



API Developmentfor Everyone

Cos' è swagger?

Swagger è uno strumento open source e professionale per aiutare sviluppatori, team e organizzazioni a fornire API eccezionali.

Fornisce gli strumenti necessari per:

- Creazione delle API attraverso un file yml o json utilizzando le specifiche OAS 2.0 & OAS 3.0.
- Documentazione delle API.
- Generazione SERVER e CLIENT in diversi linguaggi.
- Visualizzazione e test delle API attraverso un'interfaccia grafica.

OpenAPI (OAS)

La specifica OpenAPI (OAS) definisce un'interfaccia standard, indipendente dalla lingua, per le API RESTful che consente sia alle persone che ai computer di scoprire e comprendere le capacità del servizio senza accesso al codice sorgente, alla documentazione o tramite l'ispezione del traffico di rete. Se definito correttamente, un consumatore può comprendere e interagire con il servizio remoto con una quantità minima di logica di implementazione.

Una definizione OpenAPI può quindi essere utilizzata dagli strumenti di generazione della documentazione per visualizzare l'API, strumenti di generazione di codice per generare server e client in vari linguaggi di programmazione, strumenti di test e molti altri casi d'uso.

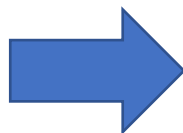
Swagger

- Come posso progettare le mie api?
- Come posso validare le mie api?
- Come posso versionare e manterne le modifiche?
- Come posso condividere le specifiche delle mie api?

Swagger

Progetta, descrivi e documenta le tue API attraverso un file yml o json. basate su OpenAPI.

Esempio di file yml



```
1  swagger: "2.0"
2  info:
3    description: "This is a sample server Petstore server.  You can find out more about Swagger at
      [http://swagger.io](http://swagger.io) or on [irc.freenode.net, #swagger](http://swagger.io/irc/).
      For this sample, you can use the api key `special-key` to test the authorization filters."
4    version: "1.0.0"
5    title: "Swagger Petstore"
6    termsOfService: "http://swagger.io/terms/"
7    contact:
8      email: "apiteam@swagger.io"
9    license:
10      name: "Apache 2.0"
11      url: "http://www.apache.org/licenses/LICENSE-2.0.html"
12  host: "petstore.swagger.io"
13  basePath: "/v2"
14  tags:
15  - name: "pet"
16    description: "Everything about your Pets"
17    externalDocs:
18      description: "Find out more"
19      url: "http://swagger.io"
20  - name: "store"
21    description: "Access to Petstore orders"
22  - name: "user"
23    description: "Operations about user"
24    externalDocs:
25      description: "Find out more about our store"
26      url: "http://swagger.io"
27  schemes:
28  - "https"
29  - "http"
30  paths:
31  - /pet:
32    post:
33      tags:
34      - "pet"
35      summary: "Add a new pet to the store"
36      description: ""
37      operationId: "addPet"
38      consumes:
39      - "application/json"
40      - "application/xml"
41      produces:
```

Swagger tools

- Swagger Editor – Design
- Swagger CodeGen – Generazione Server e Client
- Swagger UI - Documentazione

