

OpenAPI Generator



Software Systems Architecture and Integration

January 2021

Víctor Graván Bru (vgravanbru@gmail.com)

Juan Antonio Ortiz Guerra (juanantonioortizguerra@gmail.com)

María del Mar Vallejo Gamboa (vallejo.mmar@gmail.com)

Alberto Antonio Tokos Matas (belatm12@protonmail.com)

Nuria Gómez Arias (nuriafyq23@gmail.com)

Tutor: Sergio Segura Rueda

Group number: L6 - 02

Link to the project in GitHub: <https://github.com/OpenAPITools/openapi-generator.git>

VERSION HISTORY

Date	Version	Description	Participants
07/01/2021	1.0	- Initial version.	

Índice

1	Introducción	4
2	Visión general	6
3	Participantes	8
4	Vistas	9
4.1	Vista de contexto	9
4.2	Escenarios de uso	10
4.3	Vista funcional	11
4.4	Vista de despliegue	13
4.5	Vista de Desarrollo	15
5	Puntos de variabilidad y extensión	16
6	Análisis de atributos de calidad	17
7	Sugerencias de mejora	19
8	Contribuciones al proyecto	20
9	Conclusiones	24
	Referencias	25

1 Introducción



Originalmente conocido como especificación Swagger¹, **OpenApi** es una especificación que proporciona interfaces entendibles por máquina para realizar, consumir, describir y visualizar servicios web RESTful².

Comenzó siendo parte del marco Swagger pero más adelante se transformó en un proyecto separado a partir del 2016, supervisados por la iniciativa OpenAPI³, el cual colabora con código abierto en la fundación Linux swagger y otras herramientas para generar código, documentación y casos de prueba.

Versión	Fecha	Notas
3.0.3	2020-02-20	Lanzamiento de parche de la especificación OpenAPI 3.0.3
3.0.2	2018-10-08	Lanzamiento de parche de la especificación OpenAPI 3.0.2
3.0.1	2017-12-06	Lanzamiento de parche de la especificación OpenAPI 3.0.1
3.0.0	2017-07-26	Lanzamiento de la especificación OpenAPI 3.0.0
2.0	2014-09-08	Lanzamiento de Swagger 2.0
1.2	2014-03-14	Publicación inicial del documento formal
1.1	2012-08-22	Lanzamiento de Swagger 1.1
1.0	2011-08-10	Primera versión de la especificación Swagger

La principal diferencia entre OpenApi y OpenApi Generator

1. OpenApi: se centra en las especificaciones.
2. OpenApi generator: genera todo el código a partir de OpenApi mediante una serie de instrucciones que le puede dar el usuario.

OpenApi generator

En julio del 2018, Willian Chen, el mayor contribuyente de Swagger Codegen y más de 40 colaboradores de Swagger decidieron bifurcar todo el código hacia un proyecto llamado OpenApi Generator bajo la organización OpenApi Tools.

Organización OpenApi Tools




Ha sido creada por unos expertos que creen firmemente en la inmensa importancia de realizar estándares para describir las APIS.

Por otro lado la gobernanza esta bajo la fundación linux, sin embargo OpenApi organisation esta más enfocada en crear y promover un formato para mejorar la descripción de las APIS.

Además se ha basado en la especificación Swagger que fue donada por SmartBear Software.


Miembros de la organización

Current Members




Ron Ratovsky
2016 – CURRENT

[Twitter](#) [LinkedIn](#) [Website](#)




Darrell Miller
2016 – CURRENT

[Twitter](#) [LinkedIn](#) [Website](#)




Marsh Gardiner
2016 – CURRENT

[Twitter](#) [LinkedIn](#) [Website](#)




Jeremy Whitlock
2016 – CURRENT

[Twitter](#) [LinkedIn](#) [Website](#)



Uri Sarid
2018 – CURRENT

[Twitter](#) [LinkedIn](#) [Website](#)



Mike Ralphson
2018 – CURRENT

[Twitter](#) [Website](#)

Former Members

- Tony Tam (2016-2017)
- Jason Harmon (2016-2017)

2 Visión general

OpenAPI Generator proporciona generación de librerías para clientes API (generación de SDK), server stubs (implementaciones API de testeo), documentación y configuración de forma automática dada una OpenAPI Spec (están soportadas tanto 2.0 como 3.0). Actualmente, están soportados los siguientes lenguajes/frameworks:

	Lenguajes/Frameworks
Clientes API	ActionScript, Ada, Apex, Bash, C, C# (.net 2.0, 3.5 or later, .NET Standard 1.3 - 2.0, .NET Core 2.0, .NET 5.0. Libraries: RestSharp, HttpClient), C++ (cpp-restsdk, Qt5, Tizen, Unreal Engine 4), Clojure, Crystal, Dart, Elixir, Elm, Eiffel, Erlang, Go, Groovy, Haskell (http-client, Servant), Java (Jersey1.x, Jersey2.x, OkHttp, Retrofit1.x, Retrofit2.x, Feign, RestTemplate, RESTEasy, Vertx, Google API Client Library for Java, Rest-assured, Spring 5 Web Client, MicroProfile Rest Client), k6, Kotlin, Lua, Nim, Node.js/JavaScript (ES5, ES6, AngularJS with Google Closure Compiler annotations, Flow types, Apollo GraphQL DataStore), Objective-C, OCaml, Perl, PHP, PowerShell, Python, R, Ruby, Rust (hyper, reqwest, rust-server), Scala (akka, http4s, scalaz, sttp, swagger-async-httpclient), Swift (2.x, 3.x, 4.x, 5.x), Typescript (AngularJS, Angular (2.x - 11.x), Aurelia, Axios, Fetch, Inversify, jQuery, Nestjs, Node, redux-query, Rxjs)
Server stubs	Ada, C# (ASP.NET Core, NancyFx), C++ (Pistache, Restbed, Qt5 QHTTPEngine), Erlang, F# (Giraffe), Go (net/http, Gin), Haskell (Servant), Java

