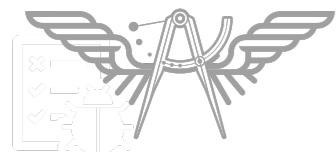


Automated Testing

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

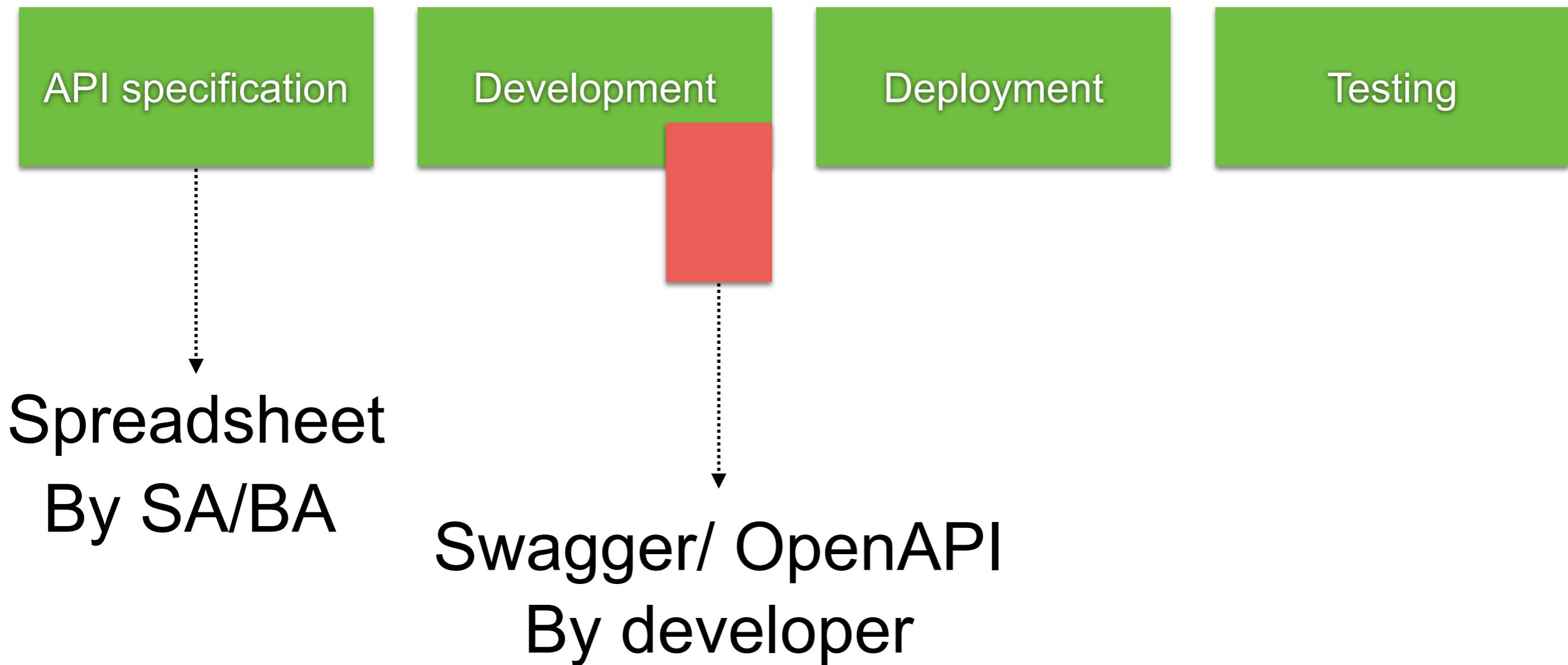


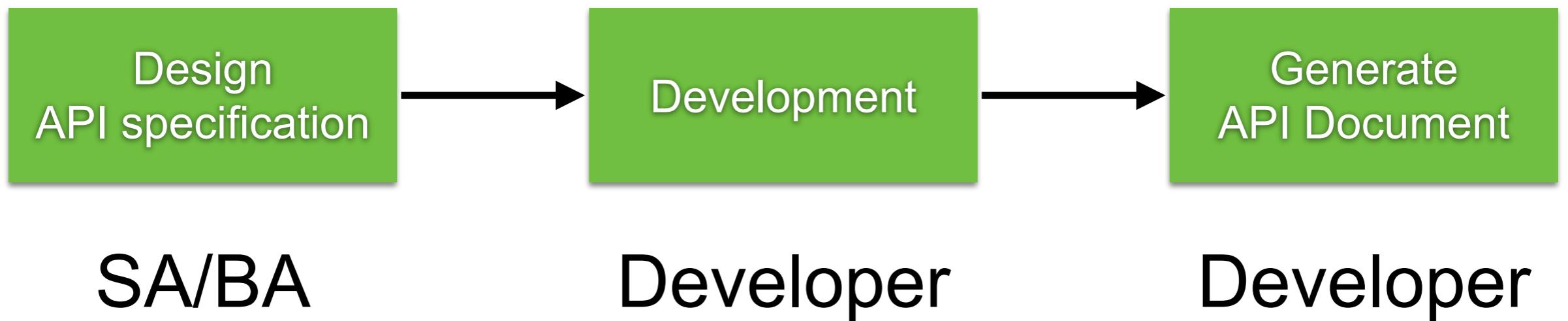
Spreadsheet
By SA/BA



Delivery process

Need API documentation





Spec vs Code Sync ?



Code-first vs API-first development

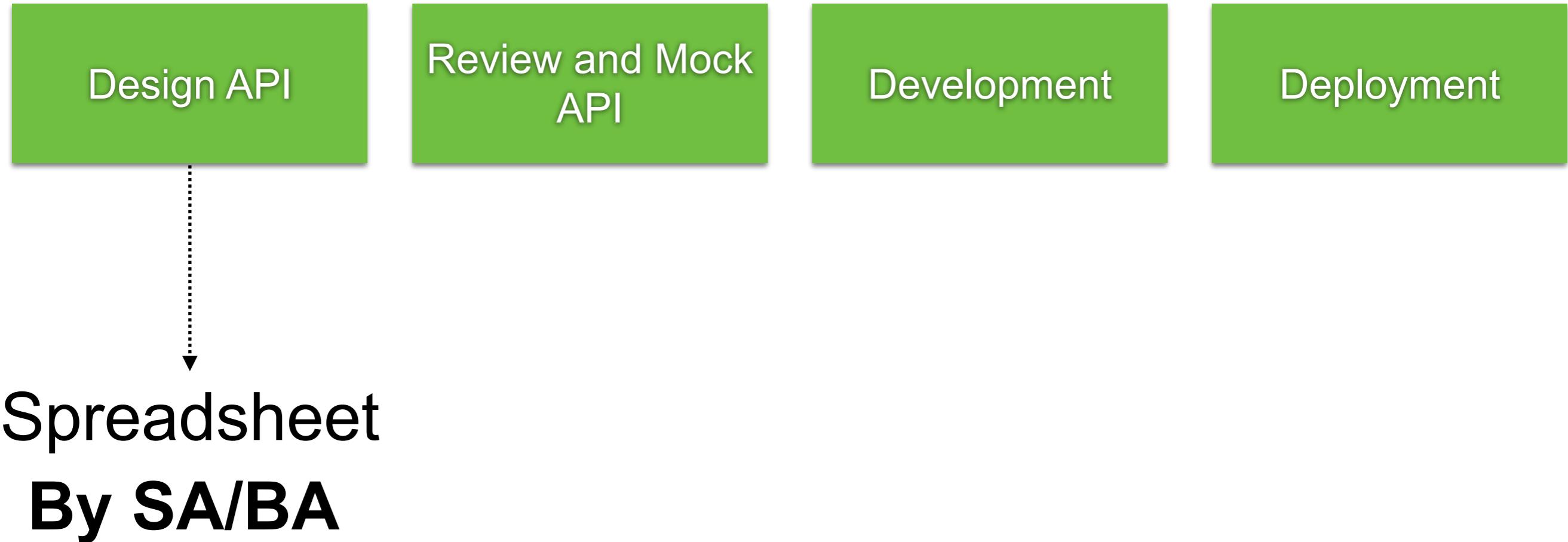
Code
First

VS.

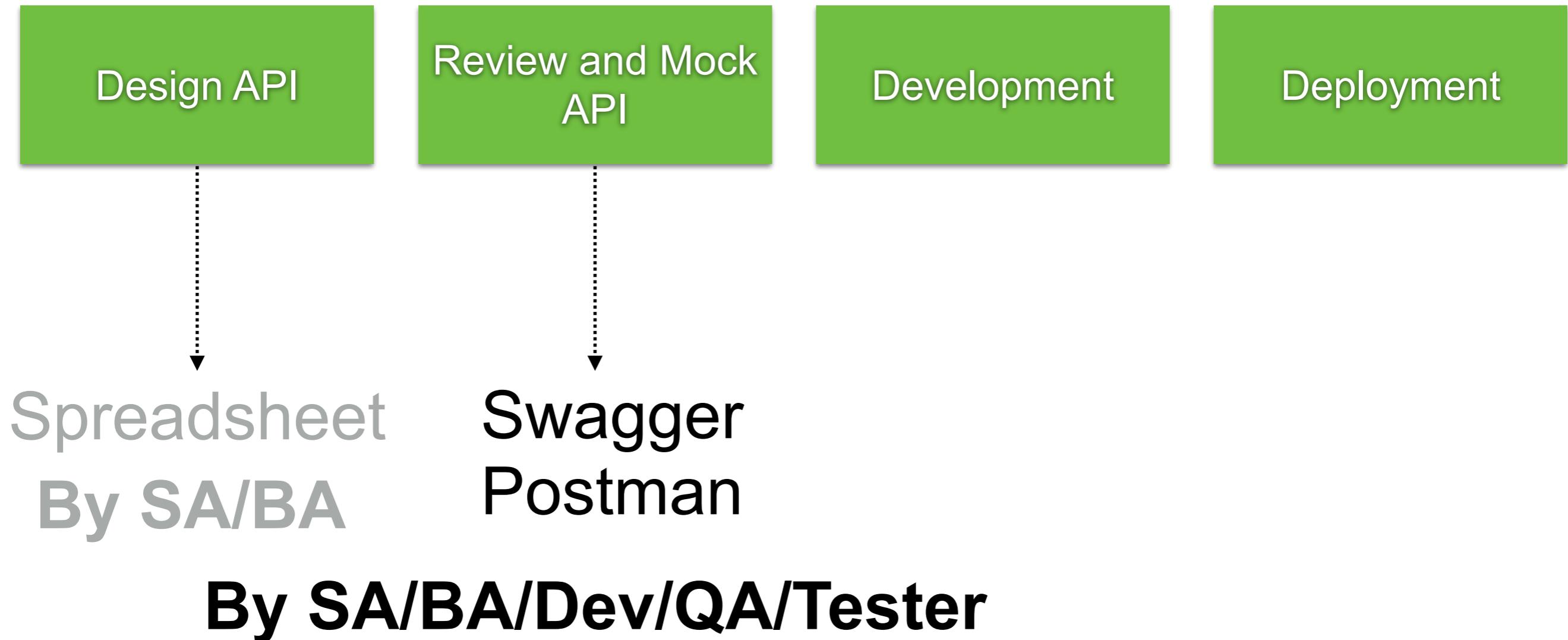
API
First



API-first development



API-first development

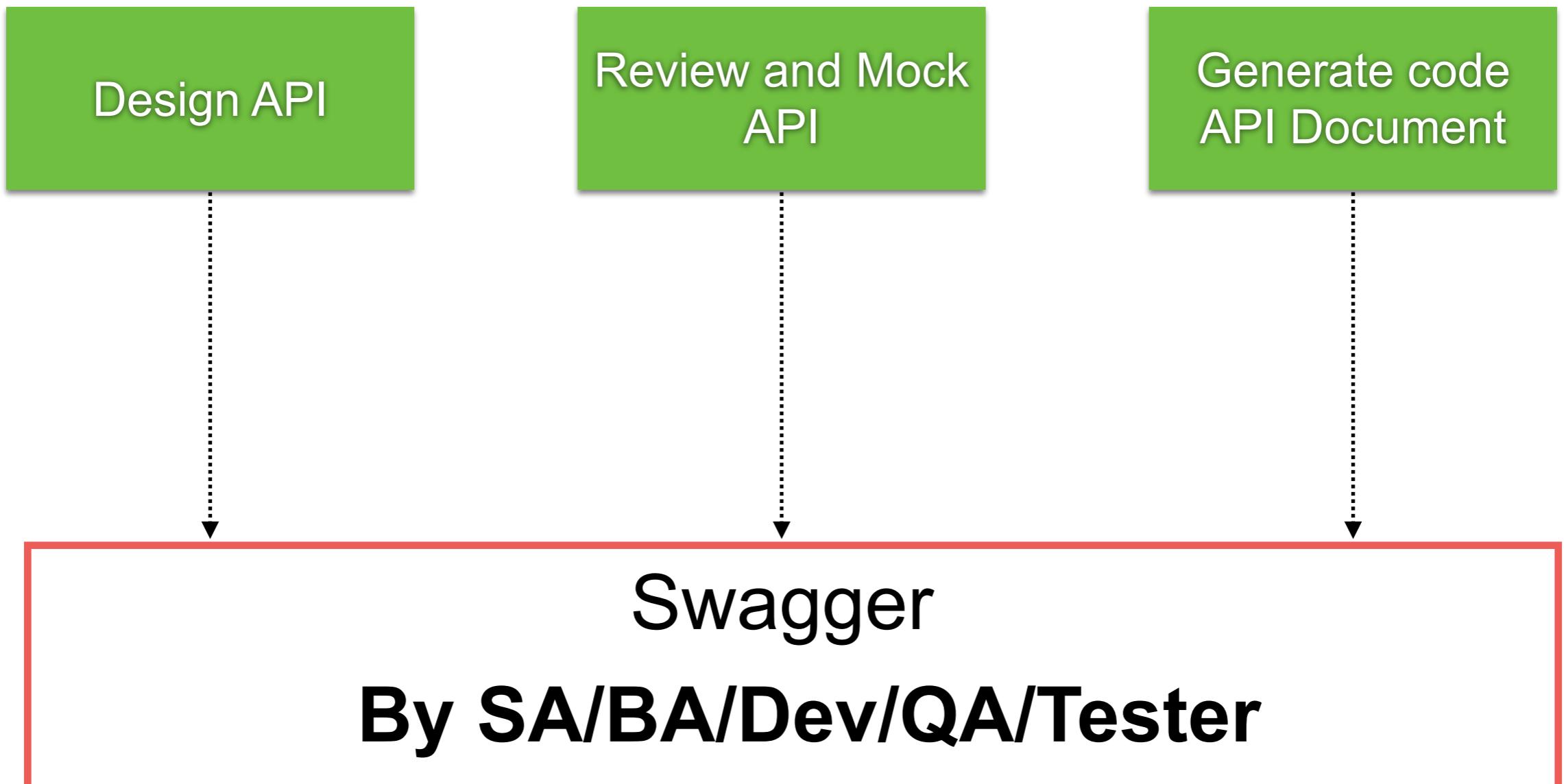


Working with Swagger

<https://swagger.io/>



Process with Swagger



What is OpenAPI ?

