



HELP CONTENTS

- About SwaggerHub
- Creating a SwaggerHub Account
- ▼ Tutorials
 - Getting Started With SwaggerHub
 - Collaboration on SwaggerHub
 - Swagger 2.0 Tutorial
 - OpenAPI 3.0 Tutorial
 - OpenAPI/Swagger How-To
- ▶ UI Overview
- ▶ APIs
- ▶ Domains
- ▶ Collaboration
- ▶ Organizations, Projects, and Teams
- ▶ Integrations
- ▶ Account Management
- ▶ Plans and Billing
- Email Notifications
- OpenAPI 3.0 Support
- ▶ On-Premise Admin Guide

Getting Started With SwaggerHub

This tutorial provides a brief overview of SwaggerHub, and walks you through creating and publishing a REST API specification.

In This Tutorial

- ⇒ What is SwaggerHub?
- ⇒ What is OpenAPI?
- ⇒ What is Swagger?
- ⇒ SwaggerHub UI overview
- ⇒ Creating a sample API with SwaggerHub
 - Step 1. Create an API
 - Step 2. Add server info
 - Step 3. Define a GET operation
 - Step 4. Define parameters
 - Step 5. Define responses
 - Step 6. Publish the API
 - Other things to do in the SwaggerHub editor
- ⇒ What's next

Requirements

To complete this tutorial, you need:

- A SwaggerHub account (sign up (/signup), if you do not have it).
- Basic knowledge of REST APIs.

What is SwaggerHub?

SwaggerHub is an online platform where you can design your REST APIs – be it public APIs, internal private APIs or microservices. The principle behind SwaggerHub is **Design First, Code Later**. That is, you start by laying out your API, its resources, operations and endpoints. Once the design is complete you implement the business logic.



(7)

API definitions are written in the OpenAPI format (formerly known as Swagger). They are saved in the SwaggerHub cloud and synchronized with external systems like GitHub or Amazon API Gateway. You can also collaborate with your team on Swagger multiple versions of APIs as it evolves.

SIGN UP

LOG IN

What is OpenAPI?

OpenAPI Specification (<https://github.com/OAI/OpenAPI-Specification>) (formerly known as Swagger Specification) is an API for REST APIs. It is equally suitable both for designing new APIs and for documenting your existing APIs. OpenAPI lets you describe an API, including the available endpoints, operations, request and response formats, supported authentication methods, support information.

OpenAPI Specification has undergone several versions since the original release. SwaggerHub supports OpenAPI 3.0.0 (the latest version) and OpenAPI 2.0 (also known as Swagger 2.0).

OpenAPI definitions can be written in YAML or JSON. SwaggerHub Editor uses YAML, but you can import and download both formats.

A sample OpenAPI 3.0 specification (in the YAML format) looks like this:

```
openapi: 3.0.0
info:
  title: Sample API
  version: '1.0'
servers:
  - url: https://api.example.com/v1
paths:
  /hello:
    get:
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                type: string
```

This simple format is human-readable, machine-readable and self-explanatory. This is one of the reasons why OpenAPI is so popular among developers.

What is Swagger?

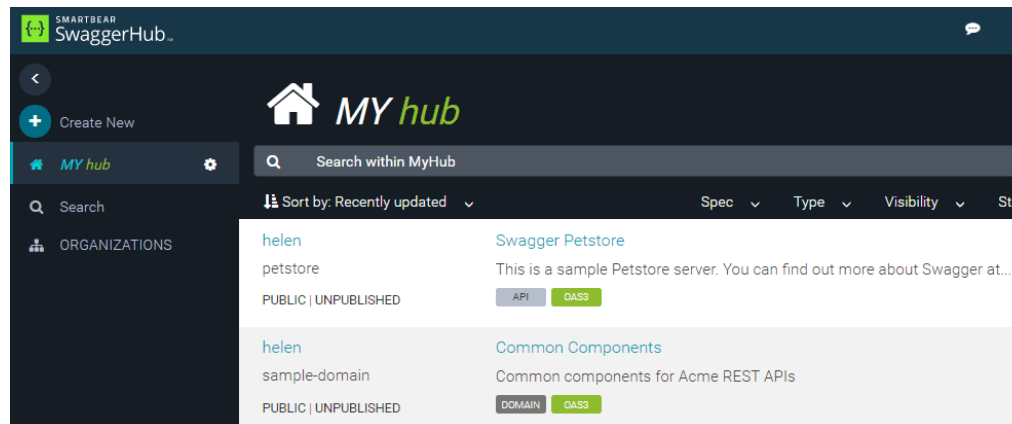
The great thing about OpenAPI Specification is that there are a lot of API development tools that support it. An example is Swagger, a set of open-source tools for designing, building, documenting and consuming REST APIs. The major Swagger tools include:

- Swagger Editor – a browser-based editor where you can write OpenAPI definitions.
- Swagger UI – renders OpenAPI definitions as interactive API documentation, like the one at <http://petstore.swagger.io> (<http://petstore.swagger.io>).
- Swagger Codegen – generates server code and client libraries based on an OpenAPI definition.

SwaggerHub combines these tools for a seamless experience, and adds hosted storage, access control and collaboration features.


SwaggerHub UI overview

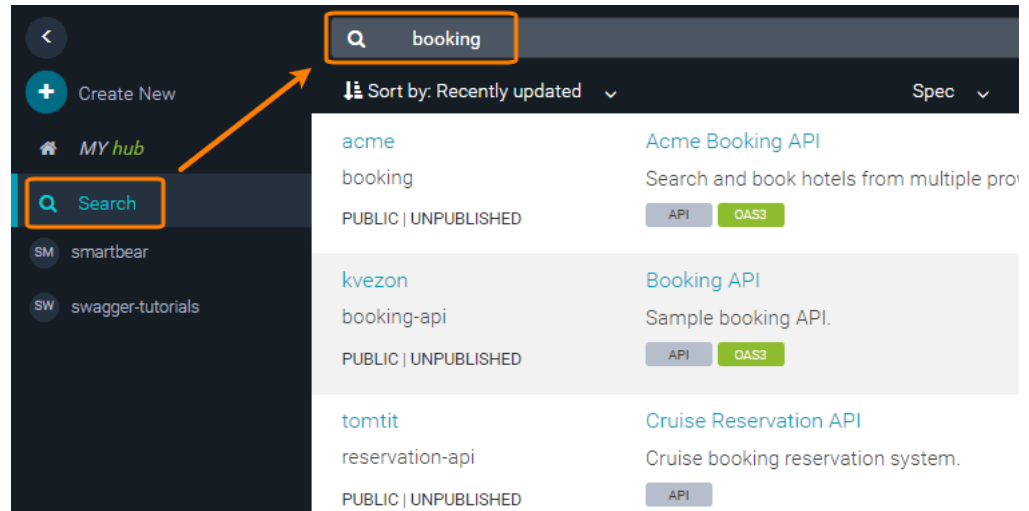
When you are logged in to SwaggerHub, the **MY hub** page lists all the APIs you have access to. These are both the APIs you created and the APIs you were invited to collaborate on. If you are new to SwaggerHub and have no APIs yet, the list will be empty, but it will encourage you to start creating APIs.



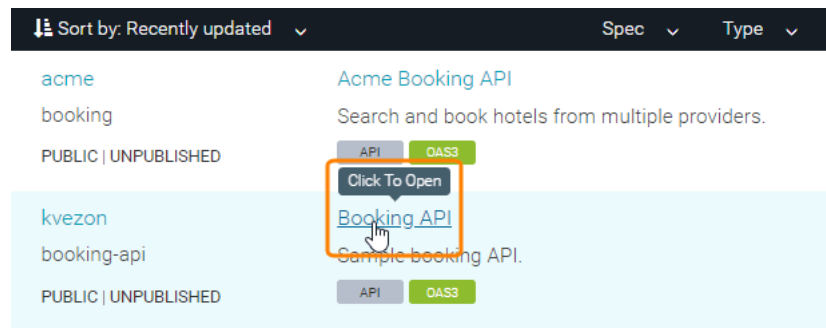


(1)

