*Swagger*

|  |  |  |
| --- | --- | --- |
| S.No | Contents | Page No |
| 1 | Open API Specification Introduction | 2 |
| 2 | Getting started with Swagger | 3 |
| 3 | Swagger Core Components | 4 |
| 4 | Swagger Editor | 5 |
| 5 | Swagger UI | 6 |
| 6 | Swagger code Gen | 7 |
| 7 | Application Execution | 9 |

**Open API Specification**

The goal of The OpenAPI Specification is to define a standard, language-agnostic interface to REST 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 via OpenAPI, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Similar to what interfaces have done for lower-level programming, OpenAPI removes the guesswork in calling the service.

Use cases for machine-readable API interfaces include interactive documentation, code generation for documentation, client, and server, as well as automated test cases. OpenAPI-enabled APIs expose JSON files that correctly adhere to the OpenAPI Specification, documented in this repository. These files can either be produced and served statically, or be generated dynamically from your application.

Without going into a long history of interfaces to Web Services, this is not the first attempt to do so. We can learn from CORBA, WSDL and WADL. These specifications had good intentions but were limited by proprietary vendor-specific implementations, being bound to a specific programming language, and goals which were too open-ended. In the end, they failed to gain traction.

OpenAPI does not require you to rewrite your existing API. It does not require binding any software to a service--the service being described may not even be yours. It does, however, require the capabilities of the service be described in the structure of the OpenAPI Specification. Not all services can be described by OpenAPI--this specification is not intended to cover every possible use-case of a REST-ful API. OpenAPI does not define a specific development process such as design-first or code-first. It does facilitate either technique by establishing clear interactions with a REST API.

Currently in market Swagger is the popular in all Open API Specification Technologies.

**Getting Started with Swagger**

With the advancement of web technologies and different ways to implement them, there have been many different schools of thought about how to make it easy for end users and to address different challenges into the space of web. With this the inception of web-services proliferated majority into two prime categories REST and SOAP. Both of them are common way to expose web services, where SOAP has the significant contractual obligation between service consumer and service producer. It standardizes the request response structure through mutual understanding and adheres the contracts. But with Restful services the story is different. Describing to users how to use and interact with a REST API is a daunting task.

There are no set standards to expose a REST contract other than HTTP method attributes governance. Many APIs resort to manually edited human-readable documentation, which is hard to maintain from a synchronization standpoint with the API. Hence, defining the standards for your REST services, and keeping the documentation updated in real time is a big challenge.

Swagger is one of the most popular specifications for REST APIs for a number of reasons:

* Swagger generates an interactive API console for people to quickly learn about and try the API.
* Swagger generates the client SDK code needed for implementations on various platforms.
* The Swagger file can be auto-generated from code annotations on a lot of different platforms.
* Swagger has a strong community with helpful contributors.

The Swagger spec provides a way to describe your API using a specific JSON or YAML schema that outlines the names, order, and other details of the API.

You can code this Swagger file by hand in a text editor, or you can auto-generate it from annotations in your source code. Different tools can consume the Swagger file to generate the interactive API documentation.

**What is Swagger?**

The goal of Swagger™ is to define a standard, language-agnostic interface to REST 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 via Swagger, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Similar to what interfaces have done for lower-level programming, Swagger removes the guesswork in calling the service.

Technically speaking - Swagger is a [formal specification](http://swagger.io/getting-started/specification) surrounded by a large ecosystem of [tools](http://swagger.io/tools), which includes everything from front-end user interfaces, low-level code libraries and commercial API management solutions.

***Advantages***

* With the Swagger framework, the server, client and documentation team can be in synchronization simultaneously.
* As Swagger is a language-agnostic specification, with its declarative resource specification, clients can easily understand and consume services without any prior knowledge of server implementation or access to the server code.
* The Swagger UI framework allows both implementers and users to interact with the API. It gives clear insight into how the API responds to parameters and options.
* Swagger responses are in JSON and XML, with additional formats in progress.
* Swagger implementations are available for various technologies like Scala, Java, and HTML5.
* Client generators are currently available for Scala, Java, JavaScript, Ruby, PHP, and ActionScript 3, with more client support underway.

***Swagger Approaches:***

***A top-down approach*** where you would use the [Swagger Editor](http://editor.swagger.io/) to create your Swagger definition and then use the integrated [Swagger Codegen](http://swagger.io/getting-started/swagger-codegen) tools to generate server implementation.

***A bottom-up approach*** where you have an existing REST API for which you want to create a Swagger definition. Either you create the definition manually (using the same Swagger Editor mentioned above),

**Swagger Core Components**

The Swagger framework is supported by a set of core tools for designing, building, and documenting Restful APIs. All of these tools are free and open source projects available on GitHub.