What Is OpenAPI?

OpenAPI Specification (formerly Swagger Specification) is an API description format for REST APIs. An OpenAPI specification contains all the information you need to consume 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 latest version of 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, document, and consume REST 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

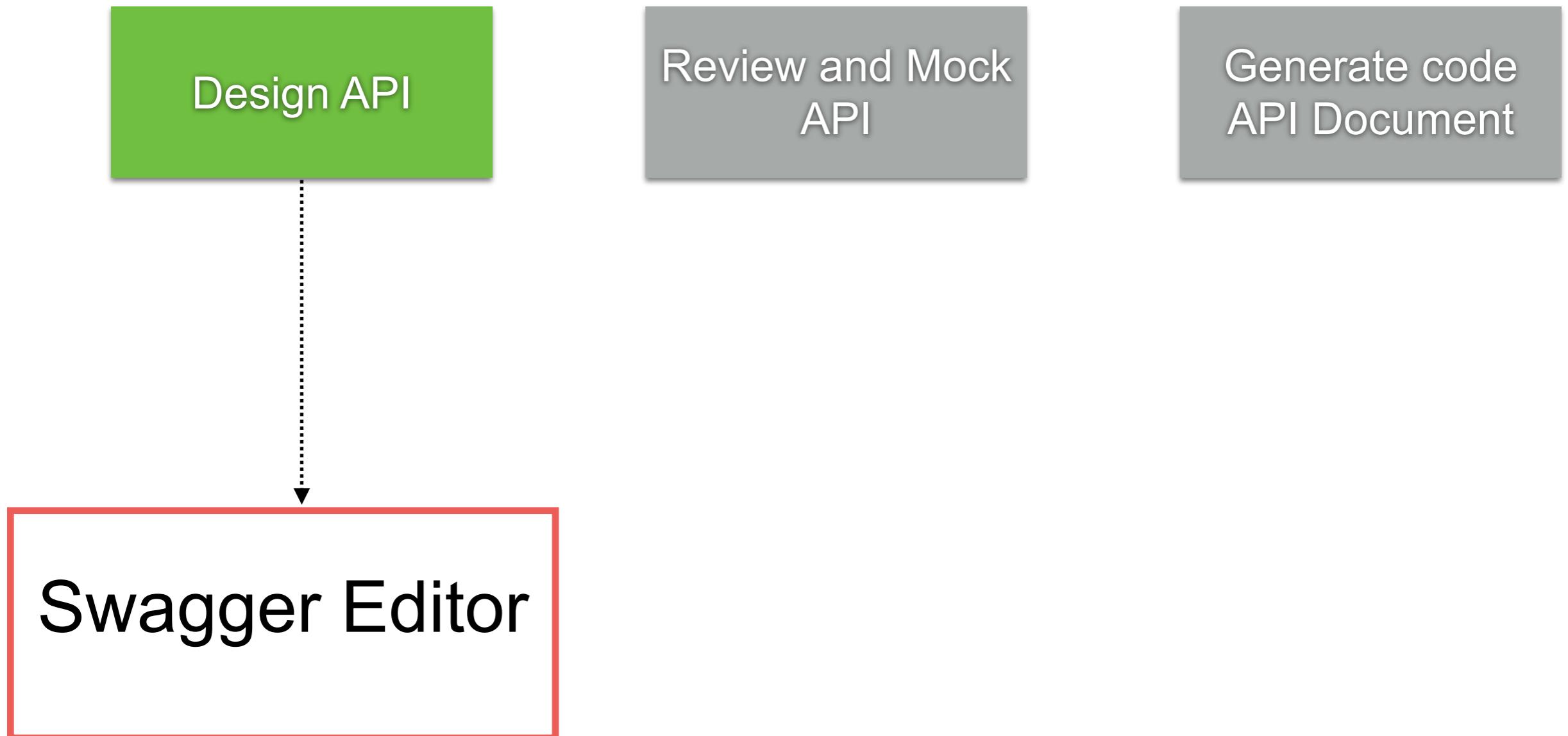
The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to HTTP APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a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



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

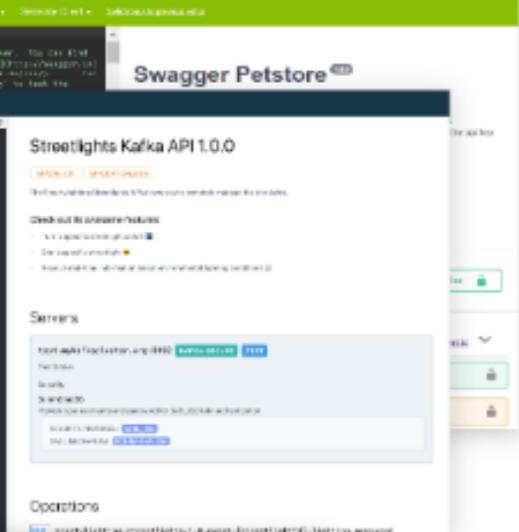
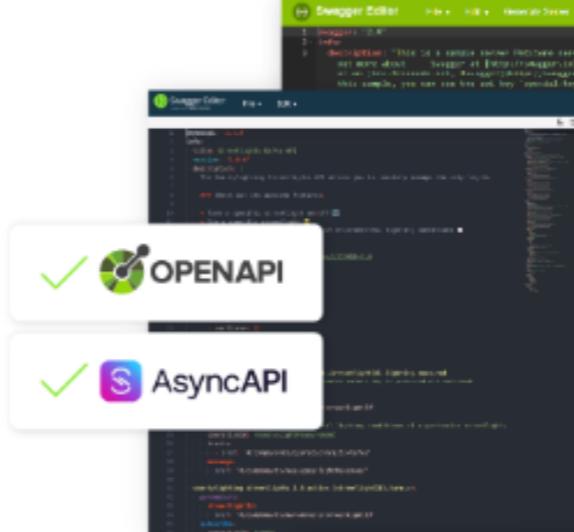


Swagger Editor

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

The screenshot shows the Swagger Editor interface. On the left, there is a code editor displaying an OpenAPI specification. The code is as follows:

```
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
```

The main panel displays the API documentation. At the top, it says "Demo 1.0.0 OAS 3.0". Below that, there is a "Servers" dropdown set to "http://localhost:3000". The "default" endpoint is selected, showing a "GET /ping" operation. The "Parameters" section indicates "No parameters". The "Responses" section shows a "Curl" command:

```
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



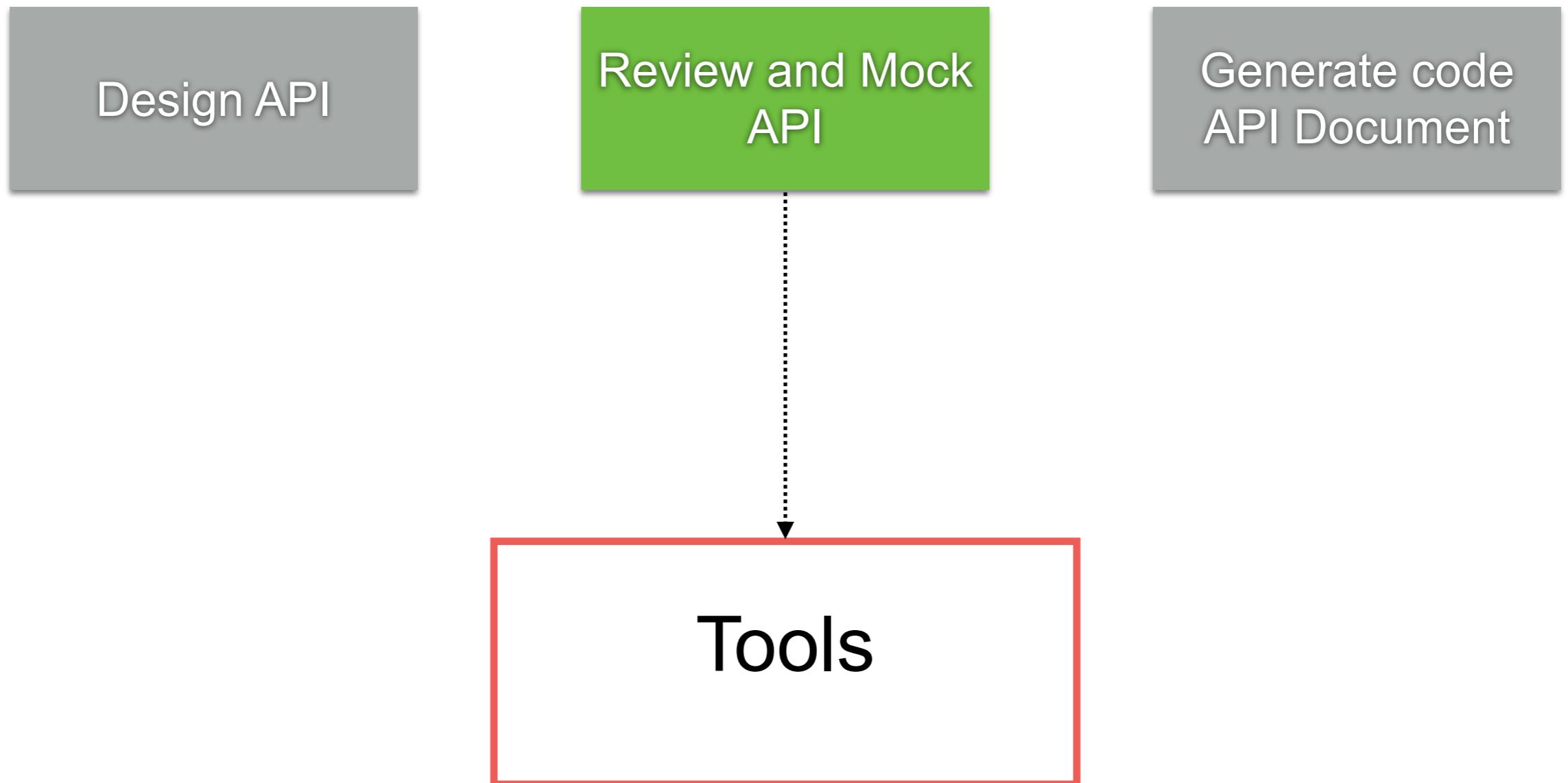
Add more paths

```
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 Open Source website with the following elements:

- Header:** Includes the Swagger logo (green circle with three dots), the text "Swagger Supported by SMARTBEAR", a search bar, and a "Sign In" button.
- Navigation:** Links for "Why Swagger?", "Tools", "Resources", "Swagger Open Source", "Editor", "Codegen", "UI", and a green "Try SwaggerHub" button.
- Section:** A large green header "Swagger Codegen".
- Text:** A paragraph explaining that Swagger Codegen can simplify the build process by generating server stubs and client SDKs for any API defined with the OpenAPI specification.
- Button:** A green "Download" button.
- Code Preview:** A dark terminal-like window displaying a list of available clients and servers. The 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 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.

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



Installation and how to use !!

