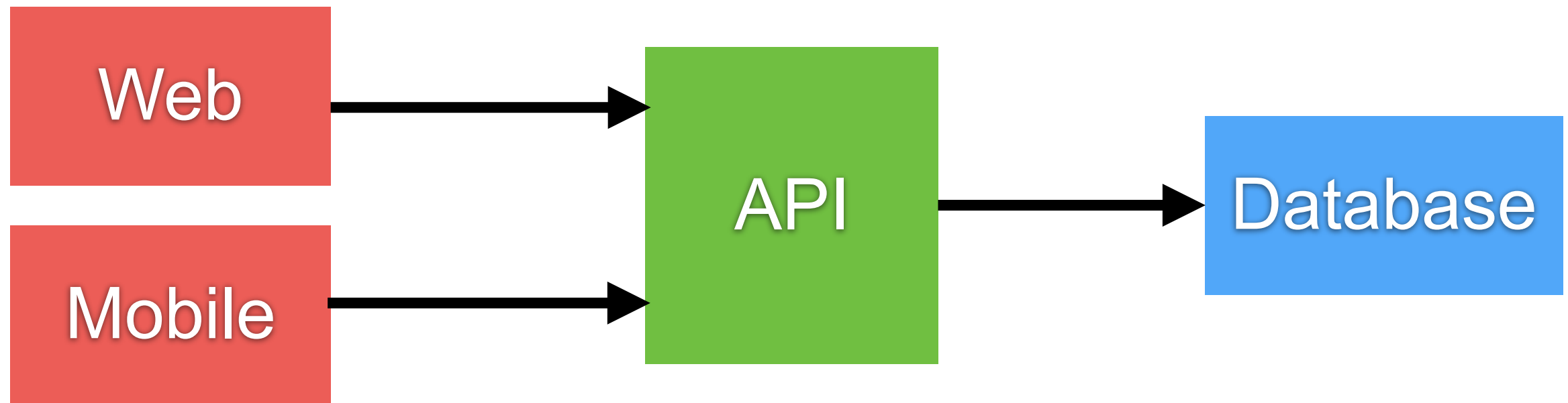
<https://www.npmjs.com/package/drakov>



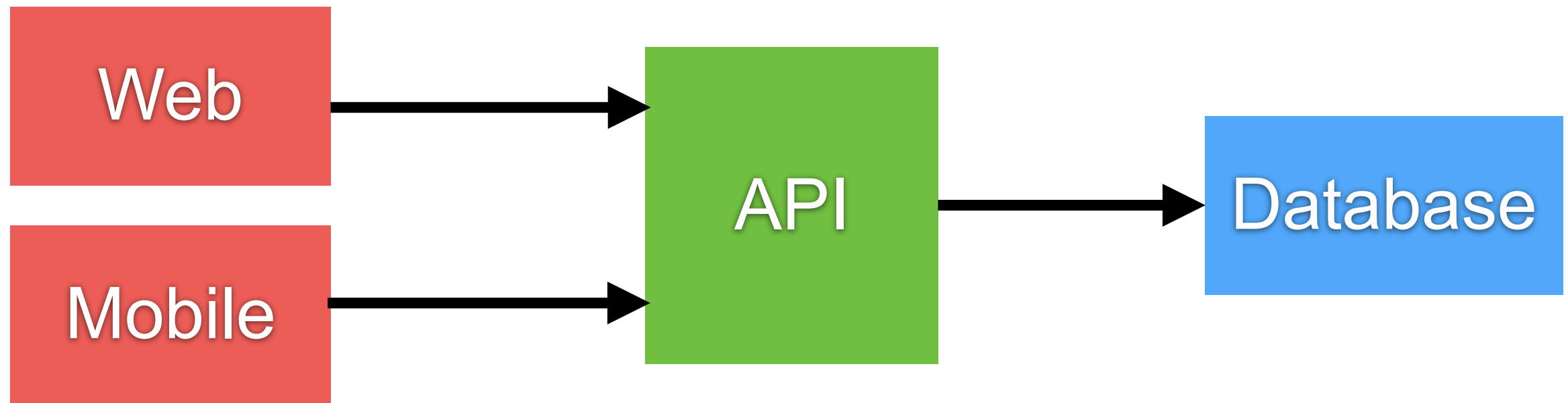
What, Why and How to Test ?