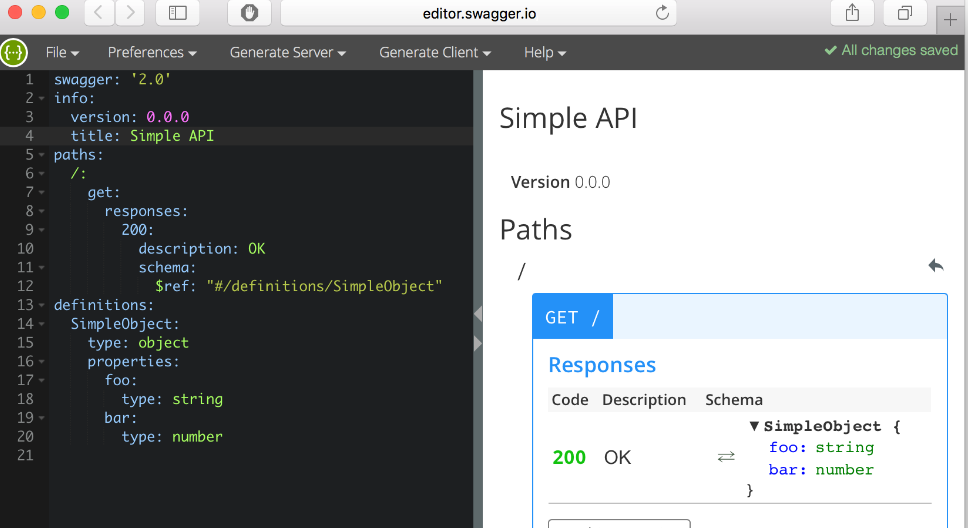
1. Swagger Editor.

2. Swagger Codegen.

3.Swagger UI

# Swagger Editor

Swagger Editor lets you edit [Swagger API specifications](https://github.com/swagger-api/swagger-spec/blob/master/versions/2.0.md) in YAML inside your browser and to preview documentations in real time. Valid Swagger JSON descriptions can then be generated and used with the full Swagger tooling (code generation, documentation, etc).



Local installation:

Make sure you have [Node.js](http://nodejs.org/) installed.

git clone https://github.com/swagger-api/swagger-editor.git

cd swagger-editor

npm install

npm start

Online access: <http://editor.swagger.io/#!/>

Features:

* The Editor works in any development environment, be it locally or in the web
* Easy to configure and customize anything, from line-spacing to themes.
* Render your API specification visually and interact with your API while still defining it
* Generate server stubs and client libraries for your API in every popular language.
* Write syntax faster with a smart and intelligent auto-completion.

# Swagger UI

Swagger UI allows anyone — be it your development team or your end consumers — to visualize and interact with the API’s resources without having any of the implementation logic in place. It’s automatically generated from your Swagger specification, with the visual documentation making it easy for back end implementation and client side consumption.

Features:

* Host your Swagger UI in any environment.
* Allow end developers to effortlessly interact and try out every single operation your API exposes for easy consumption.
* Quickly find and work with resources and endpoints with neatly categorized documentation.
* Cater to every possible scenario with Swagger UI working in all major browsers.
* Style and tweak your Swagger UI the way you want with full source code access.

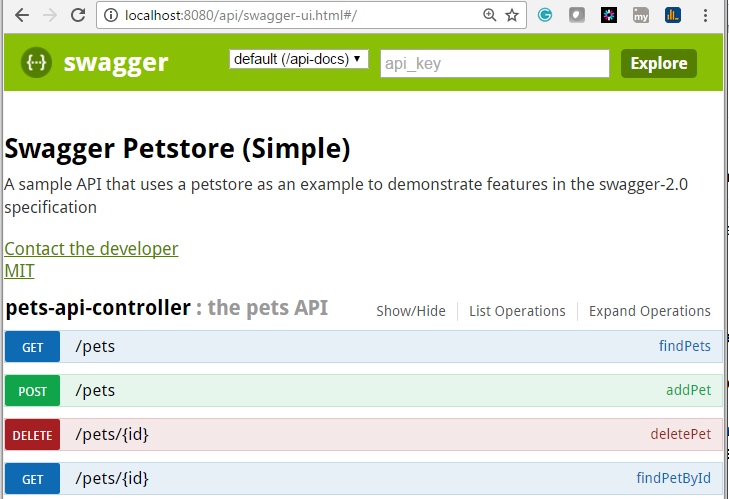


Fig: Sample Pet Store API documentation.

# Swagger Codegen

Build APIs quicker and improve consumption of your Swagger-defined APIs in every popular language with Swagger Codegen. Swagger Codegen can simplify your build process by generating server stubs and client SDKs from your Swagger specification, so your team can focus better on your API’s implementation and adoption.