	(MSF4J, Spring, Undertow, JAX-RS: CDI, CXF, Inflector, Jersey, RestEasy, Play Framework, PKMST ⁴ , Vert.x ⁵), Kotlin (Spring Boot, Ktor, Vertx), PHP (Laravel, Lumen, Slim, Silex, Symfony ⁶ , Zend Expressive ⁷), Python (Flask), NodeJS, Ruby (Sinatra, Rails5), Rust (rust-server), Scala (Akka, Finch ⁸ , Lagom ⁹ , Play ¹⁰ , Scalatra)
Generadores de documentación API	HTML, Confluence Wiki, AsciiDoc, Markdown, PlantUML
Ficheros de configuración	Apache2 ¹¹
Otros	GraphQL, JMeter, Ktorm, MySQL Schema, Protocol Buffer

FLUJO DE TRABAJO

1. Se especifica el fichero .yaml para describir los modelos y relaciones de la base de datos.
2. Se llama internamente a generadores abstractos de endpoints
3. Se crea un modelo por cada uno ya especificado en archivo .yaml
4. Se generan las apis.
5. Creación de test en base a las apis.
6. Se repite mientras queden modelos en el fichero indicado anteriormente
7. Se genera la seguridad con api key.
8. Por último obtenemos un proyecto con la documentación, endpoints y test cases.



Figura 1: Flujo de trabajo OpenAPI Generator

3 Participantes

Adquisidores/Proveedores

Los adquisidores son los participantes que financian el proyecto. En el caso de OpenAPI Generator son los patrocinadores, que también proporcionan la infraestructura necesaria. Algunos ejemplos son GoDaddy¹² (proporciona dominios web), o Linode¹³ (proporciona una VPS (Virtual Private Server, o servidor virtual privado))

Asesores

No hay un rol específico de asesor en este proyecto. OpenAPI Generator posee licencia Apache License 2.0. A menos que sea requerido por ley o se acuerde por escrito, todo software distribuido bajo esta licencia se distribuye bajo el criterio “tal cual”, sin garantías o condiciones de ningún tipo, ya sea de forma explícita o implícita.

Comunicadores/Soporte Técnico/Responsables de Mantenimiento

Está formado por el *Comité Técnico de OpenAPI Generator* (CTOG), formado por una parte de los colaboradores y por usuarios, que se dedican a guiar a otros usuarios, realizan mejoras al generador, y revisar problemas del programa, entre otras cosas.

Desarrolladores

Son miembros de la comunidad (principalmente miembros del CTOG), que se dedican a mejorar el generador o crear plantillas para el generador

Usuarios/Testeadores

Son los que ponen a prueba y utilizan la infraestructura para generar librerías API, software de relleno para servidores o configuración de forma automática, como grandes empresas tecnológicas o redes sociales. Algunos ejemplos son Allianz¹⁴, Kubernetes¹⁵, o Twitter¹⁶. Se puede encontrar una lista con todas las compañías o proyectos que lo utilizan aquí¹⁷.

Competencia

Está formada por aquellos sistemas que ofrecen un servicio similar a OpenAPI Generator, como generator-rest¹⁸

4 Vistas

4.1 Vista de contexto

La vista de contexto contiene las relaciones entre el proyecto y los participantes y dependencias involucradas en el entorno de desarrollo del proyecto. La **Figura 2** muestra el diagrama de contexto de OpenApi-generator. El proyecto se encuentra en un repositorio de github, con el objetivo de facilitar el control de versiones y la documentación del proyecto, así como a su vez, llevar el manejo de las distintas issues que se puedan crear. OpenApi se trata de un proyecto de código abierto, por lo cual cualquiera puede formar parte del equipo de desarrollo. No obstante, cabe mencionar que hay un equipo principal de desarrolladores. Varias organizaciones patrocinan y usan la herramienta para automatizar la generación de APIs. El proyecto está principalmente desarrollado en Java y usa Maven para automatizar el manejo de dependencias. No obstante, se pueden apreciar otros lenguajes como JavaScript o TypeScript, haciendo posible la instalación de la herramienta de diferentes maneras. Esta multitud de posibilidades de instalación, hacen posible que la herramienta se pueda usar en cualquier sistema operativo, ya que soporta docker y vagrant, así como otras formas a través de npm y Homebrew. A su vez, la herramienta necesita de diferentes dependencias para ser testeada, tales como se pueden ver en la **figura 2**; y utiliza Swagger como herramientas de apoyo. Finalmente, OpenApi generator, compite con otras herramientas para la autogeneración de APIs del tipo REST. Estas son rest-hapi¹⁹ y Generator-rest, entre otras, aunque cabe destacar que OpenApi generator mantiene cierta competencia con AsyncApi²⁰, siendo esta última usada para autogenerar APIs asíncronas, y posee una licencia de Apache.