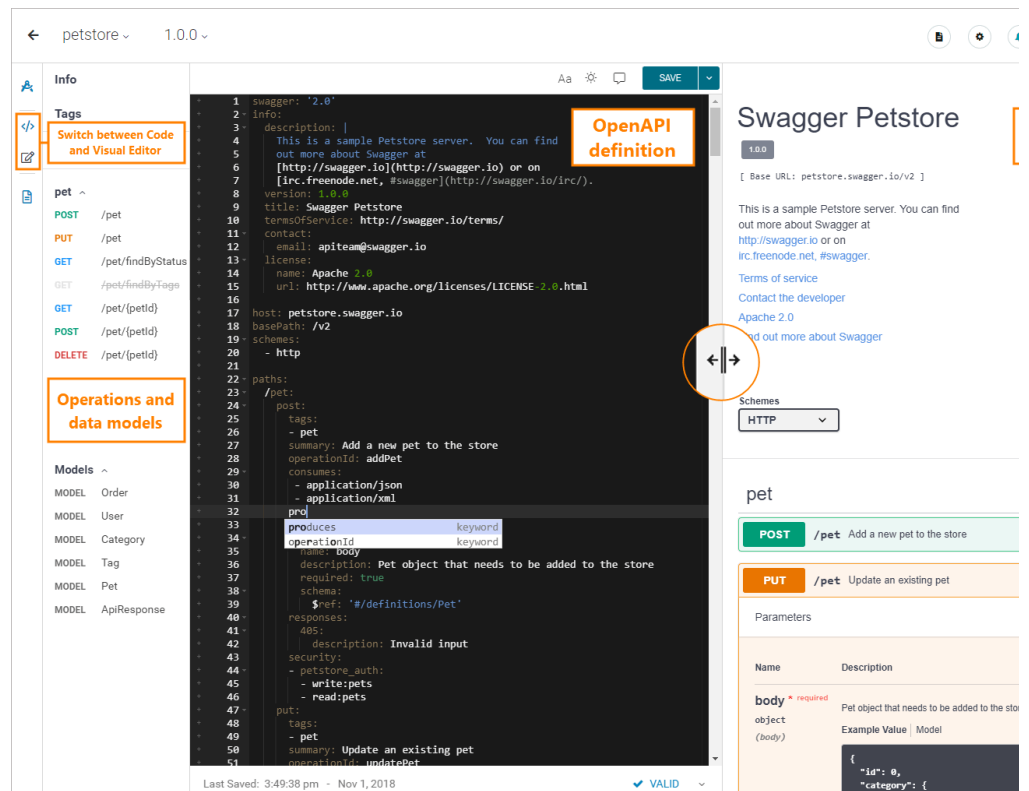
To search for existing APIs on SwaggerHub, use the search page ([../ui/searching](#)) that you can access by clicking  in the sidebar. This way you can find some great public APIs developed by other SwaggerHub users. See [Searching SwaggerHub](#) ([../ui/searching-syntax](#)) for search syntax.



To view a specific API, click it in the list.



An API page on SwaggerHub is a split view that shows the YAML code of your OpenAPI definition and beautiful reference-style documentation generated from it. The API documentation allows the users to test API calls directly in the browser. The navigation left shows a list of operations and models defined in your API and lets you jump to the corresponding place in the YAML code panels by dragging the splitter between them.

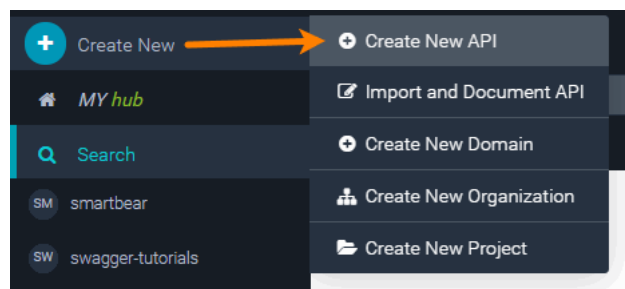


Creating a sample API with SwaggerHub

The best way to understand SwaggerHub is to create a sample API, so let's do that. Do not worry about configuring a server. will focus on designing an API.

Step 1. Create an API

In the sidebar on the left, click  and select **Create New API**.