Features:

* Remove tedious plumbing and configuration by generating boilerplate server code in over 20 different languages.
* Generate client SDKs in over 40 different languages for end developers to easily integrate with your API.
* Swagger Codegen is always updated with the latest and greatest changes in the programming world.

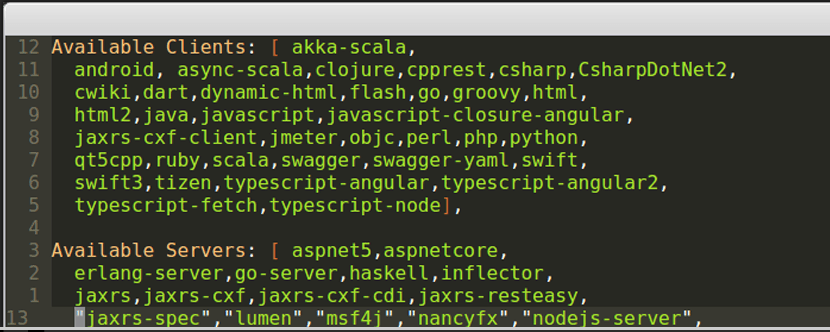
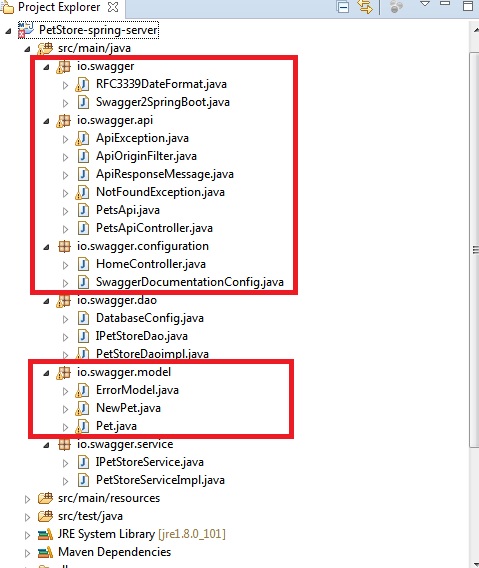


Fig: List of available technologies in which Swagger codegen supports.

*Steps to Follow:*

1. Opened Swagger Editor.
2. Open Existing Pet Store API in yaml language.
3. Generate server stub in spring and downloaded the same.
4. Import the downloaded maven project in to eclipse.



*Fig: Pet Store API Project Structure generated by swagger Codegen module.*

In the Above project Structure, the highlighted packages are swagger generated and the rest of two packages.

1. Io.swagger.dao - Dao logic
2. Io.swagger.service - Service logic are Custom written files by developer.

If we are trying to use hibernate as ORM tool ,Supposed to include hibernate related properties in side hibernate. Properties file.

Modify the POM.xml with below dependencies for data base operations.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

**Application Execution**

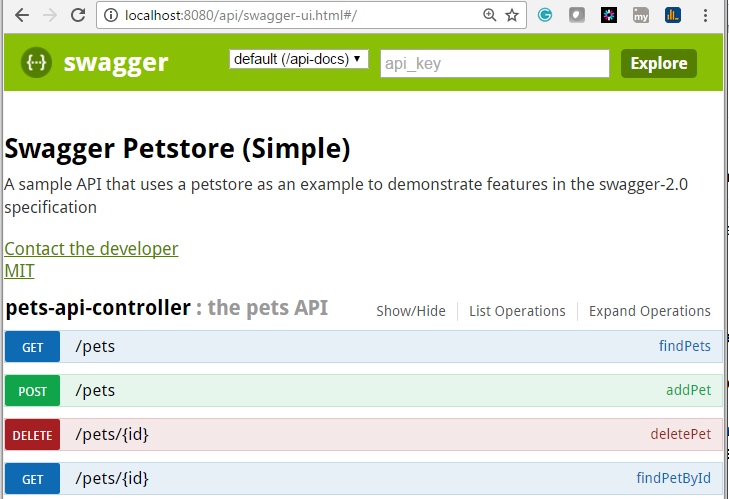
This project has been built on top of the spring boot technology, so it has in build embedded Application server (Apache/jetty).

*Steps involved:*

1. Execute io.swagger. Swagger2SpringBoot.java file , Spring boot Application will start now.
2. Open application.properties check the value for key server.contextPath .