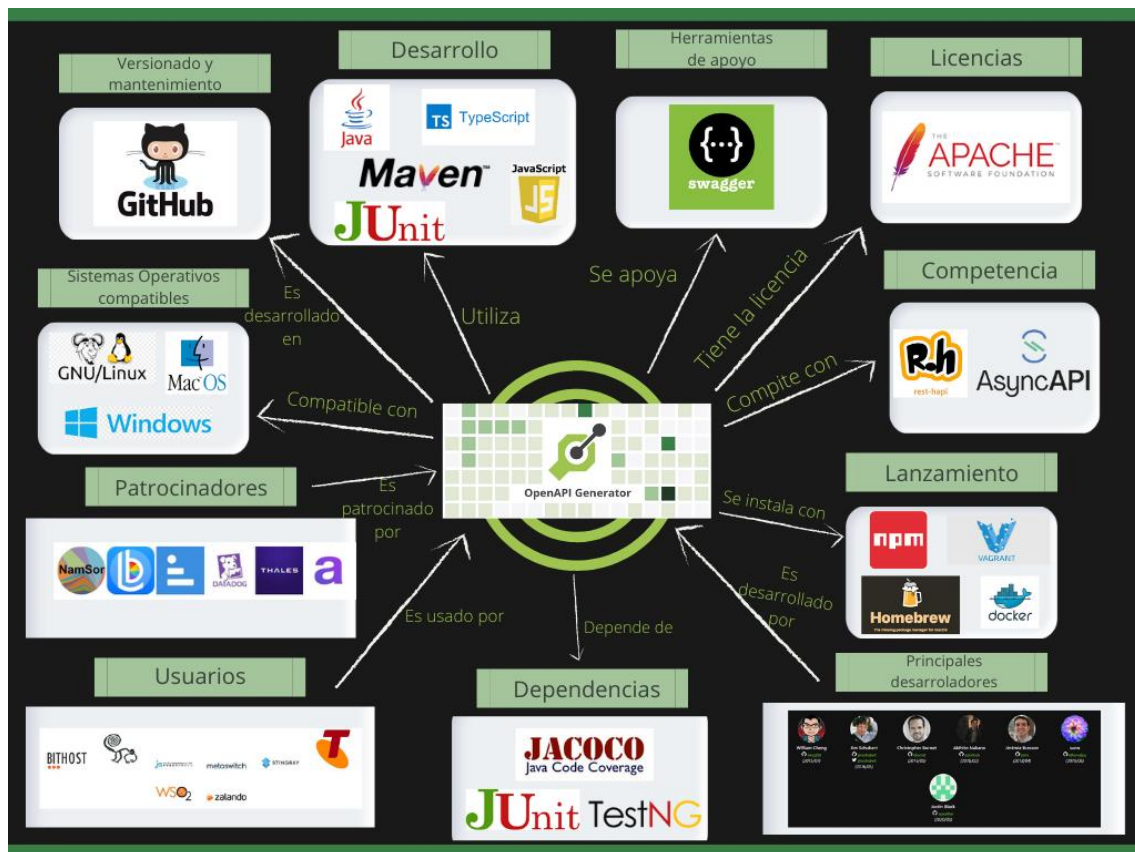


Figura 2: Diagrama de Contexto

4.2 Escenarios de uso

- **Generar API:** Este es la principal funcionalidad del proyecto.

Puede generar una API por defecto (en este caso es una API para una petstore) pero también se puede modificar un archivo de configuración y así crear automática una API según las especificaciones del propio usuario.

También cuenta con la opción de sistema de autenticación para la API.

Además, al generar la API, se crea también la documentación de esta.

- **Generación automática de test:** Además de generar APIs, OpenAPI Generator le permite al usuario la creación automática de Test una vez generada la API deseada.