\$brew install swagger-codegen

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

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



OpenAPI Mockers

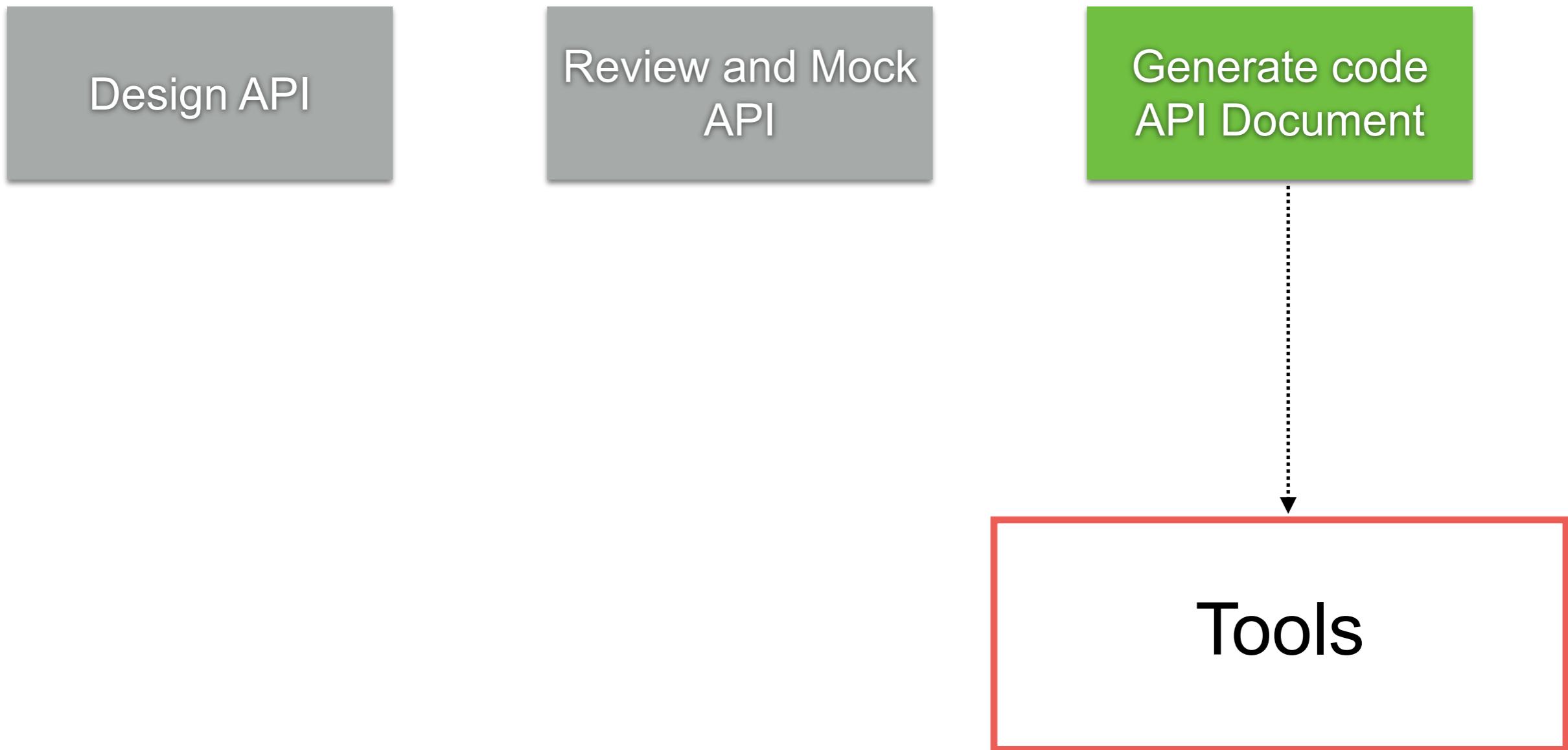
```
$npm i -g open-api-mocker  
$open-api-mocker -s api.yml -w
```

Access to <http://localhost:5000>

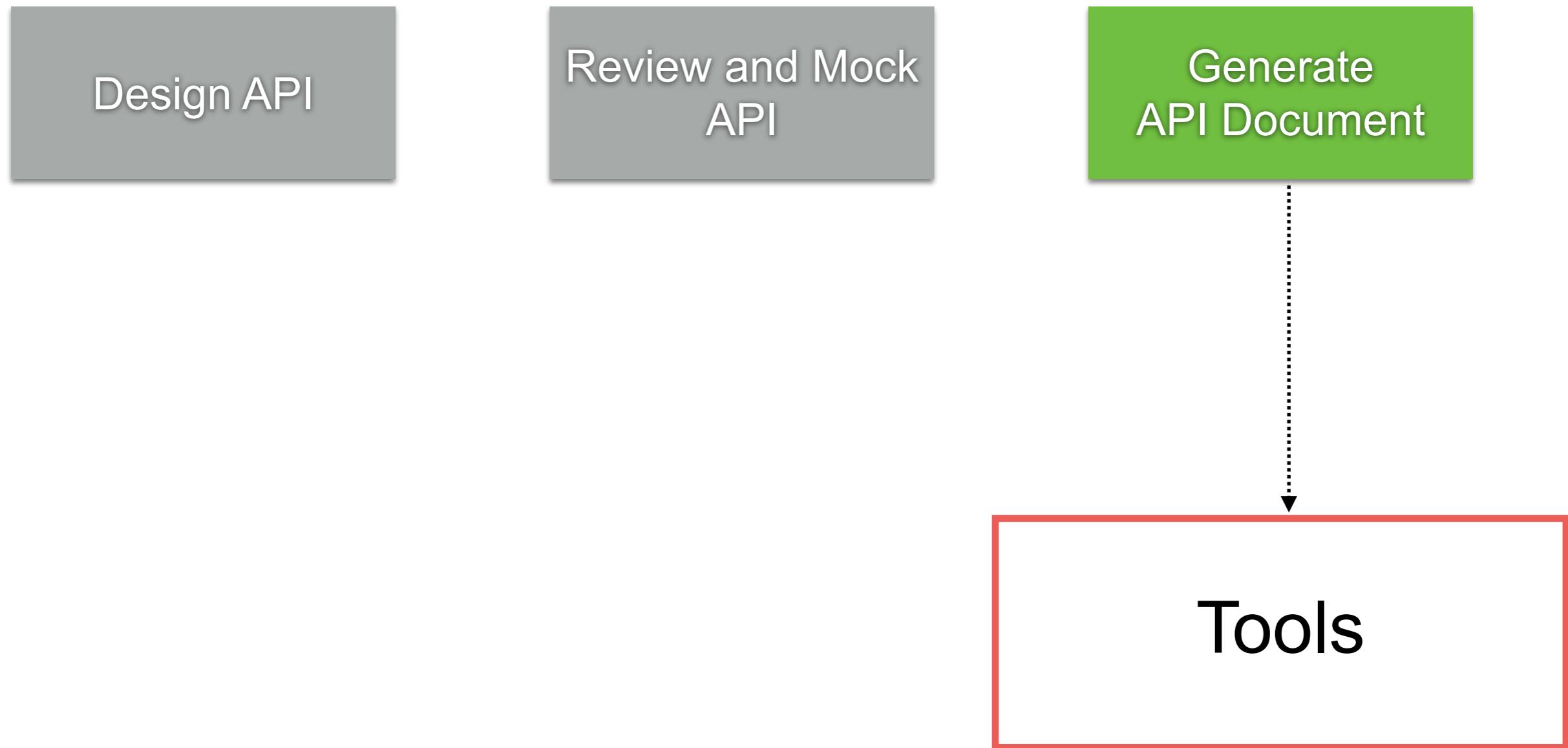
<https://github.com/jormaechea/open-api-mocker>



Step 3 :: Generate !!



Generate API Document



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 screenshot shows the Redoc API documentation interface. On the left, there's a search bar and two main sections: '/ping' and '/users/{id}'. The '/ping' section has a 'Responses' section with a green '200 OK' button. The '/users/{id}' section has a 'PATH PARAMETERS' section with a parameter 'id' (integer, required) described as 'User ID of the user'. It also has a 'Responses' section with '200 OK' and '404 User not found' buttons.

On the right, there's a dark-themed view of the same API documentation. It shows the '/ping' endpoint with a green 'GET /ping' button and a dropdown arrow. Below it, the '/users/{id}' endpoint is shown with a green 'GET /users/{id}' button and a dropdown arrow. Underneath, there's a 'Response samples' section with '200' and '404' buttons, a 'Content type application/json' section, and a JSON response sample:

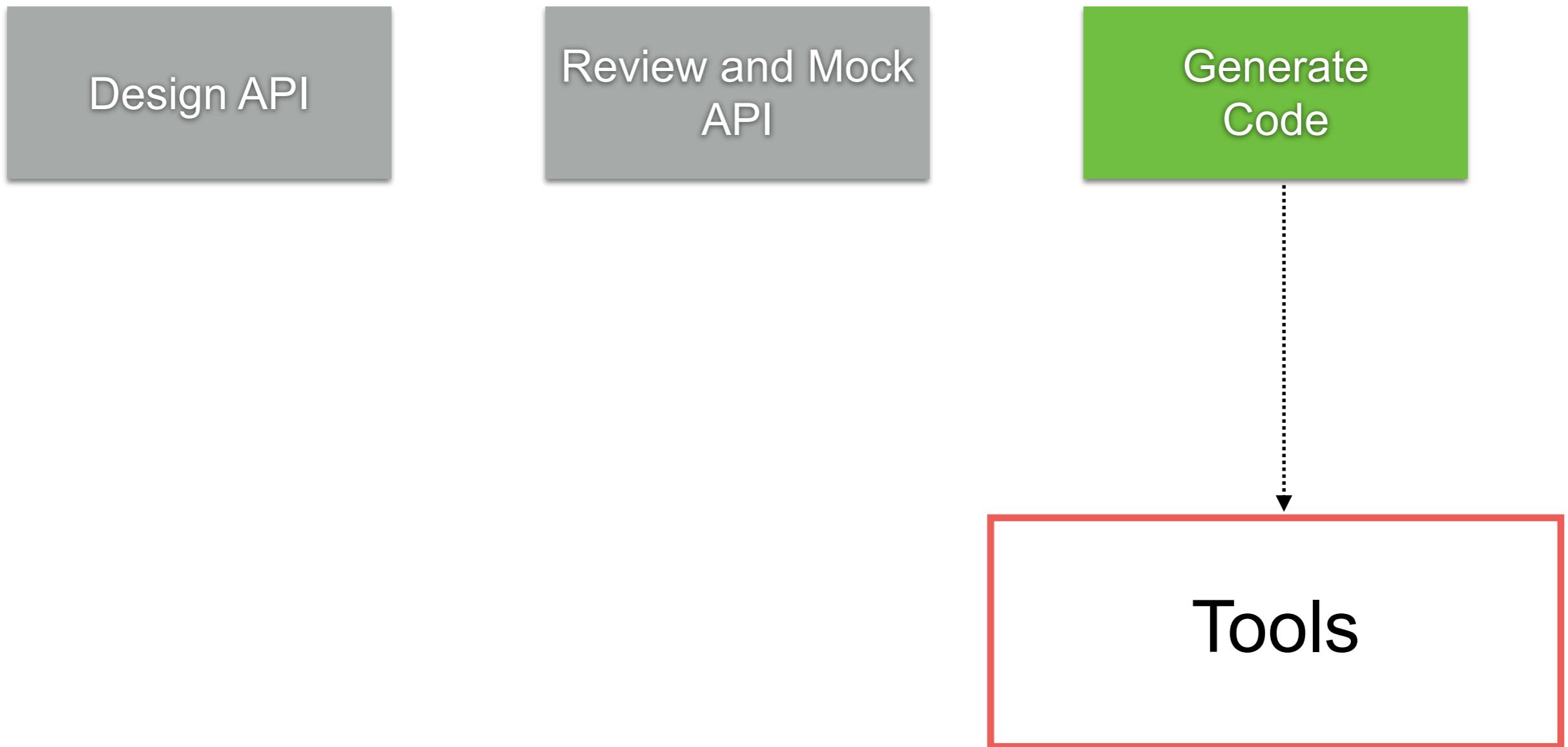
```
{  
  "id": 1,  
  "name": "User 01"  
}
```

With a 'Copy' button next to it.