Swagger Editor

- Autocompletamento del codice
- Controllo degli errori
- Preview

The image shows the Swagger Editor interface. The left pane displays the Swagger Petstore API definition in JSON format, with line numbers 1 through 41. The right pane shows the rendered preview of the API, titled "Swagger Petstore 1.0.0". The preview includes a description of the sample server, links for terms of service, contact, and finding out more about Swagger, and a list of API endpoints with their methods and descriptions.

```
1 swagger: "2.0"
2 info:
3   description: "This is a sample server Petstore server. You can find out more about Swagger at
4     [http://swagger.io](http://swagger.io) or on [irc.freenode.net, #swagger](http://swagger.io/irc/).
5     For this sample, you can use the api key 'special-key' to test the authorization filters."
6   version: "1.0.0"
7   title: "Swagger Petstore"
8   termsOfService: "http://swagger.io/terms/"
9   contact:
10     email: "apiteam@swagger.io"
11   license:
12     name: "Apache 2.0"
13     url: "http://www.apache.org/licenses/LICENSE-2.0.html"
14 host: "petstore.swagger.io"
15 basePath: "/v2"
16 tags:
17   - name: "pet"
18     description: "Everything about your Pets"
19     externalDocs:
20       description: "Find out more"
21       url: "http://swagger.io"
22   - name: "store"
23     description: "Access to Petstore orders"
24   - name: "user"
25     description: "Operations about user"
26     externalDocs:
27       description: "Find out more about our store"
28       url: "http://swagger.io"
29 schemes:
30   - "https"
31   - "http"
32 paths:
33   /pet:
34     post:
35       tags:
36       - "pet"
37       summary: "Add a new pet to the store"
38       description: ""
39       operationId: "addPet"
40       consumes:
41       - "application/json"
42       - "application/xml"
43       produces:
```

Swagger Petstore 1.0.0
[Base URL: petstore.swagger.io/v2]

This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on <irc.freenode.net, #swagger>. For this sample, you can use the api key **special-key** to test the authorization filters.

[Terms of service](#)
[Contact the developer](#)
[Apache 2.0](#)
[Find out more about Swagger](#)

Schemes: **HTTPS** [Authorize](#)

pet Everything about your Pets [Find out more](#)

- POST** /pet Add a new pet to the store
- PUT** /pet Update an existing pet
- GET** /pet/findByStatus Finds Pets by status
- GET** /pet/findByTags Finds Pets by tags
- GET** /pet/{petId} Find pet by ID