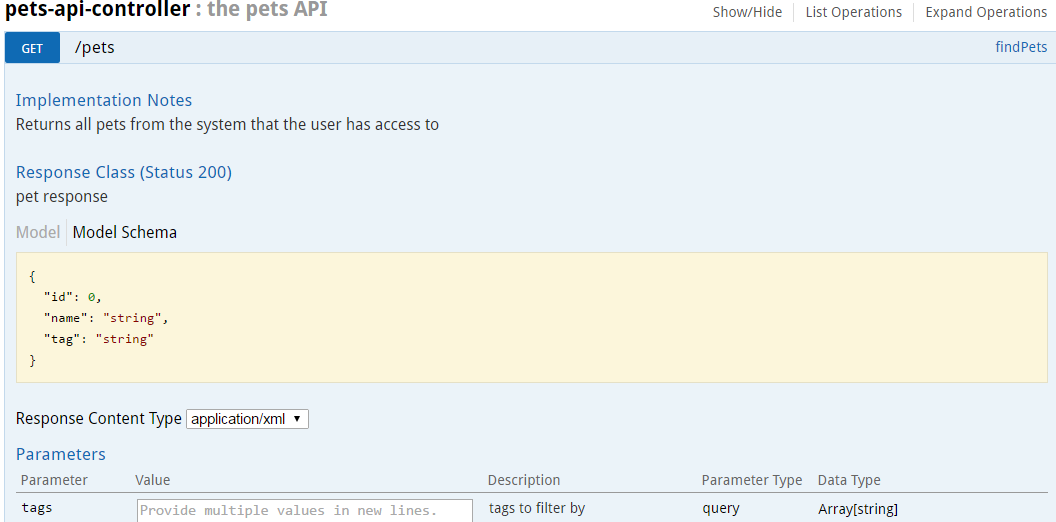
This is the base path for the application defined in the yaml file. our case its (/api)

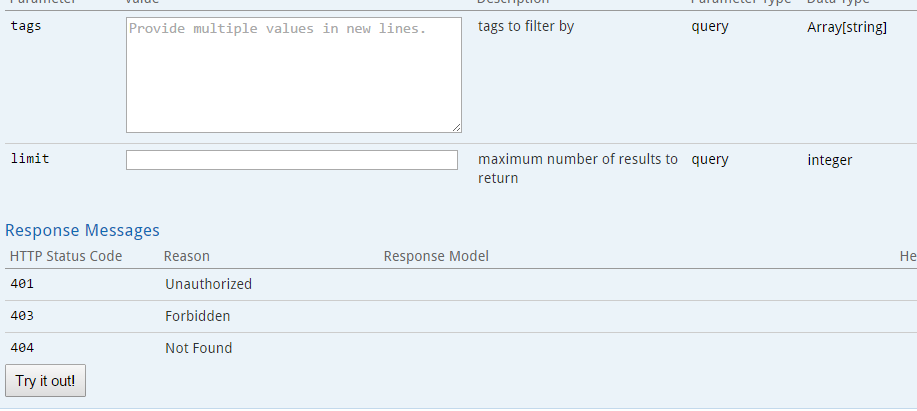
1. Open browser <http://localhost:8080/api>
2. Browser will navigate to <http://localhost:8080/api/swagger-ui.html#/> and Swagger Application will launch with REST API Documentation with End points.



1. Select any end point and check the REST Service documentation for the corresponding Http method , by default Swagger gives Response code 200 for every service.
2. We are supposed to customize that code based on our purpose in this below file.

Io.swagger.api.PetsAPiController.java





*Figure: Sample Dash board For single Rest End point.*

Try it Out Button gives the sample response including (Response body, Response code ,Request URL, Request Headers,Response Headers)for the corresponding end point.

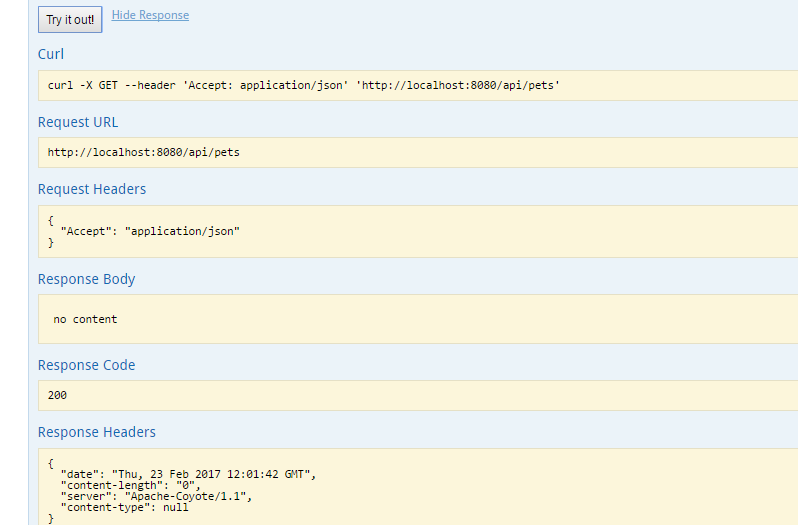


Fig : Sample response when click on Try it Out Button.