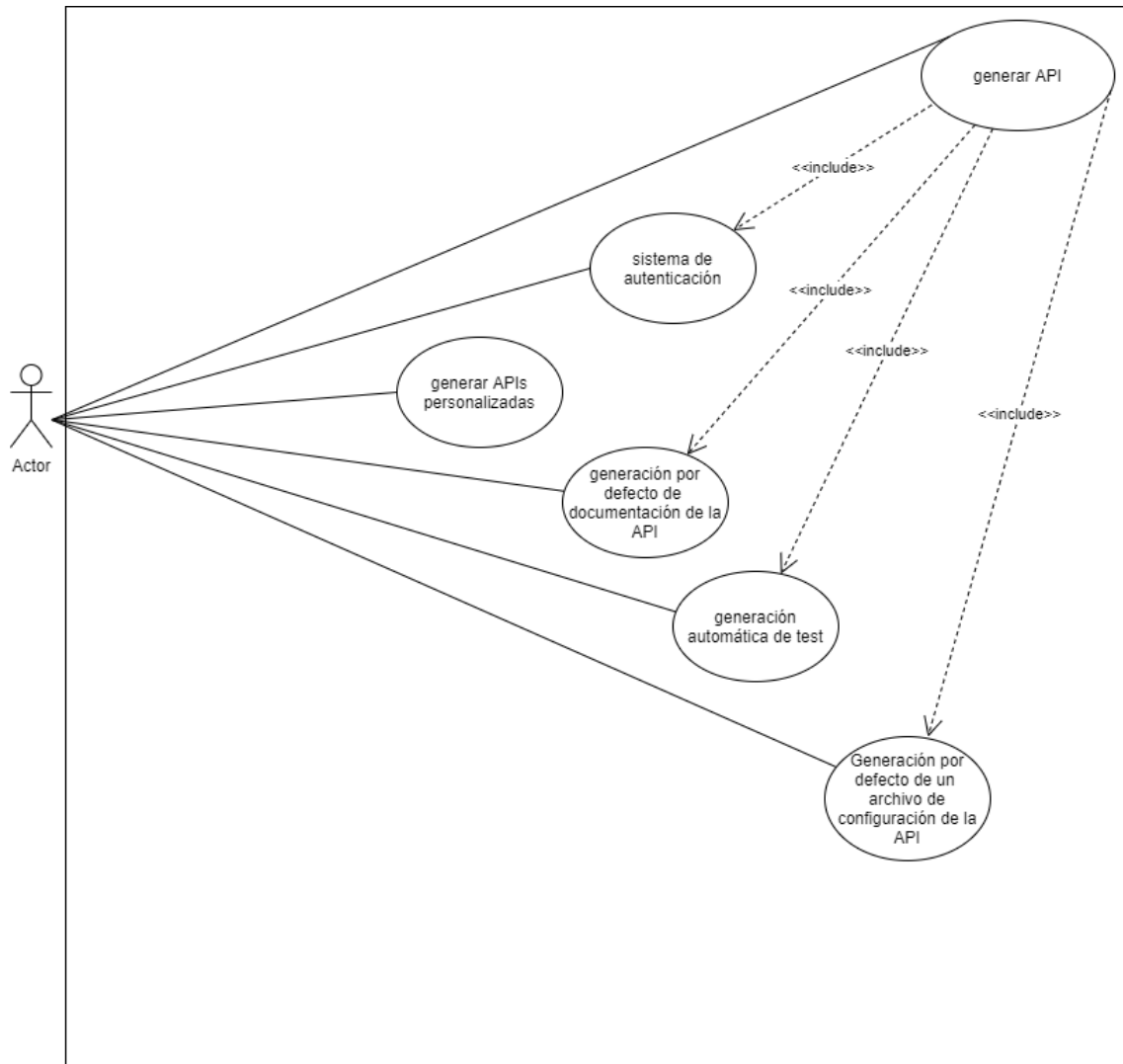


Figura 3: Diagrama UML de casos de uso

4.3 Vista funcional

La vista funcional muestra los diferentes componentes lógicos utilizados en la ejecución del sistema. OpenApi puede dividirse en los siguientes componentes lógicos:

- **EXEC CONFIG.** Se encarga de configurar el sistema para iniciar OpenApi con las especificaciones dadas. Pueden darse directamente por consola o a través de un archivo en formato json.
- **API GENERATOR.** Esta es el principal componente, debido a que se encarga de generar la API. Genera por defecto la documentación de la API, basada en servicios REST. Una vez generada, implementa los casos de test de la API. API GENERATOR utiliza ciertas clases como DefaultGenerator.java o DefaultCodeGen.java para la elaboración por defecto.
- **API Specification.** Se encarga de generar APIs mediante unas especificaciones adicionales. La clase DefaultCodeGen.java, puede presentar datos adicionales

especificados por el cliente. De ser así, se creará una API por defecto, a la que se añaden las especificaciones del cliente.

- **TEST GENERATOR.** Este componente se encarga de generar tests abstractos. OpenApi genera tests en base a los archivos generados por la API y por la respuesta generada al hacer consultas a la API. Sin embargo, la generación de casos de prueba no es la principal funcionalidad de OpenApi.
- **Test Specification.** Se encarga de generar casos de prueba específicos al pasarlo en formato json. Al no ser la principal funcionalidad de OpenApi, puede ser bastante limitado su uso.
- **Runner.** Se encarga de la generación de modelos y la documentación de la API según la implementación de API GENERATOR, así como de los casos de prueba generados por TEST GENERATOR.
- **API DOC.** Documentos generados al ejecutar OpenAPI. Actualmente, OpenAPI soporta multitud de lenguajes de programación para generar la documentación, haciendo la generación automática accesible a multitud de clientes.
- **Other Utilites.** Incluye multitud de clases que soportan operaciones adicionales en el proyecto.