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



Generate Code



Generate Code from Swagger

Server-side
Client-side



Swagger Codegen™

Supported by SMARTBEAR

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



Working with Postman

API-first

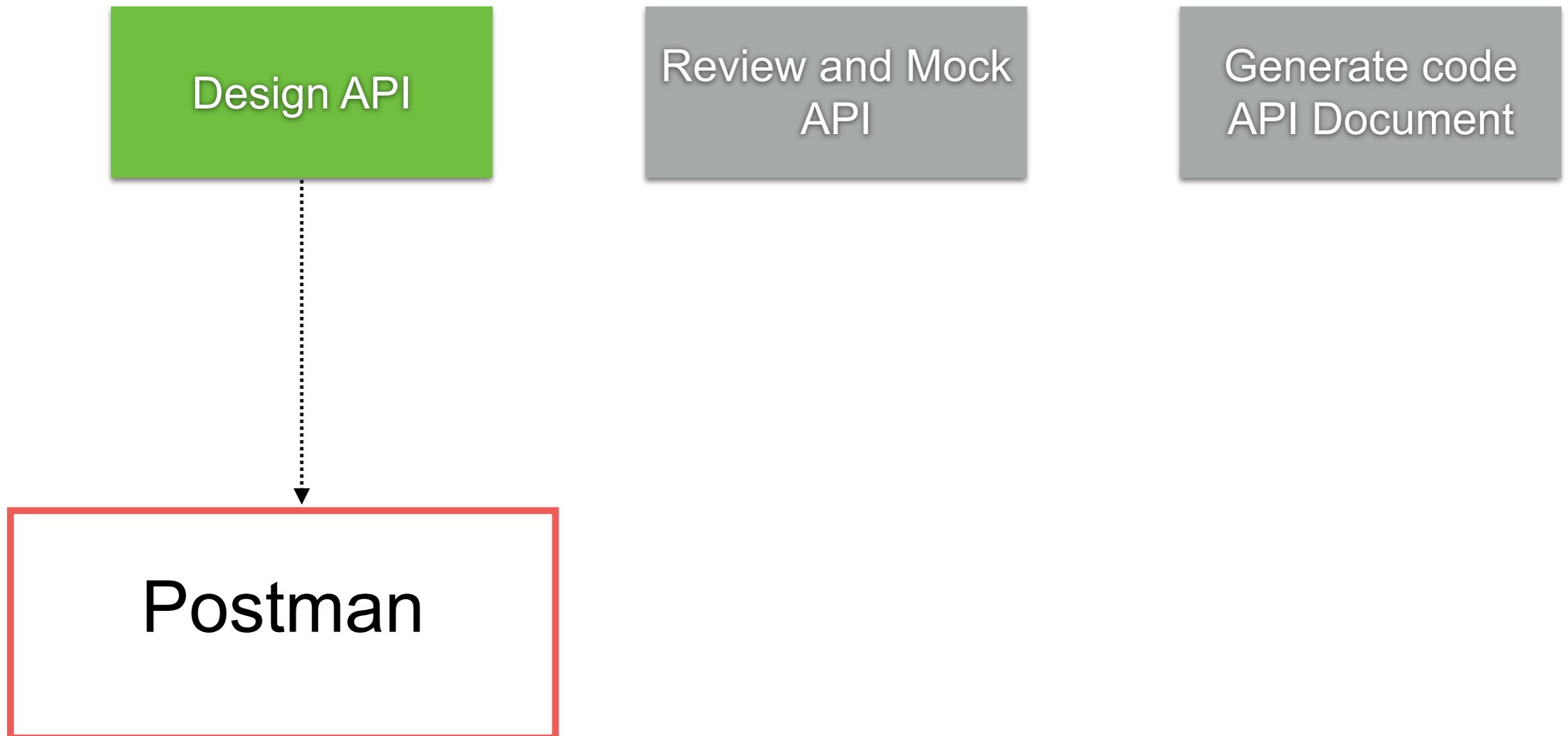


POSTMAN

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



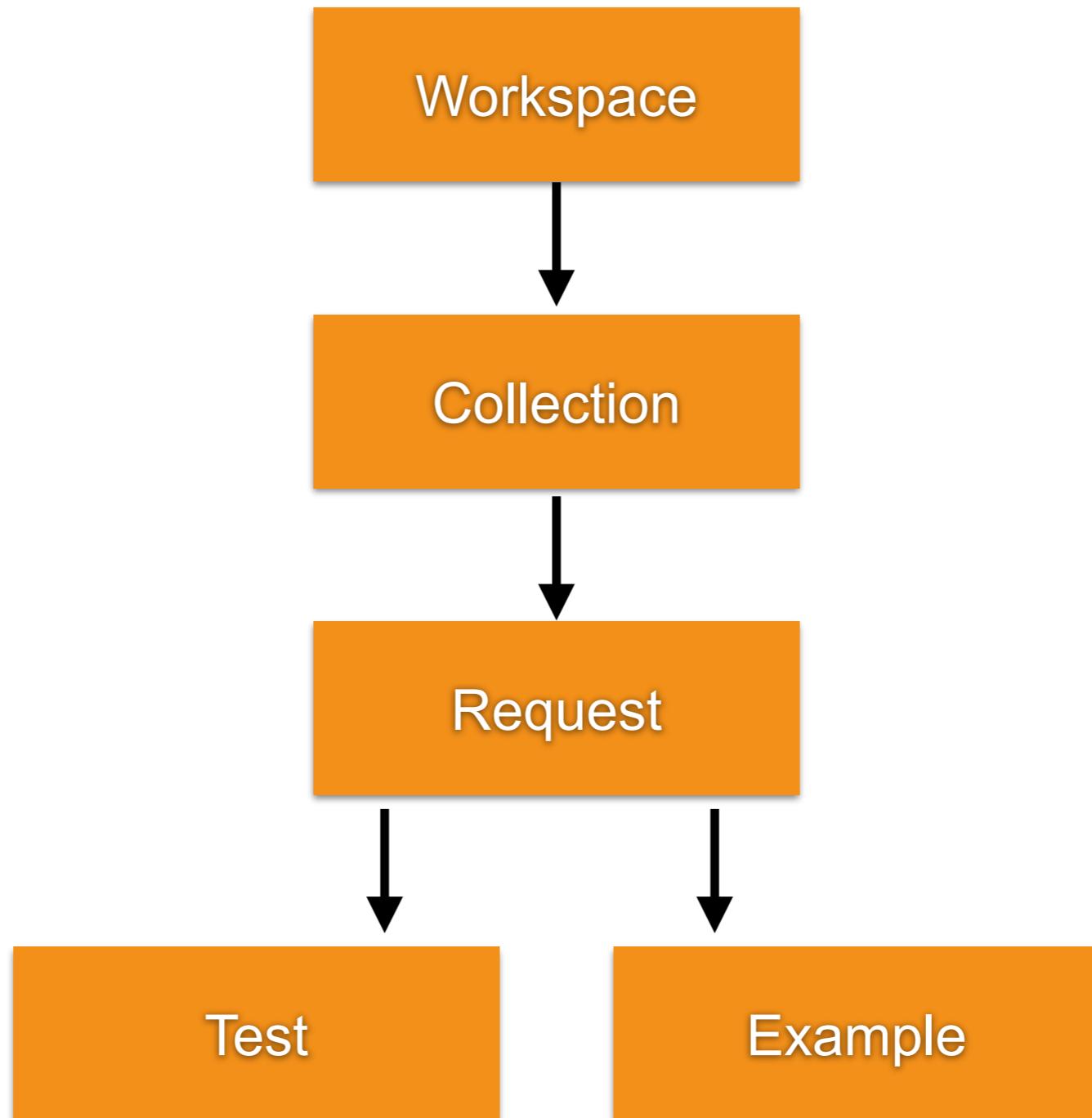
Step 1 :: Design API



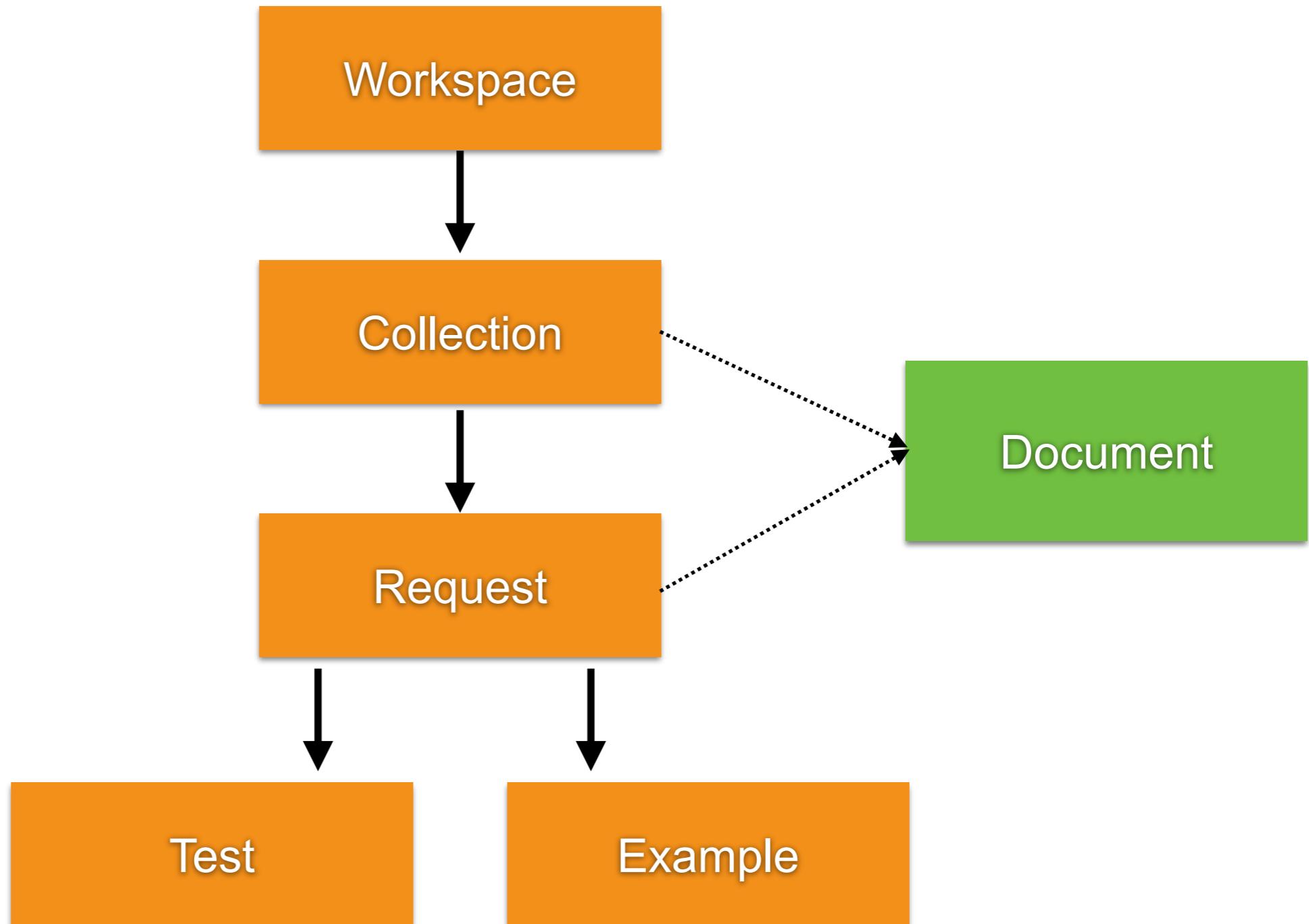
<https://www.postman.com/>



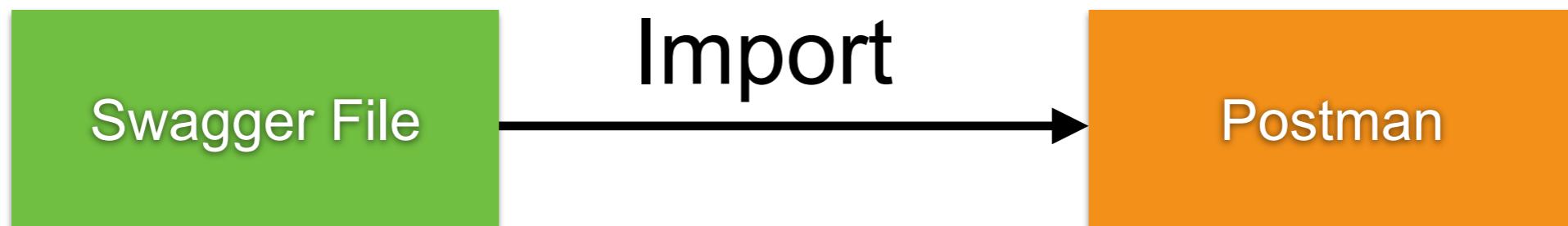
API-first in Postman



API-first in Postman



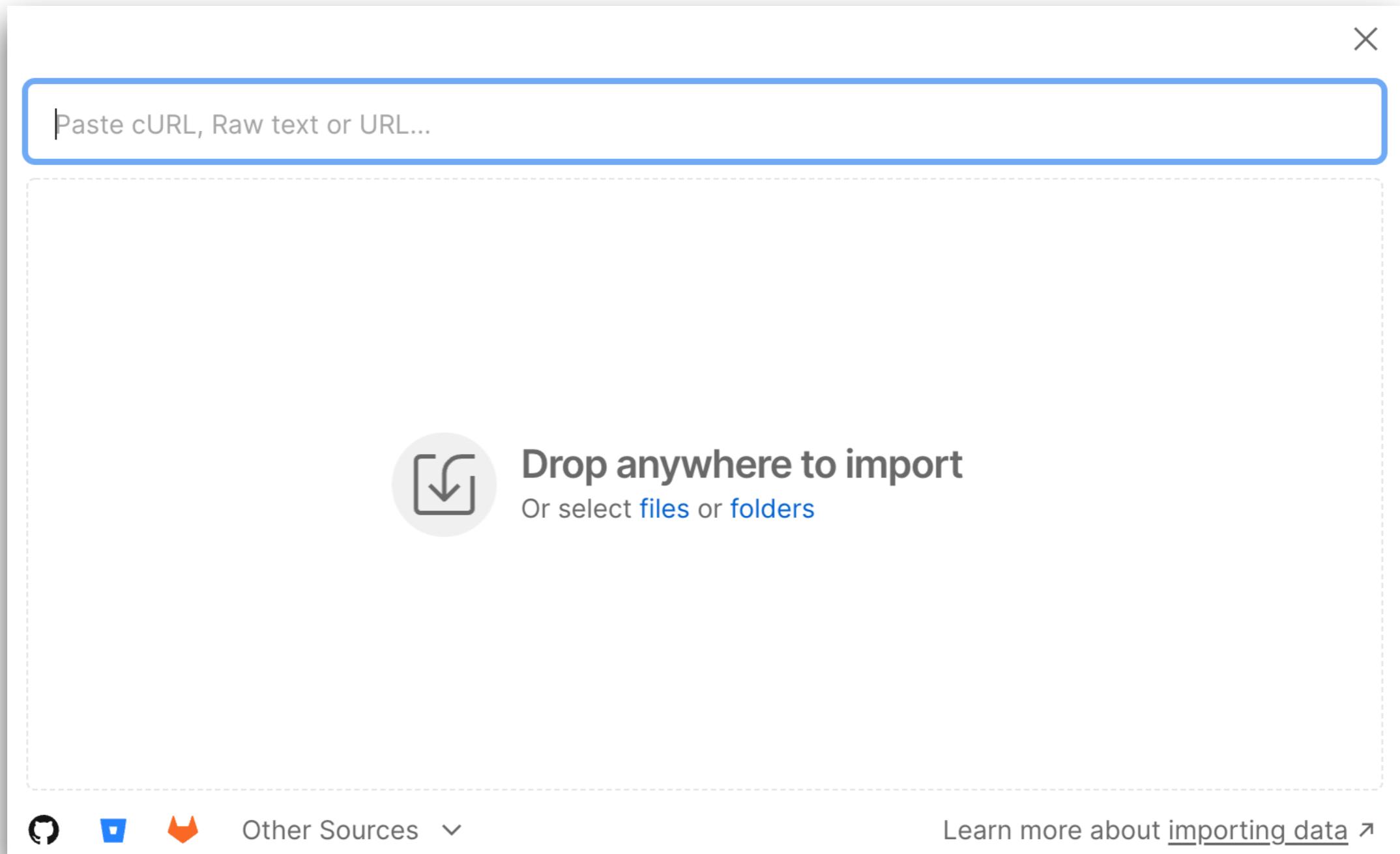
Import from Swagger/OpenAPI



POSTMAN



Import with OpenAPI file



Example for Design your APIs

The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of API endpoints under a 'Demo' category. The selected endpoint is 'GET /users/:id'. To the right, the main panel displays the request configuration for this endpoint.