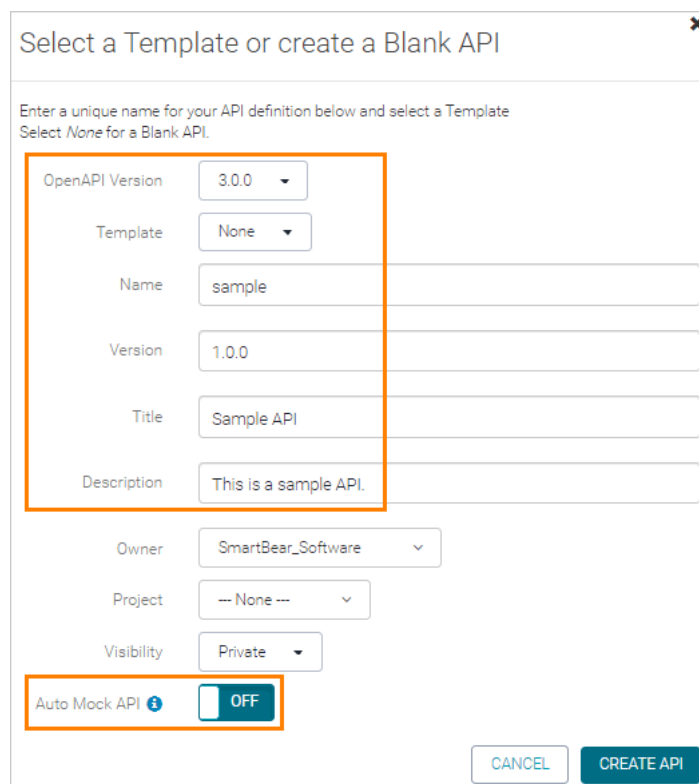


Fill in the API information as shown in the image below.

The OpenAPI version specifies the spec format, OpenAPI 3.0.0 or 2.0. Choose *3.0.0* (the latest version).

There are predefined templates, such as *Petstore*, but now, let's start with a blank API (no template).

Enter *sample* as the API name, *1.0.0* as the version, *Sample API* as the title and an arbitrary description.

A screenshot of the 'Select a Template or create a Blank API' form. The form has a title bar with a close button. Below the title, there's a prompt: 'Enter a unique name for your API definition below and select a Template. Select None for a Blank API.' The form contains several fields: 'OpenAPI Version' (dropdown set to 3.0.0), 'Template' (dropdown set to None), 'Name' (text input with 'sample'), 'Version' (text input with '1.0.0'), 'Title' (text input with 'Sample API'), 'Description' (text input with 'This is a sample API.'). Below these are 'Owner' (dropdown set to SmartBear_Software), 'Project' (dropdown set to -- None ---), and 'Visibility' (dropdown set to Private). At the bottom, there's an 'Auto Mock API' toggle switch set to 'OFF'. 'CANCEL' and 'CREATE API' buttons are at the bottom right. Orange boxes highlight the 'OpenAPI Version', 'Template', 'Name', 'Version', 'Title', 'Description' fields and the 'Auto Mock API' toggle.

Click **Create API**. The SwaggerHub editor will open, pre-populated with the API metadata you have entered:



()

The screenshot shows the SwaggerHub editor for a 'Sample API' (version 1.0.0). The left sidebar contains navigation links for Info, Tags, Servers, and a search bar. The main editor area displays the following OpenAPI 3.0.0 definition:

```

1 openapi: 3.0.0
2 info:
3   version: '1.0.0'
4   title: 'Sample API'
5   description: 'This is a sample API.'
6   paths: {}

```

Below the editor, it indicates 'Last Saved: 6:48:06 pm - Oct 4, 2019' and 'VALID'. The right sidebar shows the 'Sample API' title, version '1.0.0', and a note: 'This is a sample API. Note: "Try it out" is disabled because no servers are defined in the "servers" array. Please see: Info on OAS3 servers'. At the bottom, it states 'No operations defined in spec!' and 'SSL certificates are validated'. The footer mentions 'Routing requests via browser'.

Each API definition starts with the OpenAPI version, in our example it is **openapi: 3.0.0**.

```
openapi: 3.0.0
```

The next section, **info**, contains the metadata about the API – the API title, description, version and so on. The **info** section also contains contact information, license and other details.

```

info:
  title: Sample API
  version: 1.0.0
  description: This is a sample API.