La siguiente figura muestra el diagrama de componentes descrito:

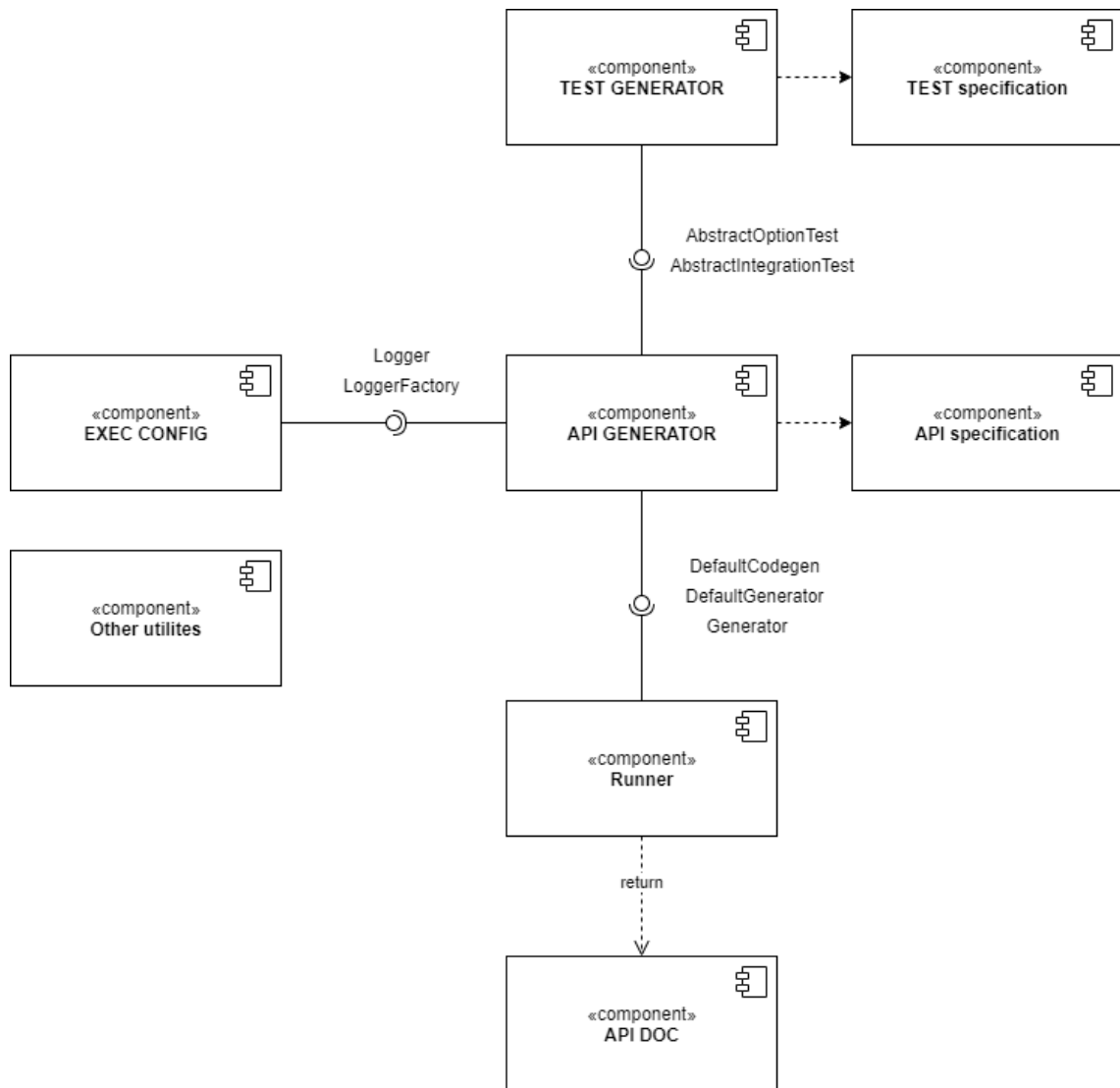


Figura 4: Diagrama UML de componentes

4.4 Vista de despliegue

La vista de despliegue describe el entorno hardware y software en el que el sistema será desplegado. En este caso, el despliegue se realiza mediante Maven, por lo que solo requiere una instalación de Java JDK (al menos Java 8) y Apache Maven (mínimo 3.3.4). Además, existen otras formas de despliegue, como crear un contenedor docker que ejecute el proyecto. El Dockerfile utiliza una versión de Java jdk 11 y maven 3.6.3.