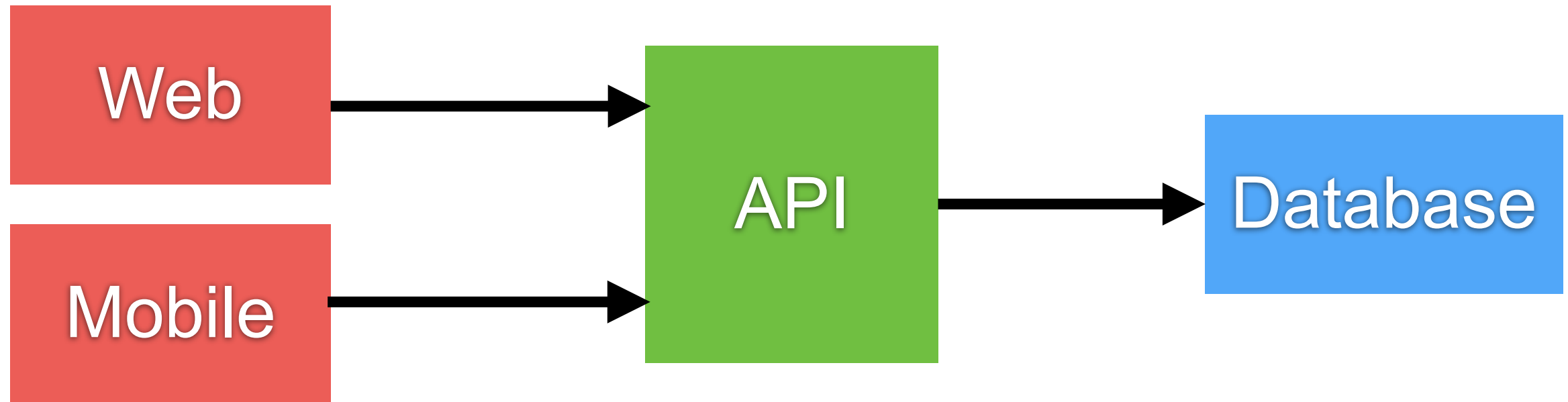
Architecture



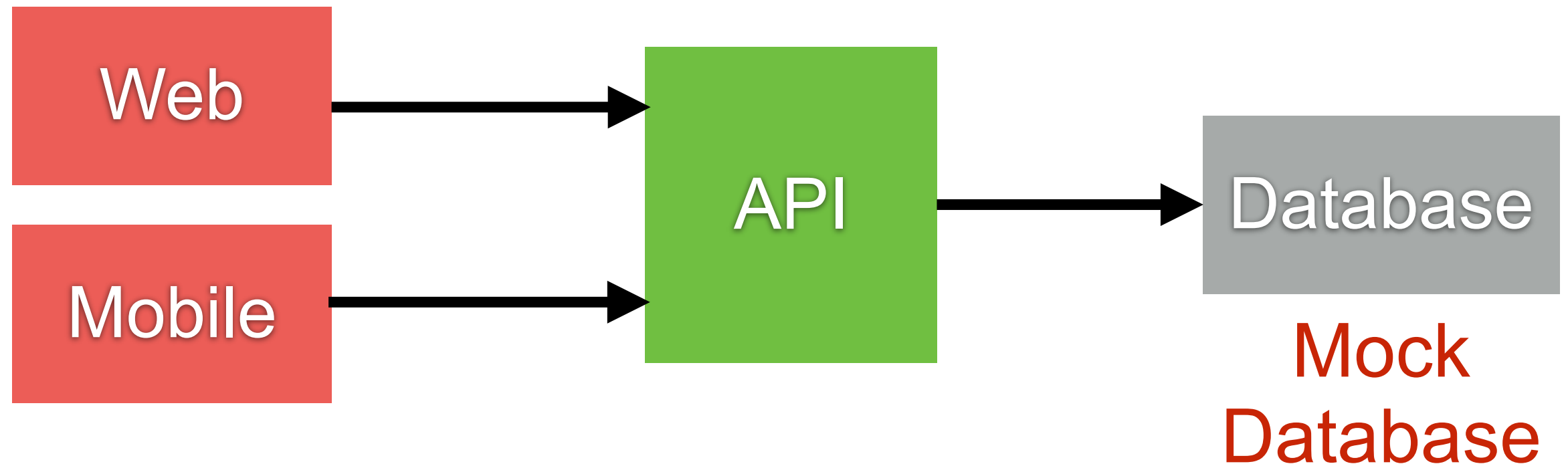
Test ?