```

The **paths** section is where you define the API operations. We will populate it later.

```
paths: {}
```

Step 2. Add server info

Next, we need to define the API server address and the base URL for API calls. Suppose the API will be located at **https://api.example.com/v1**. This can be described as:

```

servers:
  - url: https://api.example.com/v1

```

Add these lines between **info** and **paths**, like this:

```

1 openapi: 3.0.0
2 info:
3   version: '1.0.0'
4   title: 'Sample API'
5   description: 'This is a sample API.'
6
7 servers:
8   - url: https://api.example.com/v1
9
10 paths: {}

```

Step 3. Define a GET operation

Suppose our API is intended to manage a list of users. Each user has an ID, and the API needs an operation to return the user details. This operation would be a GET request like this one:

```

GET /users/3
GET /users/15

```

Using Swagger, we can describe this GET operation as follows. The **{userId}** part in **/users/{userId}** indicates the path is a URL component that can vary.

```

paths:
  /users/{userId}:
    get:
      summary: Returns a user by ID
      responses:
        '200':
          description: OK

```

Paste the code above to the editor, replacing the **paths: {}** line. Your spec should look like this:



()

```

1 openapi: 3.0.0
2 info:
3   version: '1.0.0'
4   title: 'Sample API'
5   description: 'This is a sample API.'
6 servers:
7   - url: 'https://api.example.com/v1'
8
9 paths:
10  /users/{userId}:
11    get:
12      summary: Returns a user by ID.
13      responses:
14        '200':
15          description: OK

```

Sample API SIGN UP LOG IN

1.0.0 OAS3

This is a sample API.

Servers

https://api.example.com/v1

default

GET /users/{userId} Returns a user by ID.

Parameters

No parameters

Responses

Code	Description
200	OK

Last Saved: 6:48:06 pm - Oct 4, 2019 1 Error

Note that as you edit the spec, SwaggerHub automatically updates the preview on the right.

Step 4. Define parameters

Our API operation `GET /users/{userId}` has the `userId` parameter that specifies the ID of the user to return. To define it in your spec, add the `parameter` section as follows:

```

paths:
  /users/{userId}:
    get:
      summary: Returns a user by ID
      parameters:
        - name: userId
          in: path
          required: true
          description: The ID of the user to return
          schema:
            type: integer
      responses:
        '200':
          description: OK

```

This code specifies the parameter `name` (the same as the one used in the URL path), `type`, `description` and whether it is a `path` means the parameter is passed as part of the URL path (`GET /users/15`), as opposed to, say, query string parameter `id=15`).

Note that after you add the `parameters` section, the preview is updated to reflect the newly added parameter information:



()

```

1 openapi: 3.0.0
2 info:
3   version: '1.0.0'
4   title: 'Sample API'
5   description: 'This is a sample API.'
6 servers:
7   - url: https://api.example.com/v1
8
9 paths:
10  /users/{userId}:
11    get:
12      parameters:
13        - name: userId
14          in: path
15          required: true
16          description: The ID of the user to return.
17          schema:
18            type: integer
19
20      summary: Returns a user by ID.
21      responses:
22        '200':
23          description: OK
24

```

Servers

https://api.example.com/v1

SIGN UP

LOG IN

default

GET /users/{userId} Returns a user by ID.

Parameters

Name	Description
userId * required integer (path)	The ID of the user to return. userId - The ID of the user to return.

Responses

Code	Description
200	OK

Last Saved: 6:48:06 pm - Oct 4, 2019

VALID

Step 5. Define responses

Next, we need to describe possible **responses** for the API call. A response is defined by an HTTP status code, description a **schema** (a data model for the response body, if any).

Suppose a successful call to **GET /users/{userId}** returns HTTP status 200 and this JSON object:

```

{
  "id": 42,
  "name": "Arthur Dent"
}

```

To describe this response, add the **content** section under the 200 response as follows. The **content** section specifies the contains JSON data (**application/json**). Note the **schema** element – it describes the structure of the response body, such of the returned JSON object.

```

paths:
  /users/{userId}:
    get:
      ...
      responses:
        '200':
          description: Successful response
          content:
            application/json:
              schema:
                type: object
                properties:
                  id:
                    type: integer
                  name:
                    type: string

```

You can also describe various error codes that can be returned by an API call:

```

responses:
  ...
  '400':
    description: The specified user ID is invalid (e.g. not a number)
  '404':
    description: A user with the specified ID was not found

```

The complete spec should now look like this:




(1)

The screenshot shows the SwaggerHub editor interface. On the left, the OpenAPI definition is displayed in a dark-themed code editor. The definition is for a sample API with version 1.0.0. The endpoint `/users/{userId}` has a GET method. The summary is 'Returns a user by ID.'. The parameters section defines a required path parameter `userId` of type `integer`. The responses section is highlighted with an orange box and contains two entries: a 200 response with a description 'Successful response' and a JSON schema for an object with properties `id` (integer) and `name` (string); and a 404 response with a description 'A user with the specified ID was not found'. An orange arrow points from the response section to the right-hand panel.