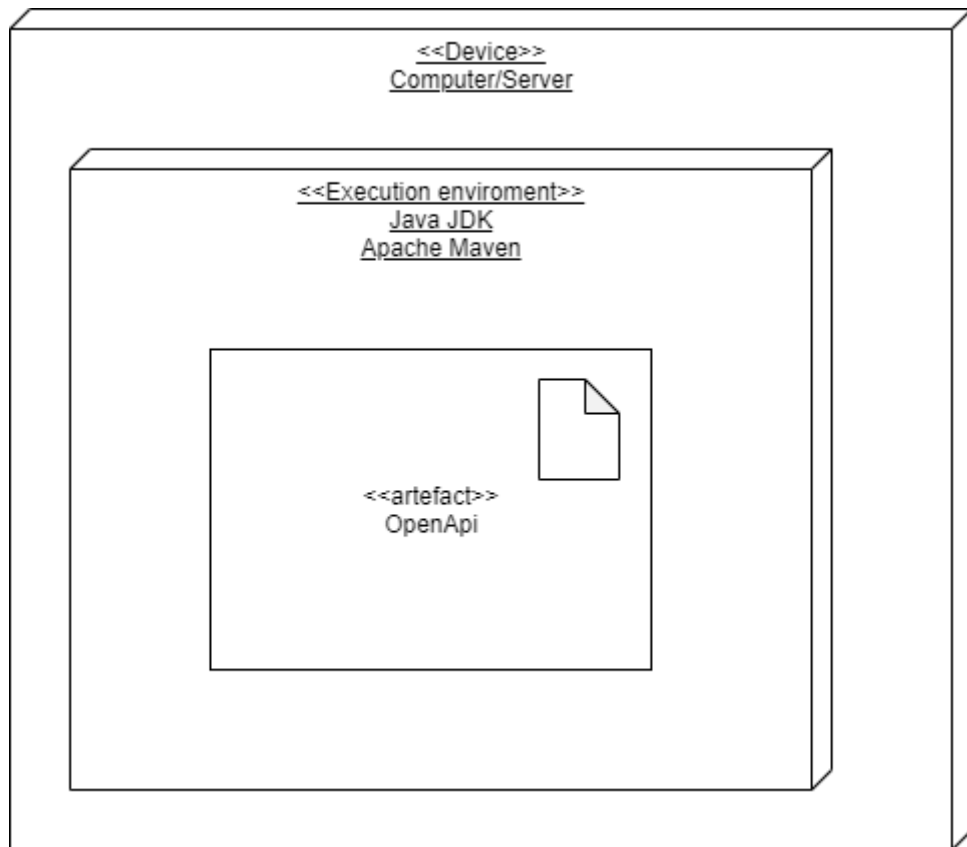


Figura 5: Diagrama UML de despliegue

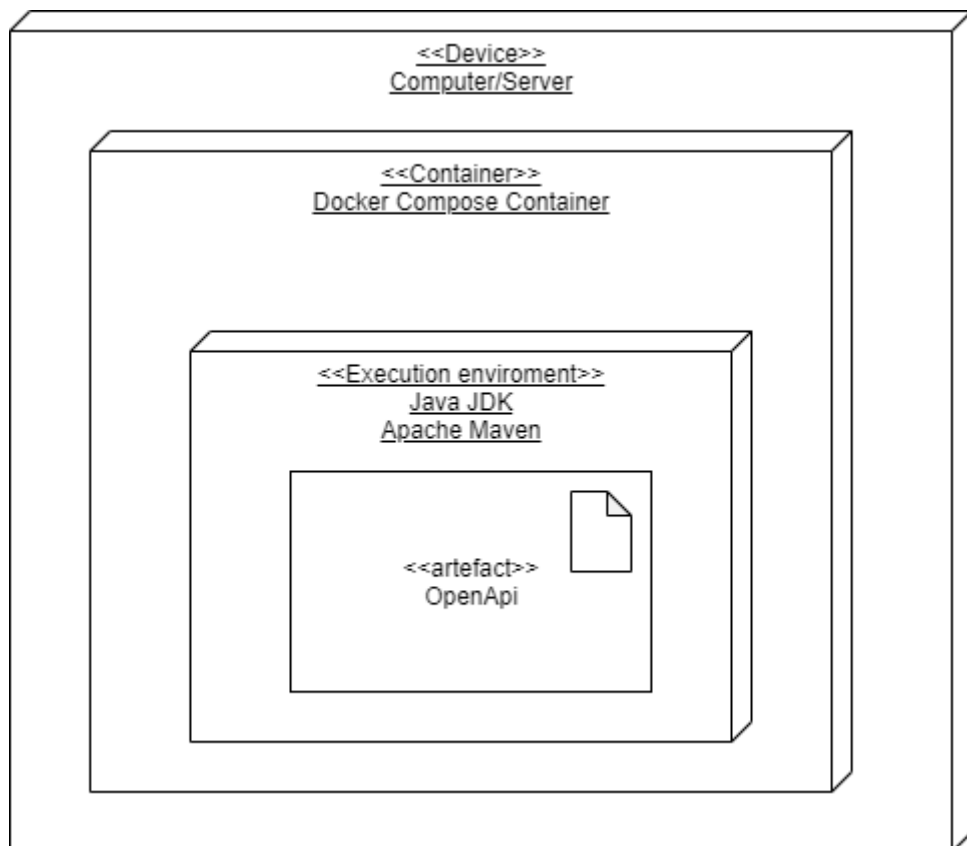
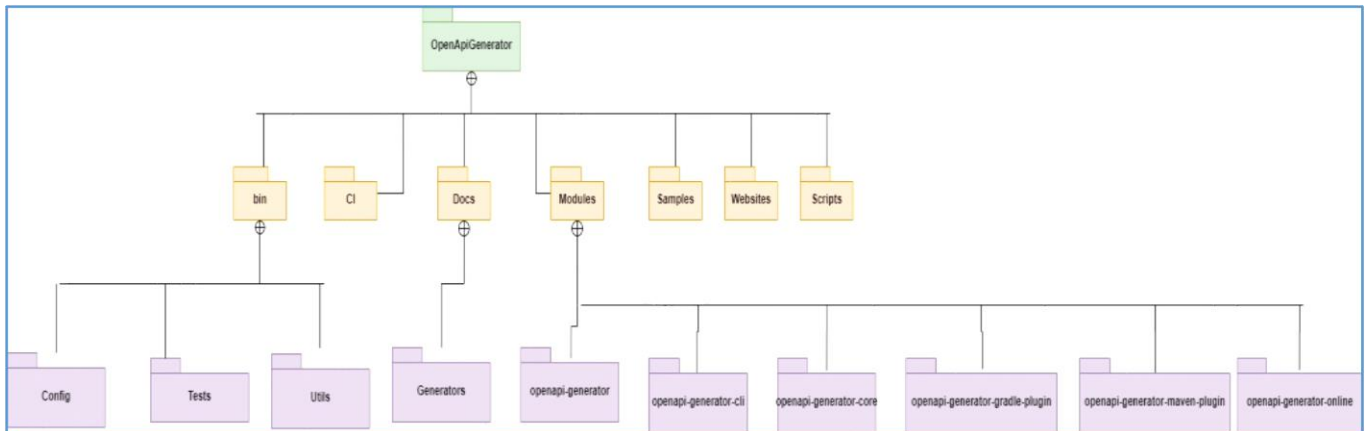