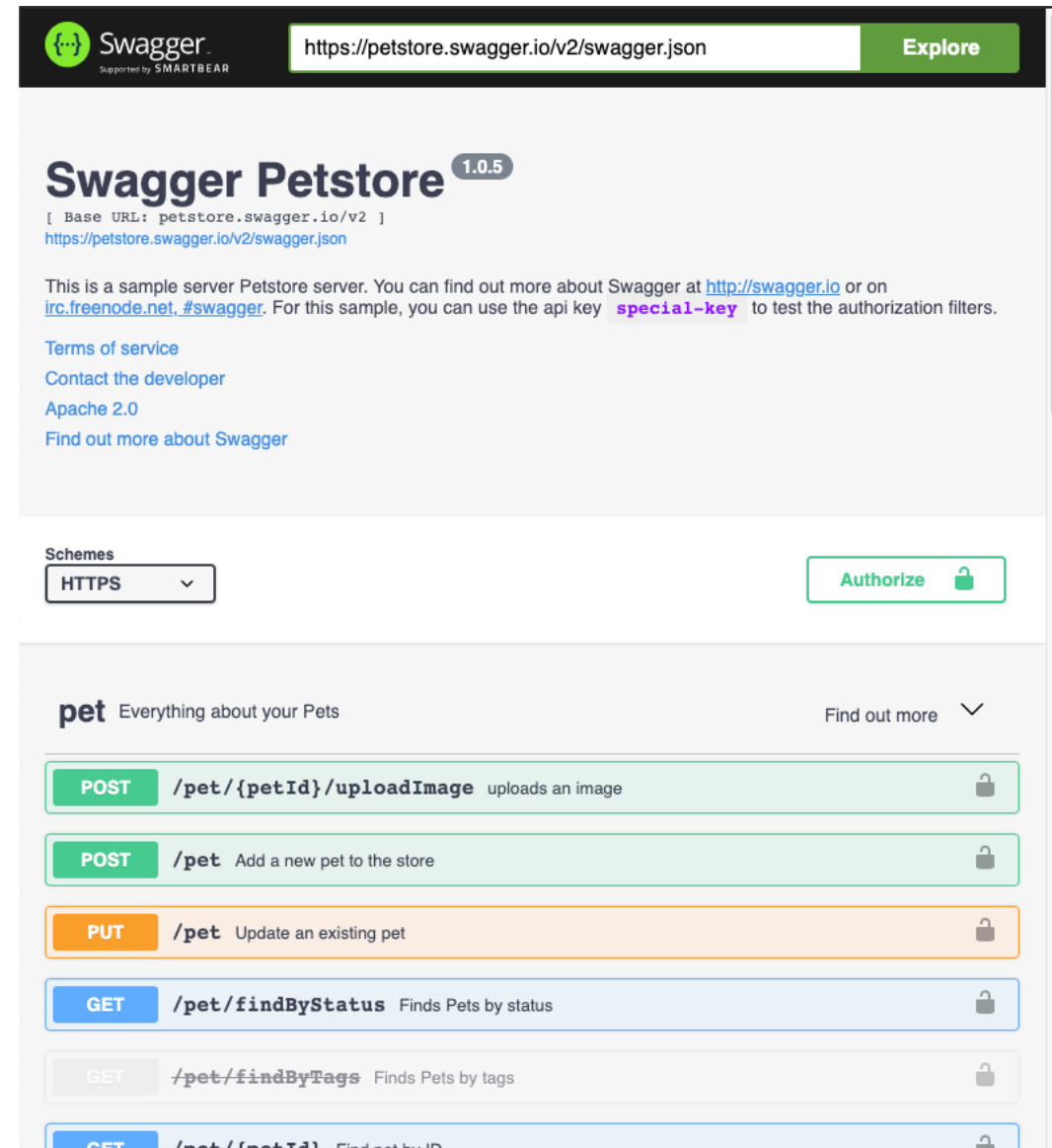
Swagger CodeGen

- Swagger Codegen può semplificare il processo di compilazione generando stub di server e SDK client per qualsiasi API, definiti con la specifica OpenAPI (precedentemente nota come Swagger), in modo che il tuo team possa concentrarsi meglio sull'implementazione e l'adozione dell'API.

```
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]
```


Swagger UI

L'interfaccia utente di Swagger consente a chiunque, sia che si tratti del tuo team di sviluppo o dei tuoi consumatori finali, di visualizzare e interagire con le risorse dell'API senza disporre di alcuna logica di implementazione. Viene generato automaticamente dalla tua specifica OpenAPI (precedentemente nota come Swagger), con la documentazione visiva che semplifica l'implementazione back-end e il consumo lato client



Code First vs Design First

Grazie a swagger e l' integrazione con Spring Boot abbiamo la possibilità di procedere in 2 modi per documentare le nostre API e per generare eventuali client sdk.

- Code First
 - Abbiamo la possibilità di integrare swagger nel nostro progetto spring boot grazie alle librerie

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
</dependency>


<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-webmvc-core</artifactId>
</dependency>
```

e senza dover scrivere una riga di configurazione, quando verrà effettuata la build della nostra applicazione, verrà generato anche il file **JSON** sotto **/v3/api-docs** con le definizioni delle nostre api oltre ad un'interfaccia **SWAGGER UI** disponibile di default sotto **/swagger-ui.html**.

Grazie alle proprietà di spring è sempre possibile cambiare i valori di default.

Ultima volta che il progetto è avviato, è possibile usare **SWAGGER CODEGEN** per la generazione di eventuali client sdk.

Swagger Code First

 **Swagger**
Supported by SMARTBEAR

/v3/api-docs

Explore

OpenAPI definition

v0 OAS3

/v3/api-docs

Servers

http://localhost:8081 - Generated server url

users-controller

GET /api/users

POST /api/users

GET /api/users/{userId}

Schemas

UserDTO >

Swagger Design First

Partiamo dalla creazione del file yml o json con a definizione delle nostre API in formato openapi.

Per questo esempio useremmo un file yml che posizioneremmo sotto la directory resource del nostro progetto spring boot.