Request URL: `HTTP {{baseUrl}} / users / {id} / /users/:id`

Method: `GET`

Params: (selected tab)

Query Params: (empty table)

Key	Value
Key	Value

Path Variables: (empty table)

Key	Value
id	<integer>



Postman Collection to OpenAPI

Transform Postman Collections into OpenAPI

postman2openapi

Fork me on GitHub

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 sample API calls. It provides endpoints for `GET`, `POST`, `PUT`, various auth mechanisms and other utility endpoints.",
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                   "",
24                   "pm.test(\"response body has json with request queries\", function () {
25                     pm.response.to.have.jsonBody('args.foo1', 'bar1'),
26                     .and.have.jsonBody('args.foo2', 'bar2');
27                   })",
28                 ],
29                 "type": "text/javascript"
30               }
31             }
32           ],
33           "request": {
34             "method": "GET",
35             "header": [],
36             "url": {
37               "raw": "https://postman-echo.com/get?foo1=bar1&foo2=bar2",
38               "protocol": "https",
39               "host": [
40                 "postman-echo",
41                 "com"
42               ],
43               "port": null,
44               "query": [
45                 {
46                   "key": "hand",
47                   "value": "wave"
48                 }
49               ],
50               "fragment": null
51             }
52           }
53         }
54       ]
55     }
56   ]
57 }
```

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

The screenshot shows the Postman website with a yellow callout pointing to the desktop application interface. The interface includes a sidebar with collections, environments, APIs, mock servers, monitors, flows, and history. The main area shows a request for 'Notion API / Databases / Retrieve a database' with a GET method and URL <https://api.notion.com/v1/databases/:id>. It displays query parameters, path variables, and a JSON response body. A sidebar on the right shows documentation for the Notion API, including authorization (Bearer Token), request header (Notion-Version 22-02-22), and path variables (id). At the bottom, there's a section for what Postman is, featuring an illustration of three astronauts launching a rocket from a laptop.

Product ▾ Pricing Enterprise ▾ Resources and Support ▾ Explore

Search Postman

Sign In Sign Up for Free

Public APIs together

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

jsmith@example.com Sign Up for Free

Download the desktop app for

Windows Apple Linux

What is Postman?

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

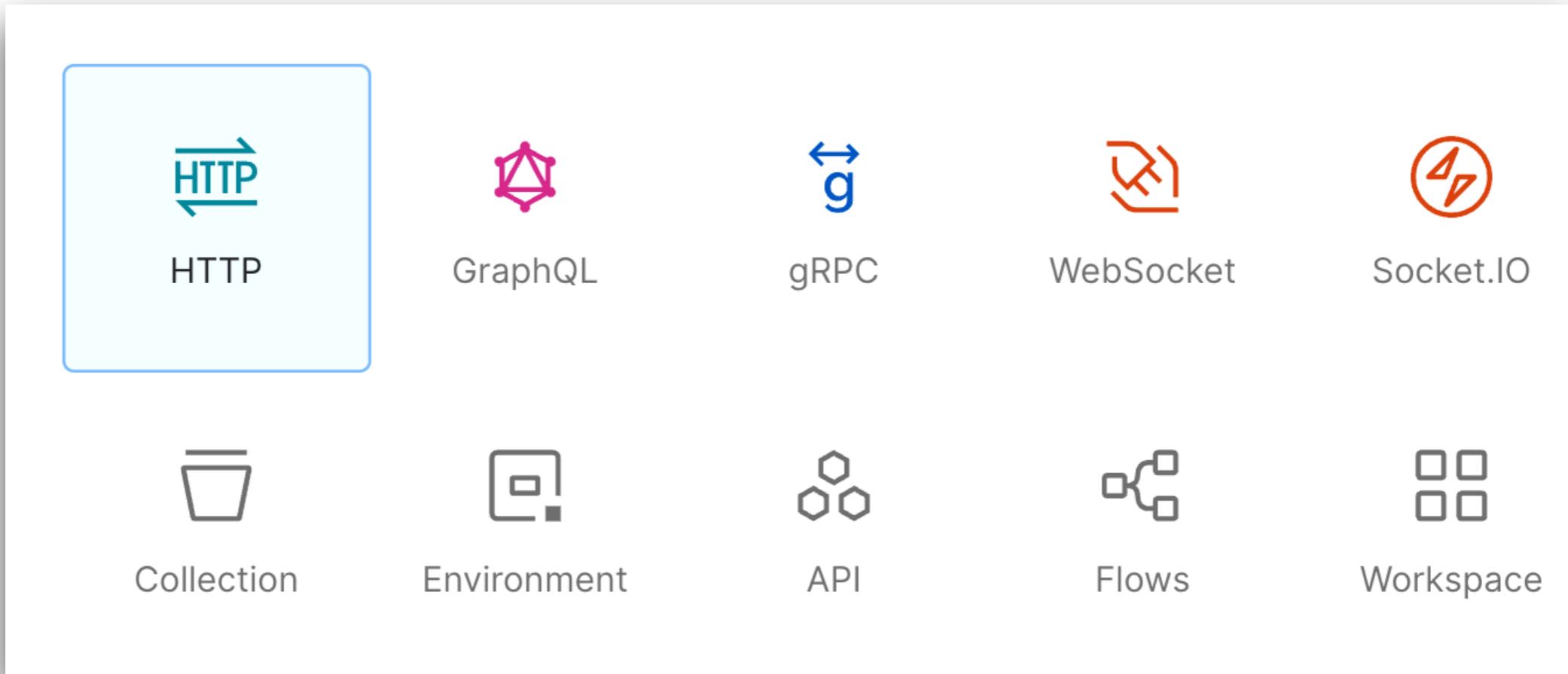


Topics with Postman

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



Testing with Postman



Topics with Postman

The screenshot shows the Postman application interface. On the left, there's a sidebar with icons for Collections, Environments, History, and a plus sign. The main area shows a collection named "demo-nodejs" with a "group01" folder. Inside "group01" is a "GET Hello API" request. The "Tests" tab is currently selected, displaying 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"
16     }
17 }
```

To the right of the code, there's a sidebar with several items:

- Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#).
- 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

At the bottom of the interface, there are tabs for Body, Cookies, Headers (7), and Test Results (3/3). The Body tab is selected, showing a JSON response with one key:

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



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 testing an HTTP request. The URL is set to `http://localhost:3000/`. The 'Tests' tab is selected, indicated by a red box and a green dot. The test script is as follows:

```
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"  
16         }  
17     }  
18 };
```

To the right of the script, there is a sidebar with several items:

- Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#) ↗
- 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>