The right-hand panel shows the visual representation of the API definition. It includes a 'Parameters' section with a table listing the `userId` parameter as a required integer. Below this, the 'Responses' section shows a table with two entries: a 200 response with a description 'Successful response' and a 404 response with a description 'A user with the specified ID was not found'. The 200 response is expanded to show its media type as `application/json` and its schema as an object with properties `id` and `name`.

Step 6. Publish the API

Our API is finished, so let's publish it. Publishing is a way to tell people that the API is in a stable state, it is going to work as designed and can be consumed by applications. Do not worry about configuring a server – this tutorial is about designing the API specification implementation.

To publish the API, click the drop-down arrow next to your API version, and then click .


The screenshot shows the SwaggerHub editor interface. The API version is 1.0.0. The 'Publish API' button is highlighted with an orange arrow. The button is labeled 'Publish API' and 'UBLISHED'. Below the button, there is a section for 'Add New Version'.

Congratulations! You have published your first API on SwaggerHub! You can copy its address from the browser's address bar and share it with others. Since this is a public API, anyone who has a link can view it, and it will show up in the search results on SwaggerHub. API private (`../apis/public-and-private-apis`) if your SwaggerHub plan allows this.

Note that published APIs become read-only, and can only be edited if the API is unpublished again. It is OK to unpublish an API to need to improve the **description** text or fix typos. But for breaking changes like new operations or parameters, you should instead be using the **Add Version** command in the SwaggerHub editor. SwaggerHub lets you maintain multiple versions of an API so you can work on the next API version while keeping the published version (the "production" version) intact.

Other things to do in the SwaggerHub editor

SwaggerHub editor is not just an editor, it provides tools for managing your API specification and integrating it into your workflow. Take a quick look at other available commands:

- The  **Settings** menu lets you rename the API, fork it, or transfer it to another owner.
- The **Export** menu lets you generate server (`../apis/generating-code/server-stub`) and client (`../apis/generating-code/client`) code for your API to help you get started with implementation. Here you can also download your OpenAPI definition as YAML or JSON.

What's next

Learn OpenAPI 3.0 Syntax



(/)

- [Tutorial \(openapi-3-tutorial\)](#)
- [Syntax Guide \(https://swagger.io/docs/specification/\)](https://swagger.io/docs/specification/)
- [Full OpenAPI 3.0 Specification \(https://swagger.io/specification/\)](https://swagger.io/specification/)

[SIGN UP](#)[LOG IN](#)

Learn OpenAPI 2.0 (Swagger 2.0) Syntax

- [Tutorial \(writing-swagger-definitions\)](#)
- [Syntax Guide \(https://swagger.io/docs/specification/2-0/basic-structure/\)](https://swagger.io/docs/specification/2-0/basic-structure/)
- [Full OpenAPI 2.0 Specification \(https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md\)](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md) (Swagger)

Proceed With API Design and Implementation

- [Invite collaborators \(collaboration\)](#)
- [Sync your API definitions with GitHub \(../integrations/github-sync\)](#), [GitLab \(../integrations/gitlab-sync\)](#), [Bitbucket \(../integrations/bitbucket-sync\)](#), or [Azure DevOps Services \(../integrations/azure-devops-services\)](#)
- [Mock your API \(../integrations/api-auto-mocking\)](#) (if you do not have a live API server yet)
- [Generate server stub \(../apis/generating-code/server-stub\)](#) and [client SDK \(../apis/generating-code/client-sdk\)](#)

See Also

[SwaggerHub Tutorials \(index\)](#)[SwaggerHub UI Overview \(../ui/overview\)](#)

© 2020 SmartBear Software