```
1  openapi: 3.0.1
2  info:
3    title: Swagger DEMO
4    description: Smaple demo hello world.
5    termsOfService: http://swagger.io/terms/
6    contact:
7      email: cmoraru@demo.com
8    license:
9      name: Apache 2.0
10     url: http://www.apache.org/licenses/LICENSE-2.0.html
11   version: 1.0.0
12  servers:
13    - url: http://localhost:8081/api
14  tags:
15    - name: demo
16      description: Hello world demo
17      externalDocs:
18        description: Find out more
19        url: localhost:8081
20  paths:
21    /hello:
22      get:
23        tags:
24          - demo
25        summary: Return helllo world!
26        description: Return hello world string
27        operationId: sayHello
28        responses:
29          200:
30            description: return hello world string
31            content: {}
32  components: {}
33
```

Swagger Design First

Aggiungiamo le dipendenze necessarie al nostro progetto spring boot base:

- Spring UI

```
<dependency>  
  <groupId>org.springdoc</groupId>  
  <artifactId>springdoc-openapi-ui</artifactId>  
  <version>1.4.8</version>  
</dependency>
```

- Swagger

```
<dependency>  
  <groupId>io.swagger</groupId>  
  <artifactId>swagger-annotations</artifactId>  
</dependency>  
<dependency>  
  <groupId>javax.validation</groupId>  
  <artifactId>validation-api</artifactId>  
</dependency>  
<dependency>  
  <groupId>org.openapitools</groupId>  
  <artifactId>jackson-databind-nullable</artifactId>  
</dependency>
```

Swagger Design First

All'interno del nostro progetto possiamo integrare il plugin per la generazione del codice in uno dei formati supportati dallo swagger codegen attraverso i task execution.

openapi-generator-maven-pluginw

L'output di default e sotto target/generated_source >>

Nel nostro case, eseguendo il comando:

mvnw compile

il file specificato in `inputSpec` verrà usato per la generazione dei sorgenti che rappresentano le nostre API definitions

```
<plugin>
  <groupId>org.openapitools</groupId>
  <artifactId>openapi-generator-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals><goal>generate</goal></goals>
      <configuration>
        <inputSpec>
          ${project.basedir}/src/main/resources/openapi.yml
        </inputSpec>
        <generatorName>spring</generatorName>
        <apiPackage>io.tej.SwaggerCodgen.api</apiPackage>
        <modelPackage>io.tej.SwaggerCodgen.model</modelPackage>
        <output>${project.basedir}</output>
        <supportingFilesToGenerate>
          ApiUtil.java
        </supportingFilesToGenerate>
        <configOptions>
          <sourceFolder>src/main/java/</sourceFolder>
          <delegatePattern>true</delegatePattern>
          <interfaceOnly>true</interfaceOnly>
        </configOptions>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Swagger Design First

Dopo la compilazione dei sorgenti delle nostre api, per ogni path definito all'interno del nostro yaml openapi, un'interfaccia corrispondente del tipo:

yaml path	java interface
/hello	HelloApi.java
/users	UsersApi.java

Ispezionando una qualunque delle interfacce, possiamo notare come all'interno nella stessa tutte le definizioni della nostra risorsa siano definite attraverso le annotations di swagger

`io.swagger.swagger-annotations`

I metodi generati corrispondono alle key operationId definite all'interno del nostro yaml openapi. I metodi sui quali fare override sono commentati con la dicitura "Override this method" all'interno dell'interfaccia generata.

Swagger Design First

Creiamo un RestController HelloController.java che estende **HelloApi** che sarà stato da swagger.

```
@RestController  
public class HelloController implements HelloApi {
```

Facciamo l'Override del metodo definito nelle nostre api spec "sayHello" e implementiamo la nostra logica.

```
@Override  
public ResponseEntity<String> sayHello() {  
    return ResponseEntity.ok("Hello World!");  
}
```

In questo modo, attraverso l'uso delle interfacce, stipuliamo un "contratto" con i client che vorranno consumare le nostre api.

FINE