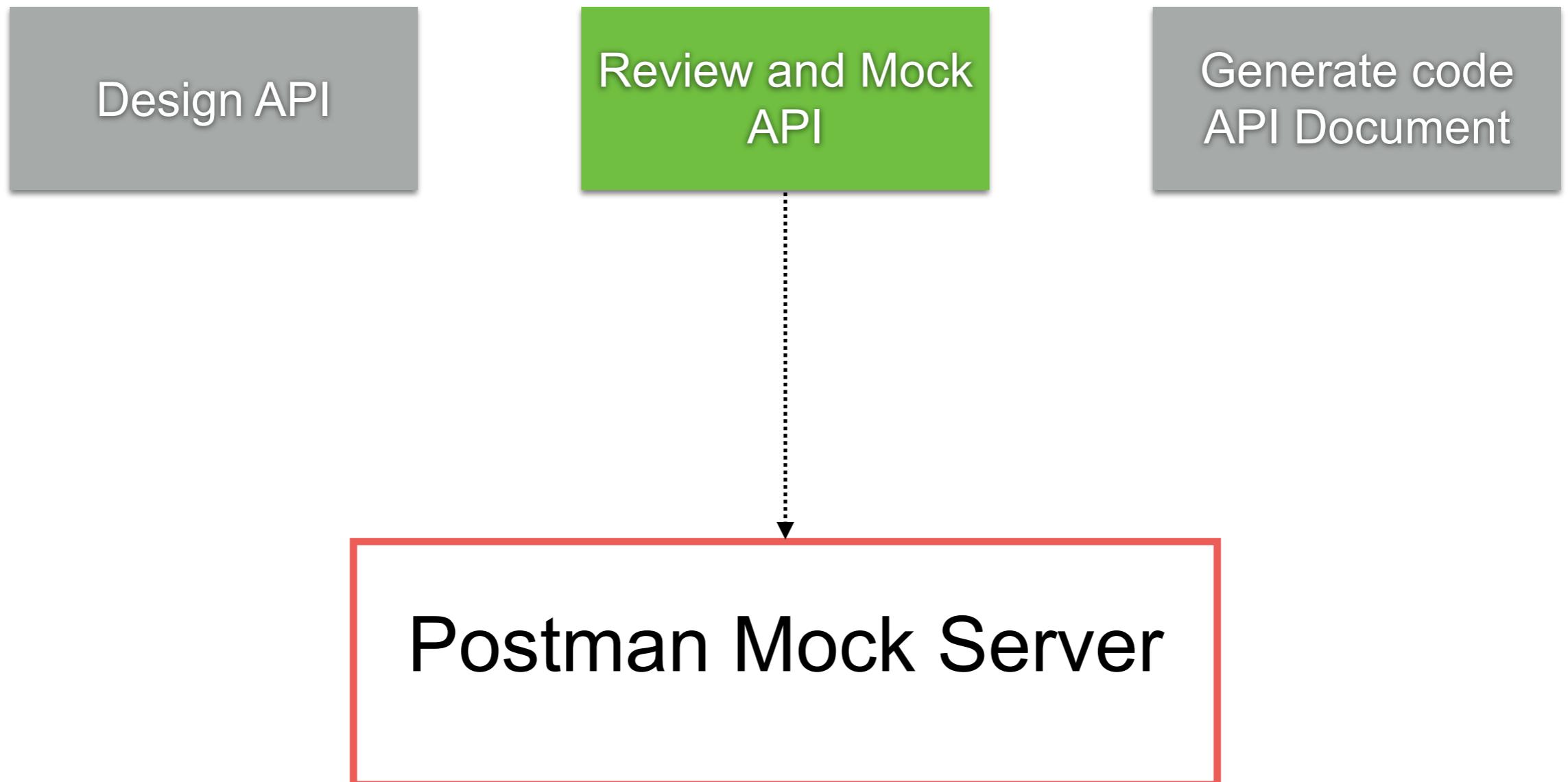
Newman and report

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

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



Step 2 :: Mock API



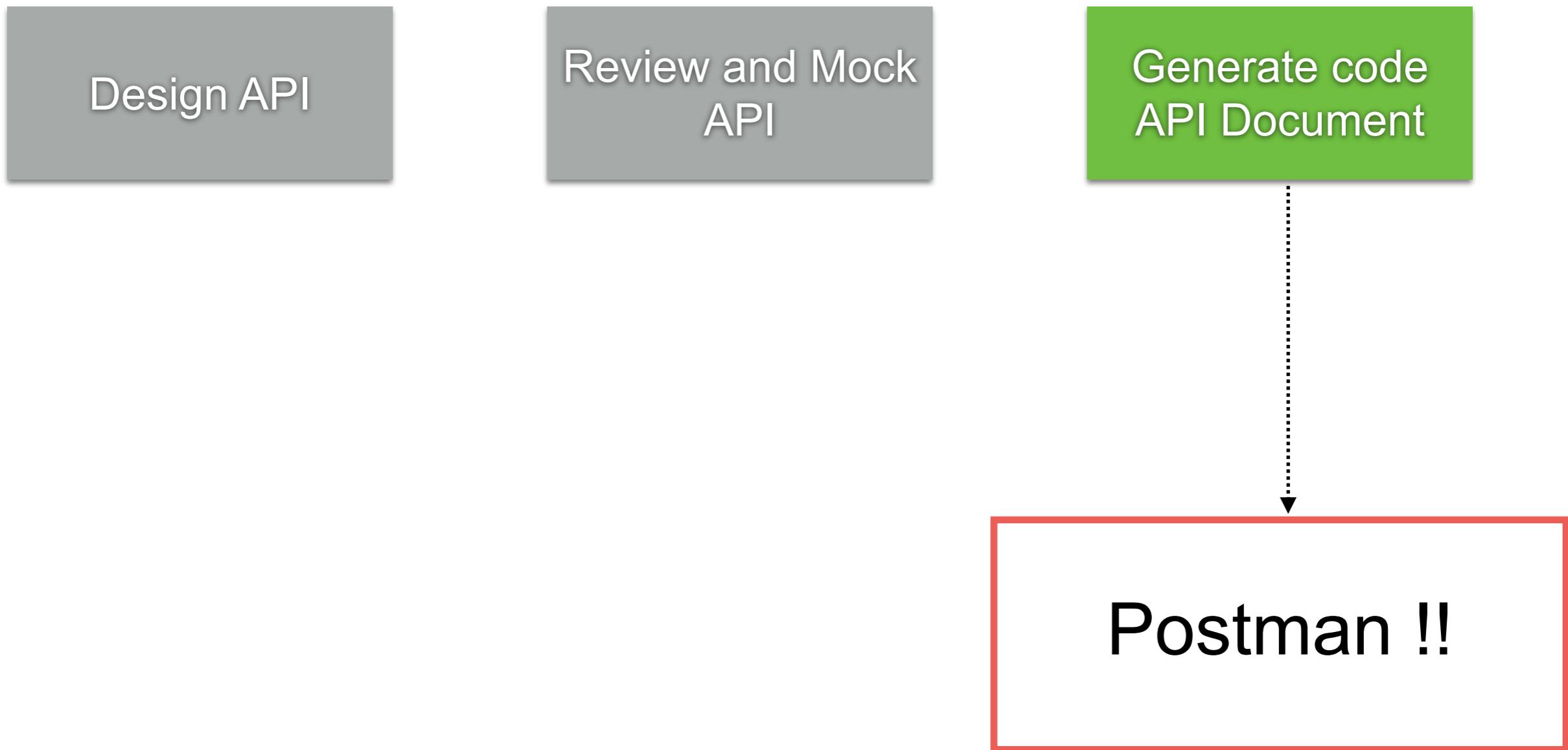
Create Mock Server from Collection

The screenshot shows the Postman application interface. On the left, there's a sidebar with icons for Collections, APIs, Environments, **Mock Servers** (which is highlighted with a red box), Flows, and History. The main area has a title "Create a mock server" with two tabs: "1. Select collection to mock" (underlined) and "2. Configuration". Below the tabs is a button bar with "Create a new collection" and "Select an existing collection". A text instruction says "Enter the requests you want to mock. Optionally, add a request body by clicking on the (...) icon." A table below shows a single row: Request Method (GET), Request URL ({{url}}/ Path), Response Code (200), and Response Body (Response Body). At the bottom are "Cancel" and "Next" buttons.

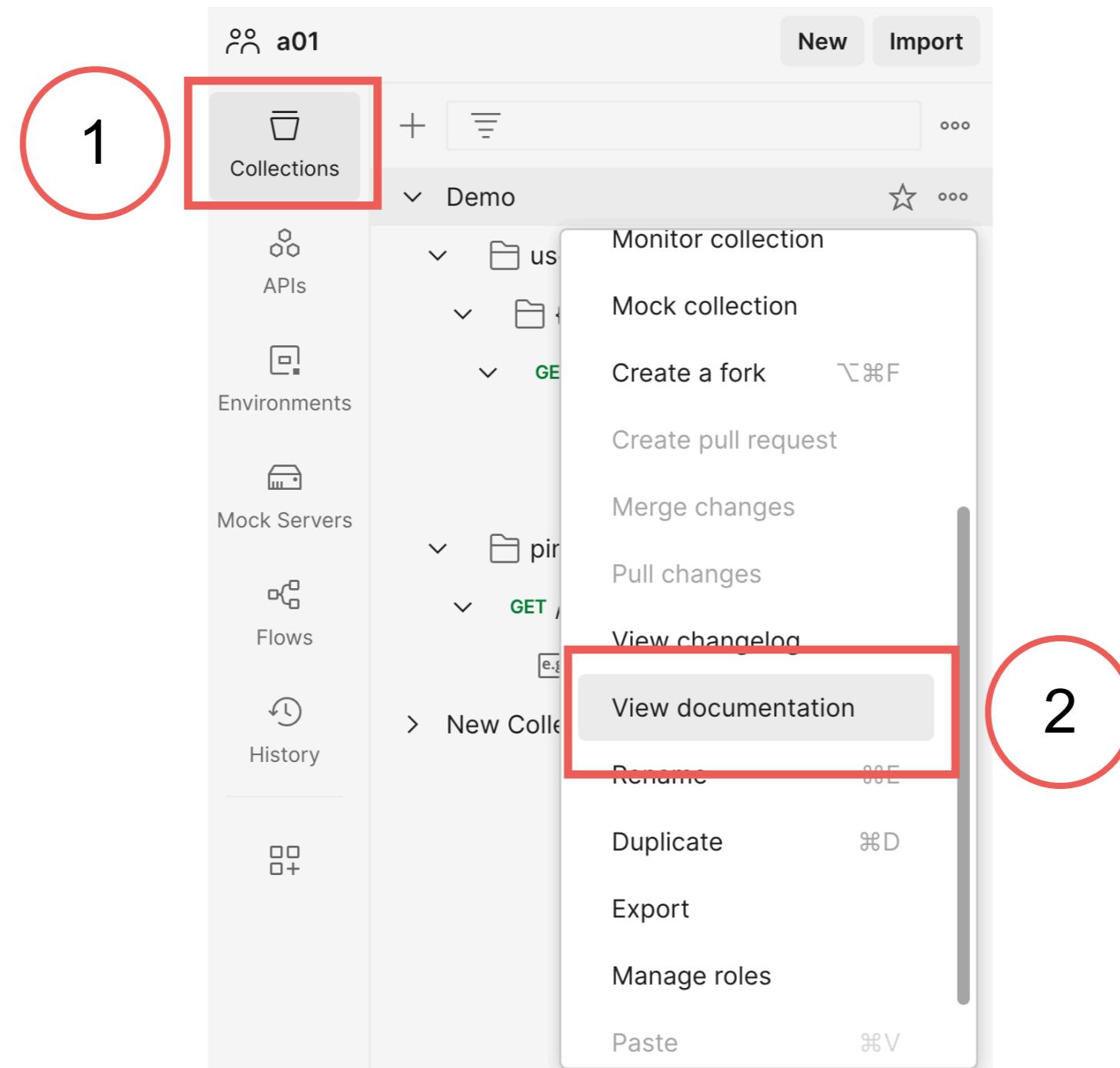
<https://learning.postman.com/docs/designing-and-developing-your-api/mockng-data/setting-up-mock/>



Step 3 :: Generate !!



Generate API Document



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



Generate Code with APIs

The screenshot shows the Postman application interface. On the left, there's a sidebar with icons for Collections, APIs, Environments, Mock Servers, Flows, History, and a plus sign. The 'APIs' icon is highlighted with a red box and circled with a red number '1'. In the main area, a 'New API' collection is selected. The interface includes a 'New API' button, a 'DEV' environment switch, a cartoon character, and a 'Quickstart your API by connecting a repository' section. Below this are buttons for GitHub, Bitbucket, GitLab SaaS, and Azure DevOps. A red box highlights the text 'Or, continue without a repository' in a white box, which is circled with a red number '2'.

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



Create APIs from Collections

The screenshot shows the Postman interface for creating a new API. On the left sidebar, under the 'APIs' section, there is a red circle with the number '3' indicating pending items. The main area is titled 'New API' and contains fields for 'About this API' (summary and description) and a 'Connect repository' section with a 'Connect' button. A large red box highlights the 'Collections' section, which lists a single item: 'Demo'. To the right of the collections is a 'Definition' section with a note to 'Add files to describe your service and setup the API definition.' On the far right, there are sections for 'Test and Automation', 'API Performance', and 'Deployments', along with a 'Publish API' button. The top navigation bar includes 'New' and 'Import' buttons, and the status bar shows 'DEV'.

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



Generate code

The screenshot shows the Postman interface for a 'New API'. A modal window titled 'Code Generation (BETA)' is open, overlaid on the main API details page. The modal contains instructions to generate server boilerplate from an API schema, a link to learn more, and a dropdown menu for selecting a language and framework. The dropdown is currently set to 'Select' and shows options for Go - Chi server, NodeJs - Express, Java - JAX-RS, and Python - Flask. The entire 'Code Generation' section is highlighted with a red border.

New API

Share 0

DEV

Code Generation (BETA)

Generate server boilerplate from your API schema. [Learn more](#)

Language and framework

Select

Only generate routes and infrastructure

Generate Code

Go - Chi server

NodeJs - Express

Java - JAX-RS

Python - Flask

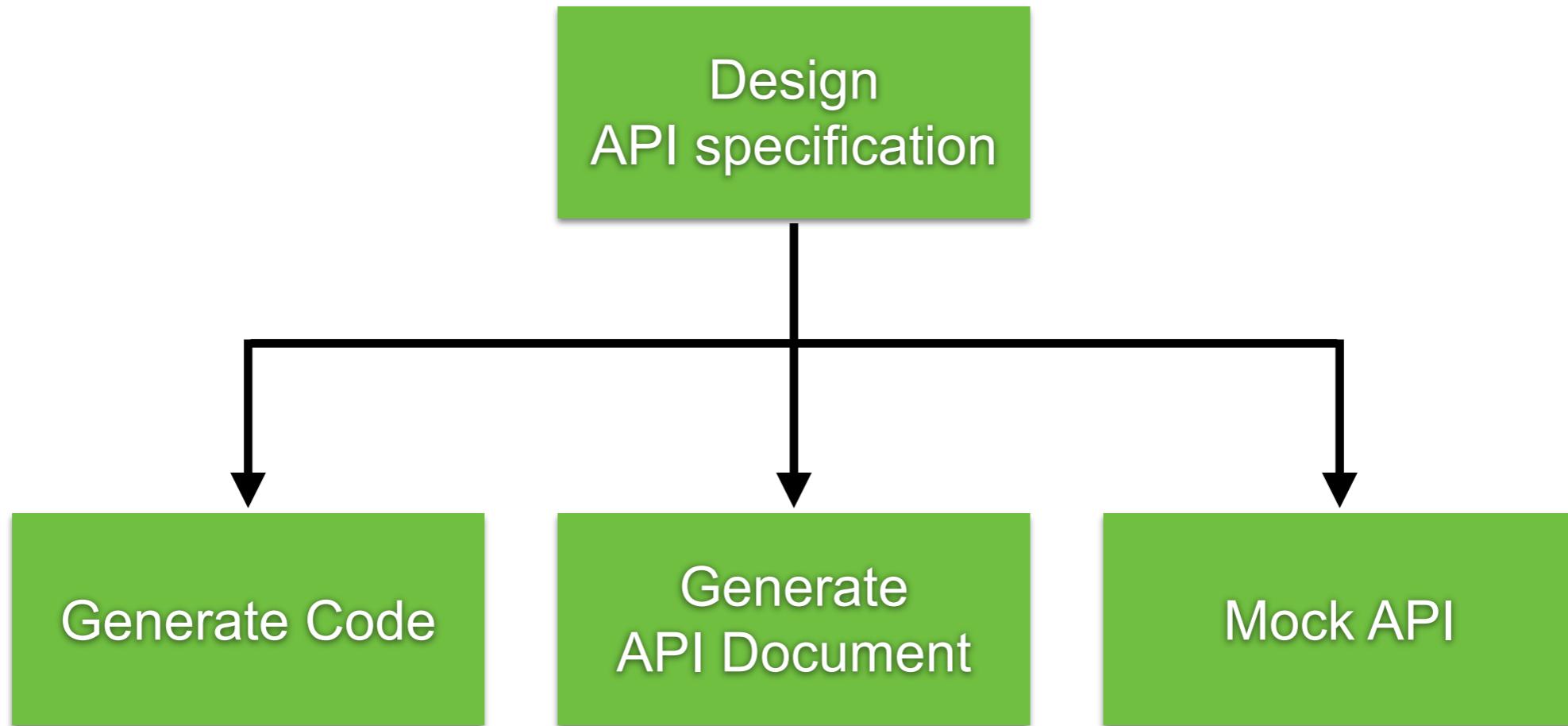
<https://learning.postman.com/docs/designing-and-developing-your-api/developing-an-api/generating-server-code/>



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/>



Automated testing
© 2017 - 2018 Siam Chamnkit Company Limited. All rights reserved.