Figura 6: Diagrama UML de despliegue con Docker

4.5 Vista de Desarrollo



1. **BIN:** archivos binarios y .sh para la ejecución del proyecto junto.
2. **CI:** configuración del entorno del proyecto
3. **DOCS:** documentación necesaria para generar los markdown de nuestro proyecto RESTful.
 - a. **Generators:** se encuentran archivos de documentación específicos para todos los lenguajes de programación disponibles en los que podemos hacer las apis.
4. **MODULES:**
 - a. **Openapi-generator:** autenticación, serializadores JSON, plantillas lambda, excepciones y ejemplos en todos los lenguajes implementados junto a sus configuraciones.
 - b. **Openapi-generator-cli:** incluye todos los comandos (junto a sus tests) que podemos usar poniendo openapi-generator-cli en la consola de comandos junto a su detallada documentación.
 - c. **Openapi-generator-core:** se encarga de generar la estructura o núcleo del proyecto.
 - d. **Samples:** ejemplos de apis creadas con open-api-generator-cli en cada lenguaje de programación disponible en el sistema, junto a su configuración archivos .yaml, configuración y tests.
 - e. **Scripts:** se implementa el comando openapi-generator-cli help el cual muestra todos los comandos disponibles en la aplicación.
 - f. **Website:** archivos json para idiomas del despliegue en la web de la app, estilos css, html, js, y páginas web.

5 Puntos de variabilidad y extensión

Los puntos de variabilidad hacen referencia a las partes donde el sistema puede variar y por lo tanto, es posible elegir entre una o varias alternativas a las que llamamos “variantes”. Hemos identificado los siguientes puntos de variabilidad:

- **Creación de plantillas:** Normalmente, sólo se modifica los archivos de plantilla Moustache para la creación del propio código personalizado. En el directorio “openapi-generator/modules/openapi-generator/src/main/java/org/openapitools/codegen/languages/” están las plantillas para cada generación de código (lógica transformadora). Cada generador creará una estructura de datos a partir del documento OpenApi.

Hay dos formas, las plantillas ya integradas, están escritas en Moustache y procesadas por jmustache²¹ y por otro lado las plantillas definidas por el usuario. En cuanto a los generadores, es posible a partir de uno ya existente en el proyecto poder implementar una plantilla personalizada pero OpenAPI Generator no solo admite archivos locales para la creación de plantillas. Como usuario puede cargar una plantilla a través de classpath o incluso puede ejecutar su propio generador personalizado para la creación de la misma con todos los archivos que necesita. Una vez compilado puede usar el generador y su propia lógica personalizada.

No siempre Moustache se adapta a las necesidades del usuario por lo que éste podrá elegir también usar plantillas Pebble²².

- **Generadores:** Como antes he mencionado en el punto anterior, el usuario puede elegir entre un generador ya integrado o hacer su propio generador personalizado. Puede controlar la generación de documentos y pruebas para api, así como qué plantillas son las necesarias para una generación específica. En cuanto a la personalización del mismo, el usuario puede optar por dejar que los archivos de configuración propios del sistema cree los valores predeterminados o por el contrario configurar por sí mismo nombres de paquetes, prefijos, etc, usando un archivo de configuración json.

Los puntos de extensión se definen como aquellas partes del sistema preparadas para que la aplicación pueda ser extendida en el futuro, es decir para facilitar futuros cambios en el proyecto.

- **Online OpenApi-Generator:** Linode ofrece un servicio público online que integra OpenApi-Generator para la ejecución online del proyecto. Sin embargo, este servicio está en beta, por lo que no garantiza la misma calidad que el proyecto.