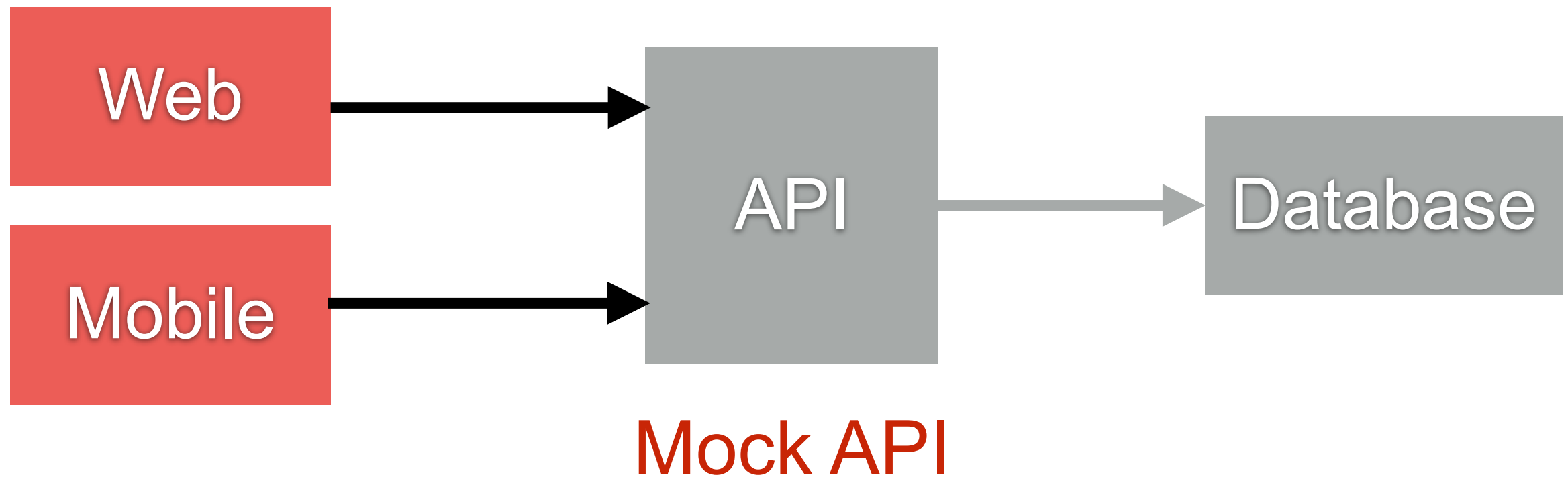
UI Test ?