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 homepage of the API Blueprint Tools website. At the top, there is a navigation bar with links for 'Docs', 'Tools' (which is underlined), 'Developers', and 'Support'. Below the navigation bar is a search bar labeled 'Search Tooling'. Under the search bar, there is a horizontal menu with tabs: 'Tools' (highlighted in purple), 'Editors', 'Testing', 'Parsers', 'Mock servers', 'Renderers', 'Converters', and 'Lexers'. The main content area features three tool cards:

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

<https://apiblueprint.org/>



Apiary Editor

The screenshot shows the Apiary Editor interface. On the left, there is a code editor window displaying API documentation in a text-based format. The code includes sections for introduction, parameters, and schema. On the right, the generated API documentation is displayed, featuring sections for introduction, reference, and specific endpoints like 'Ping service' and 'Get user by id'. The interface has a dark theme with various buttons and links at the top.

```
1 FORMAT: 1A
2 HOST: http://localhost:3000
3
4 # Demo API
5
6 Demo APIs for API-first solution !!
7
8 # Ping service [/ping]
9 ## Ping [GET]
10 + Response 200
11
12 # Get user by id [/users/{id}]
13
14 + Parameters
15   + id
16
17 ## Get user by id [GET]
18 + Response 200 (application/json)
19   + Body
20
21   {
22     "id": 1,
23     "name": "User 01"
24   }
25
26 + Schema
27
28   {
29     "$$schema": "http://json-schema.org/draft-04/schema#",
30     "type": "object",
31     "properties": {
32       "id": {
33         "type": "integer"
34       },
35       "name": {
36         "type": "string"
37       }
38     }
39
40
41
```

Demo API

INTRODUCTION

Demo APIs for API-first solution !!.

REFERENCE

Ping service

Get user by id

Ping service

Ping

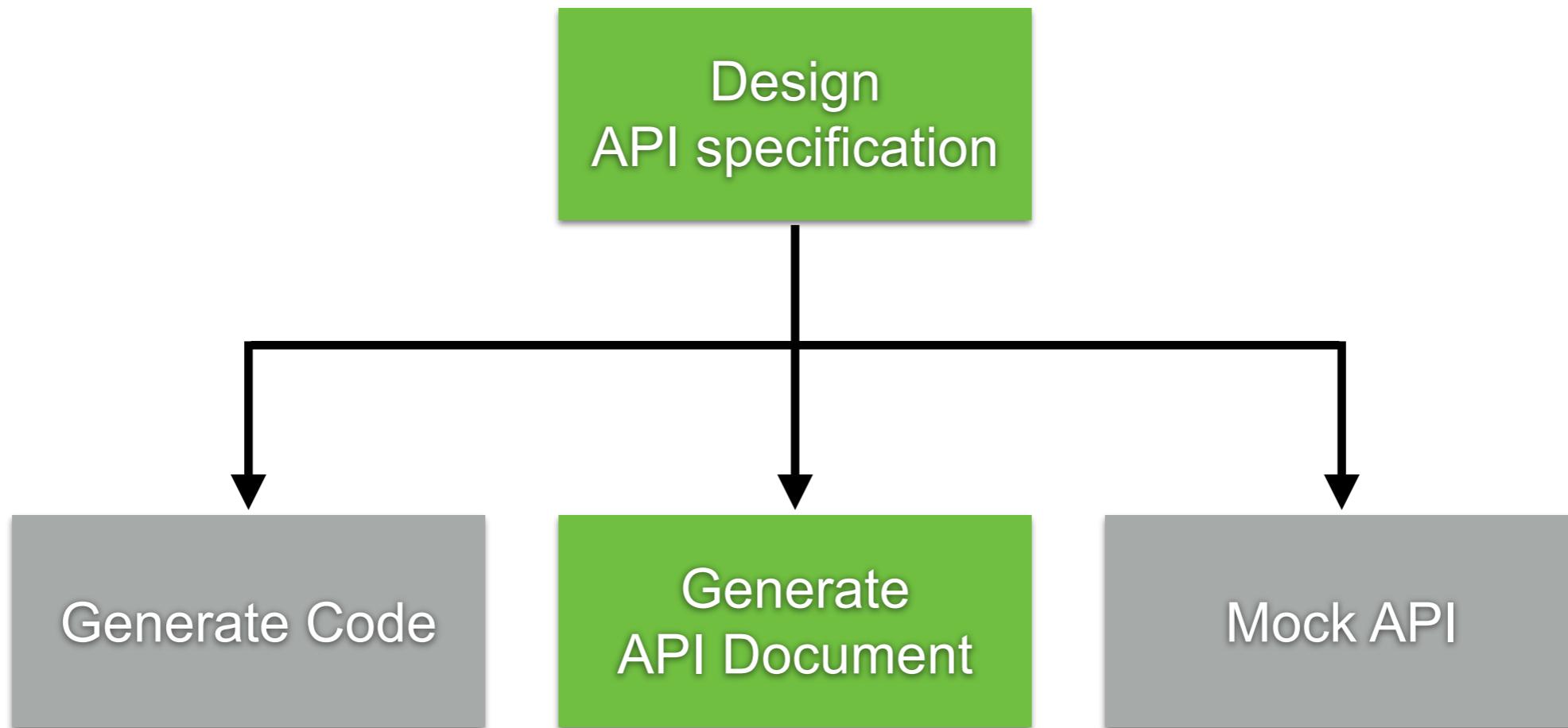
Get user by id

Get user by id

<https://apiary.io/>



API Blueprint



Generate API Document

```
$docker container run --rm -w /demo -v $(pwd):/demo apiaryio/client preview --output="docs.html"
```

The screenshot shows a web-based API documentation interface. At the top, it says "Demo API". Below that is a "INTRODUCTION" section with the text "Demo APIs for API-first solution !!.". Under "REFERENCE", there are two main service sections: "Ping service" and "Get user by id". The "Ping service" section has a "Ping" button. The "Get user by id" section has a "Get user by id" button.

<https://help.apiary.io/tools/apiary-cli/>



Generate API Document

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



aglio api blueprint renderer

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



Generate API Document

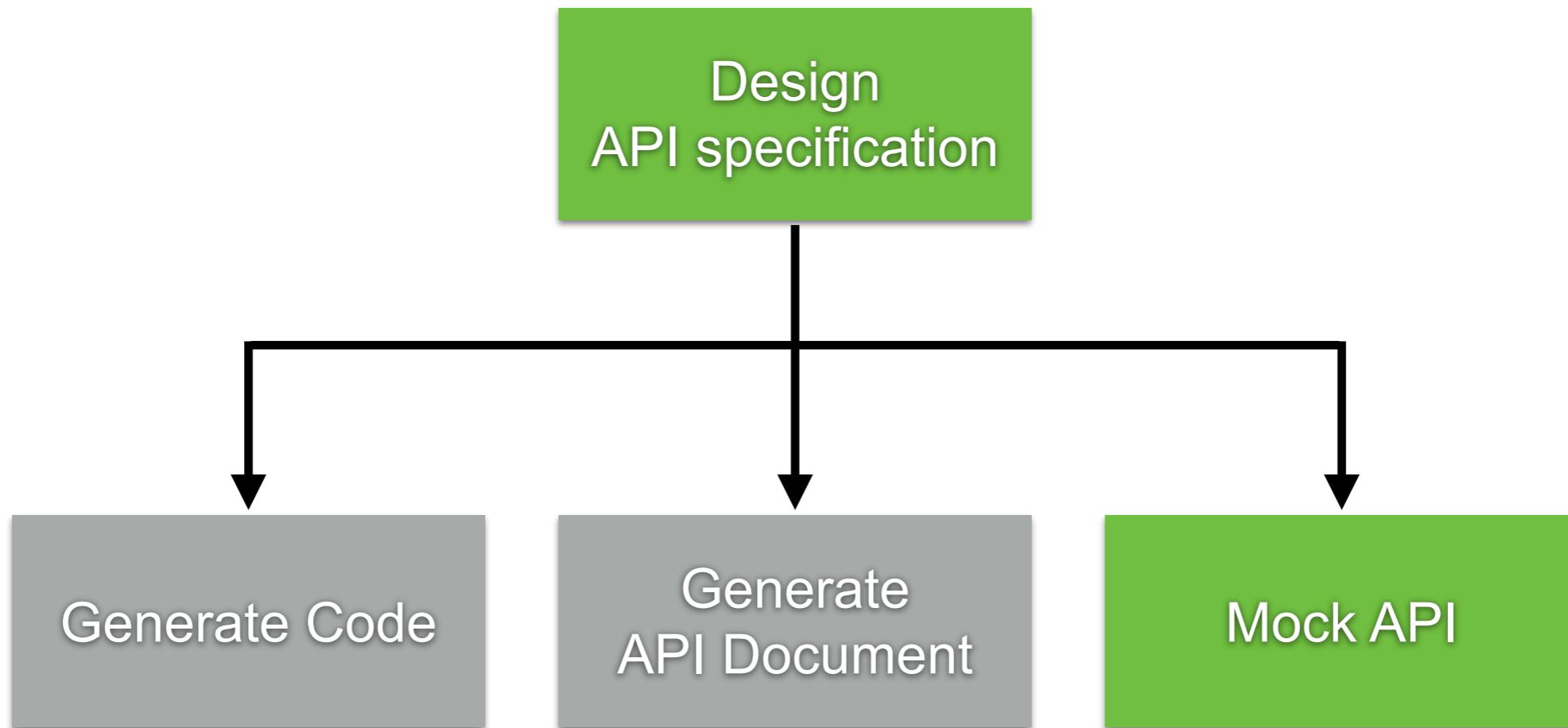
The screenshot shows the Aglio API documentation generator interface. On the left, there's a sidebar with a dropdown for 'Resource Group' currently set to 'Ping', and another dropdown for 'Get user by id'. Below the sidebar is the URL <http://localhost:3000>. The main content area is titled 'Demo API' with the subtitle 'Demo APIs for API-first solution !!.' It features a 'Resource Group' section containing two entries:

- PING SERVICE**: A GET method for '/ping' with an example URI of `GET http://localhost:3000/ping` and a response status of 200.
- GET USER BY ID**: A GET method for '/users/{id}' with an example URI of `GET http://localhost:3000/users/id`. This entry includes 'URI Parameters' with 'id' as a required string parameter, and a 'Response' status of 200. There are 'Hide' and 'Show' buttons next to the parameters.

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



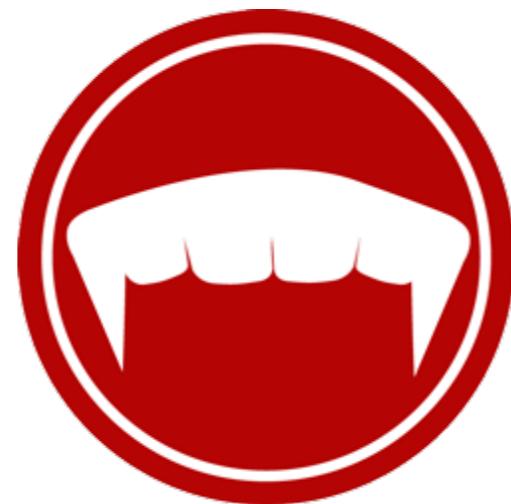
API Blueprint



Create Mock API

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

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



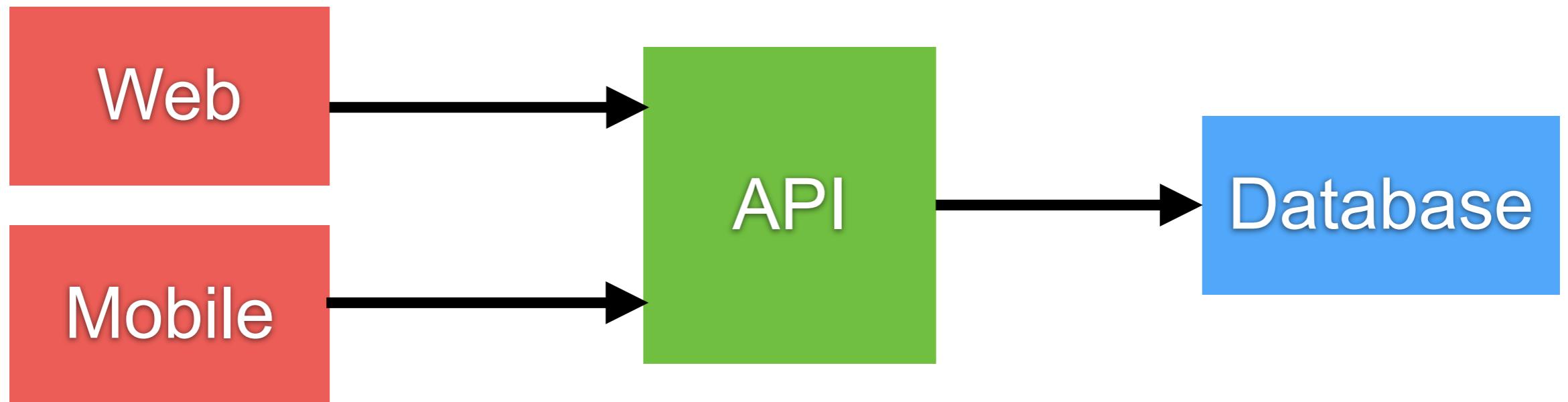
<https://www.npmjs.com/package/drakov>



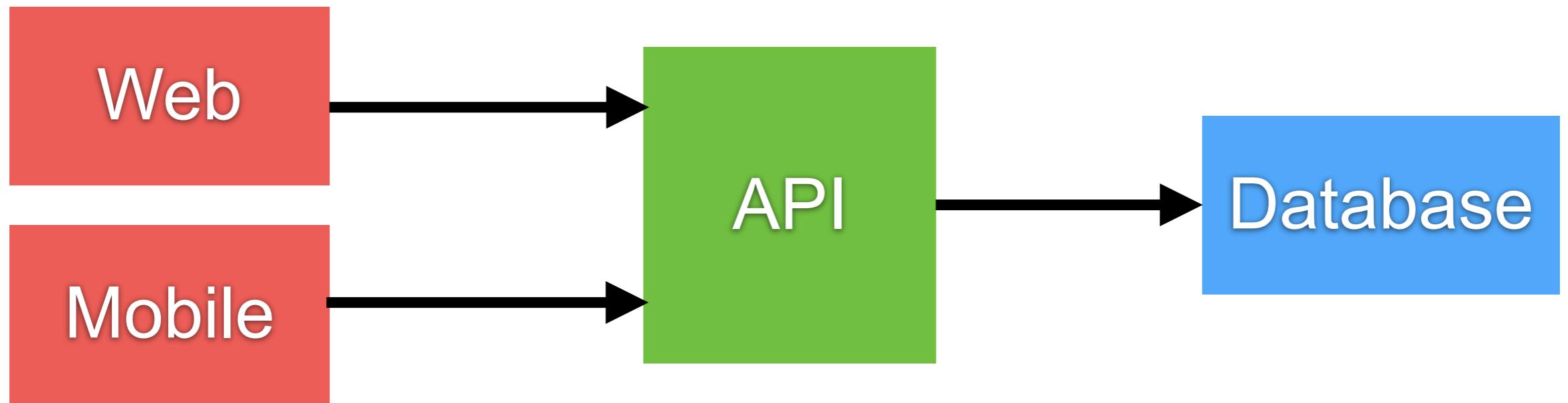
What, Why and How to Test ?