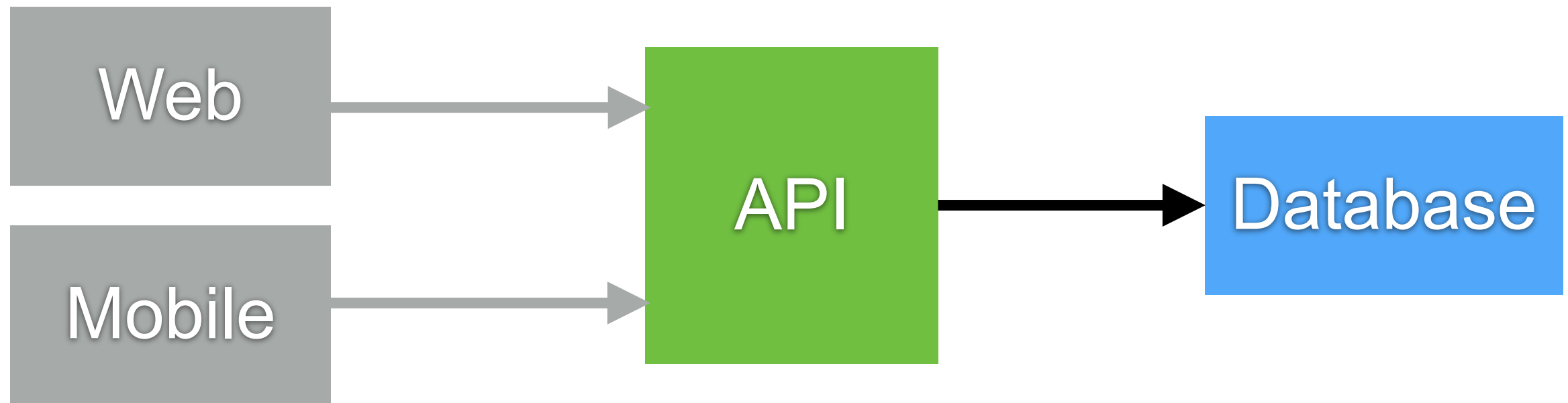
UI Test ?



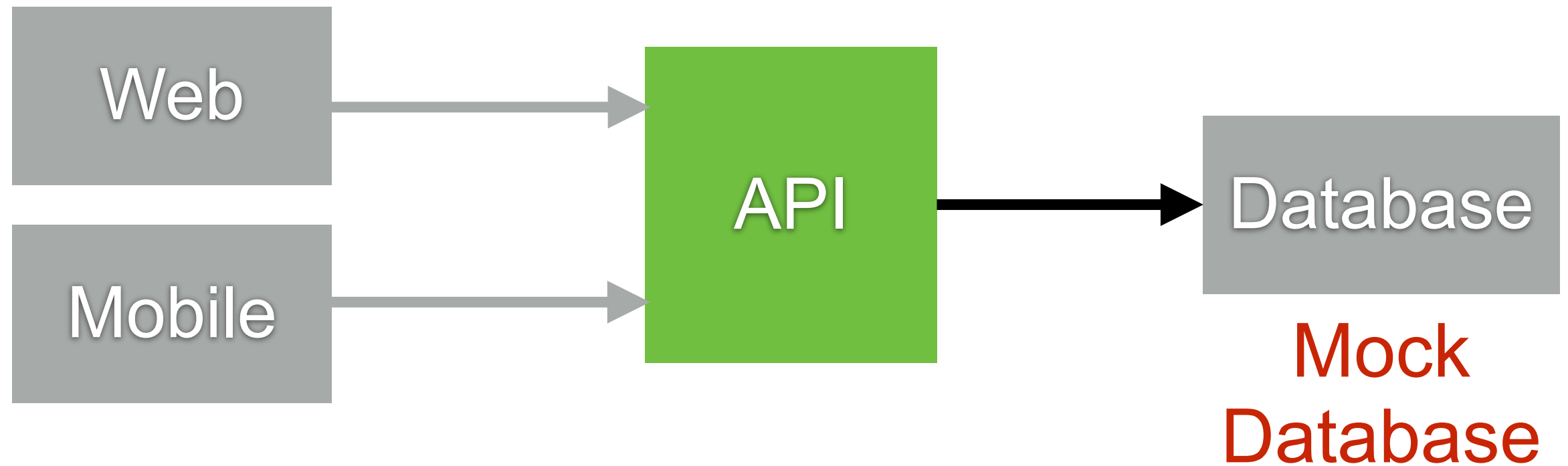
UI Test ?



API Testing ?



API Testing ?



Q/A