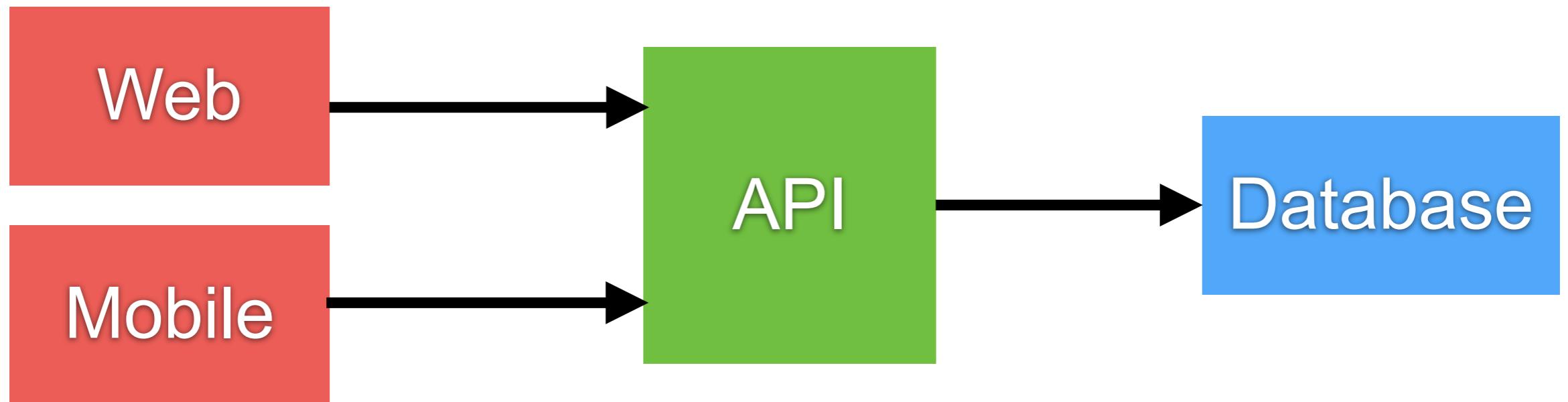
Architecture



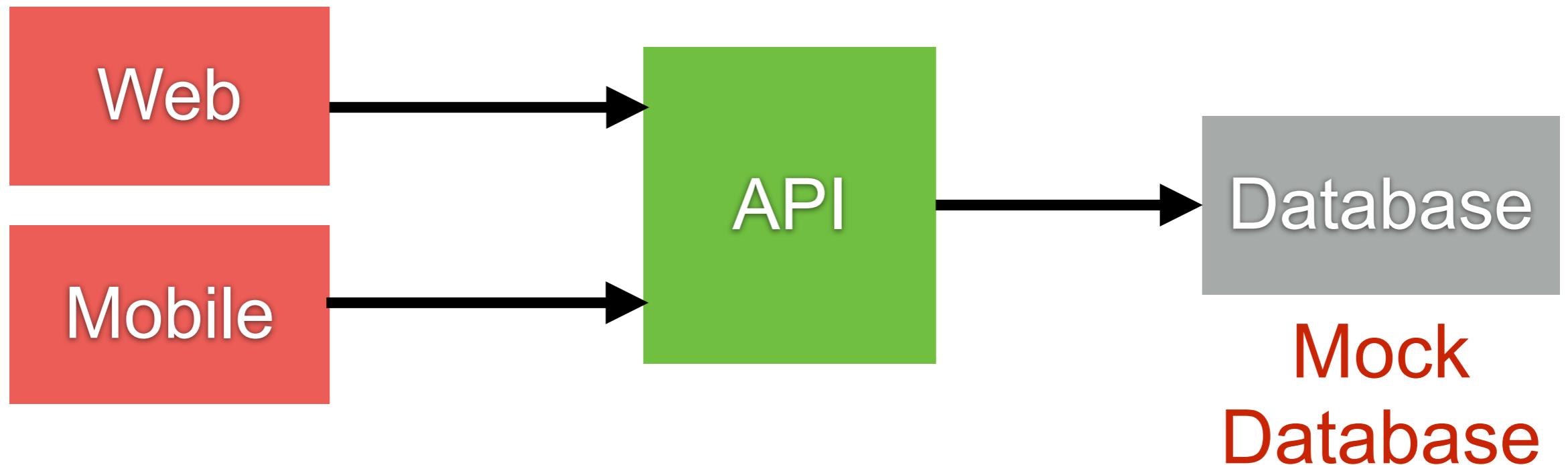
Test ?



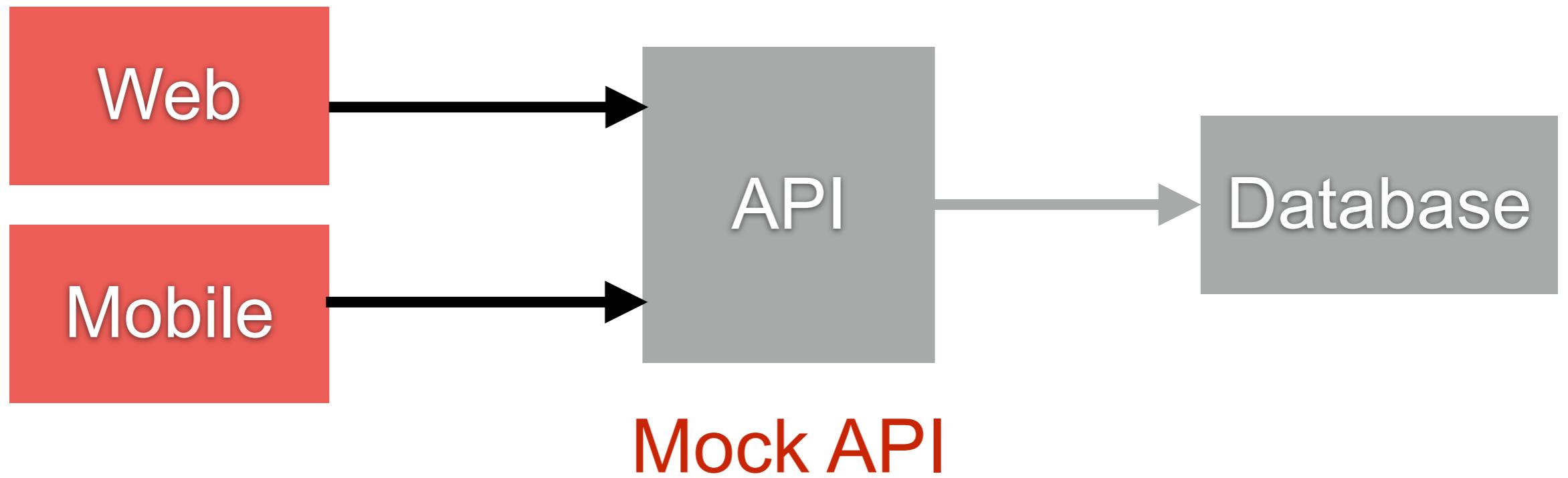
UI Test ?



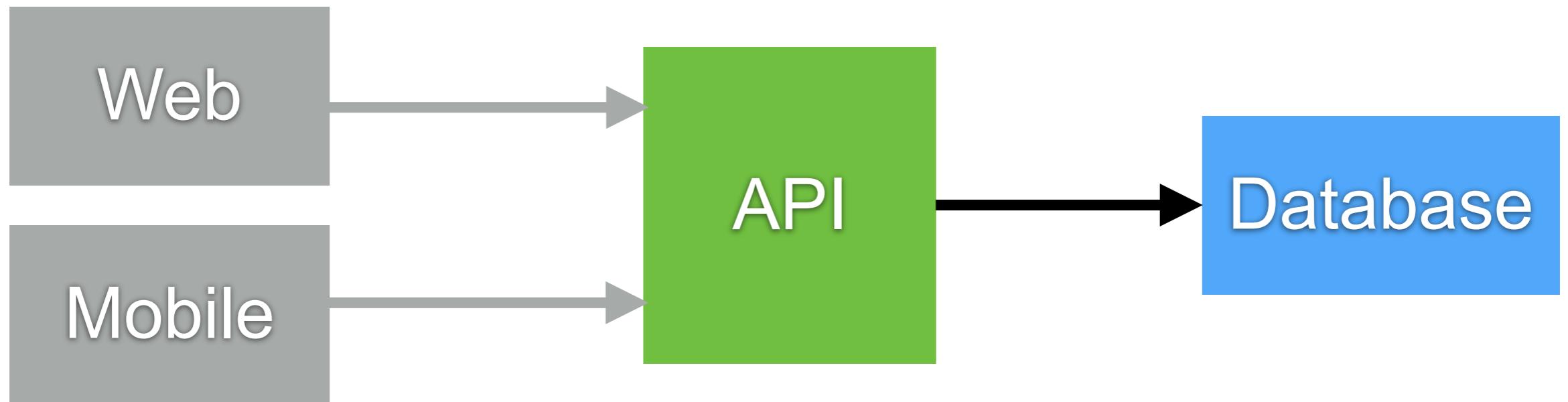
UI Test ?



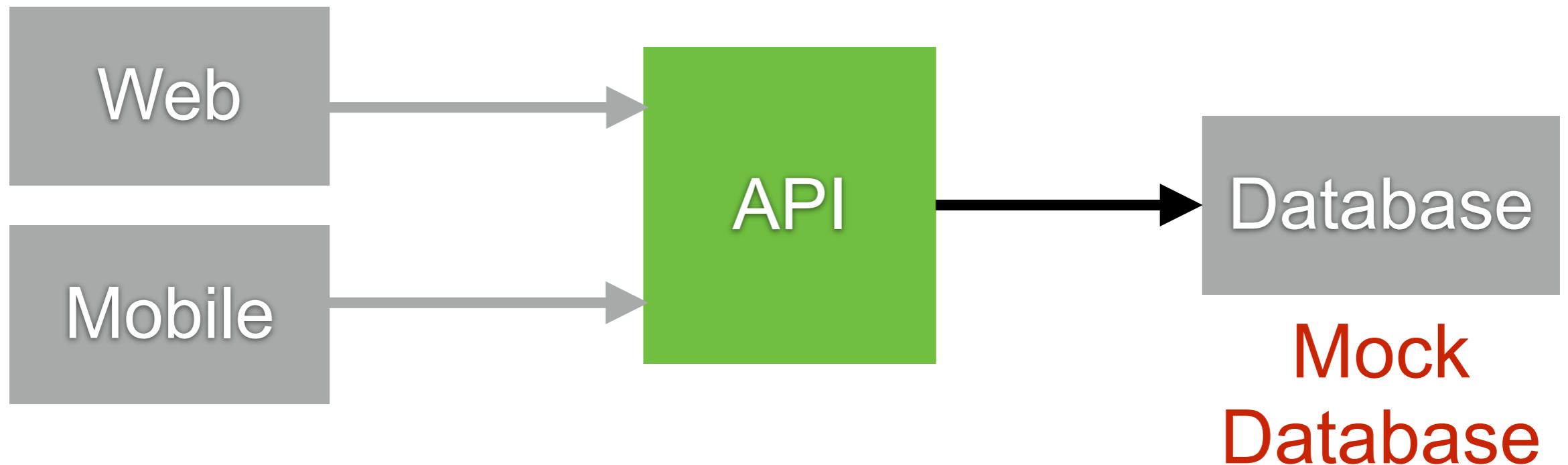
UI Test ?



API Testing ?



API Testing ?



Q/A