6 Análisis de atributos de calidad

En nuestro proyecto, hemos analizado los siguientes atributos de calidad, medidos mediante la herramienta de SonarCloud²³:

- **Fiabilidad:** Este atributo mide la capacidad del sistema para trabajar de la forma esperada. Es medido a través de los defectos (bugs) encontrados en el sistema, aunque para mayor conocimiento, debe ser requerido consultar la cobertura, así como demás atributos. SonarCloud puntúa este atributo con una calificación de **E**, ya que detecta 5 bugs en el proyecto y 5.8K de líneas de código “raro” (code smells). Estos bugs se deben a no seguir la manera aconsejada de instanciar ciertas clases con Mockito²⁴, expresiones regulares no testadas que pueden generar un “*stack overflow error*” si la entrada es grande o no considerar otras expresiones regulares que puedan repetir elementos una entrada vacía, pasando por defecto, el propio valor de la expresión regular. Además, como se puede ver en el apartado de cobertura, opinamos que no es suficiente, al solo cubrir un 1,1% del código del proyecto en un total de 72 test unitarios.
- **Seguridad:** Este atributo mide la capacidad del sistema para evitar vulnerabilidades y riesgos de acceso no autorizado a datos del sistema. OpenApi está pensado para autogenerar la documentación de una Api en local, por lo que no habría problema en un principio al ser local. SonarCloud puntúa con **A** la vulnerabilidades halladas (0), aunque cabe mencionar que hay posibles riesgos al encontrarse “puntos calientes” (48 en total) de seguridad en diversos archivos y paquetes del proyecto. Estos archivos se pueden clasificar según sonarcloud por su orden de prioridad para revisar, pudiendo encontrar tres diferentes categorías según la importancia del punto caliente. Para mayor detalle, se recomienda consultar CVE details²⁵.
 - Con una calificación **HIGH Priority**, encontramos 18 archivos que deben ser revisados debido a que sonarcloud detecta “*hard code credentials*”, es decir, posibles referencias a contraseñas y otros datos sensibles dentro del código. Sin embargo, analizando el código, creemos que son referencias a parámetros generados en la propia clase, aunque aconsejamos revisar vulnerabilidades pasadas sobre este tema, como [CVE-2019-13466](#) o [CVE-2018-15389](#).
 - Con una calificación **MEDIUM Priority**, encontramos 15 archivos que deben ser revisados. Uno de ellos, sonarcloud detecta un archivo que podría presentar una vulnerabilidad en una expresión regular no testada, pudiendo causar un ataque DoS (denegación de servicio) debido a ello. Además, los otro 14 archivos, se deben al uso de generación de números pseudoaleatorios, de forma que podrían acotarse valores esperados y provocar un fallo o un acceso no autorizado por la baja calidad criptográfica. Para mayor interés, se recomienda consultar las siguientes vulnerabilidades registradas: [CVE-2013-6386](#), [CVE-2006-3419](#) o [CVE-2008-4102](#).
 - Con una calificación **LOW Priority**, encontramos 15 archivos que deben ser revisados. 13 de ellos se deben a recordar al usuario de asegurarse que el debug esté desactivado antes de entregar los archivos a producción. Esto

se debe a que hay archivos que dentro de una sentencia *try catch*, al detectar errores, lo imprime por consola, pudiendo facilitar tanto al usuario como a un tercero malicioso información respecto al error. Para mayor información, se recomienda consultar la vulnerabilidad más reciente respecto a este tema: [CVE-2018-1999007](#). Otro archivo, tiene un punto caliente interesante, dado que se podría aprovechar de los 13 puntos calientes anteriores. Se debe a un posible log injection, ya que se trata de logear por nivel los errores registrados. Para mayor información, consultar: [CVE-2018-0285](#). Y el último, es que SonarCloud detecta un string que indica una versión de un archivo por defecto como una dirección IP, por lo que no deberíamos preocuparnos.

Por todo ello, no es de extrañar que reciba una puntuación de **E** a lo que se refiere a “puntos calientes” en el tema de seguridad.

- **Mantenibilidad:** Este atributo mide la capacidad del sistema para ser mantenido ante el cambio por implementación de nuevas funcionalidades, corrección de bugs, implementación de nuevas versiones, etc. SonarCloud le da una calificación de **A**, aunque cabe destacar que detecta 5.8 code smells en el sistema. No obstante, la mayoría de estos Code Smells se deben a la sintaxis de ciertas clases de Java, al uso de elementos genéricos o a la falta de cobertura por tests. Creemos que el motivo de los code smells por utilizar ciertas clases Java se deben a la versión de Java utilizada (Java 8). SonarCloud calcula que el sistema posee una deuda técnica de 109 días, lo cual consideramos que es bastante alta, aunque debido a ser un proyecto OpenSource con una gran cantidad de colaboradores, no es tan notable dicha deuda. Finalmente, cabe mencionar que el proyecto presenta bloques duplicados, alcanzando el 16.9% del proyecto, con un total de 1.1K de bloques duplicados.
- **Complejidad:** Este atributo mide la eficiencia de los métodos usados, además de la interacción del desarrollador al tener que realizar el mantenimiento de dichos métodos. La complejidad se puede dividir en complejidad ciclomática y cognitiva, que miden la eficiencia y la mantenibilidad respectivamente. Este proyecto presenta una calificación total de 17183 de complejidad ciclomática y de 16985 de complejidad cognitiva. Por último, se define por defecto que la complejidad cognitiva de un método debe ser a 15 para ser viable, no cumpliendo multitud de archivos, de los cuales destacan los siguientes archivos relacionados con la generación de modelos y del código de la API:
 - *modules/openapi-generator-corelib/src/main/java/org/openapitools/codegen/DefaultCodegen.java*
 - *modules/openapi-generator-corelib/src/main/java/org/openapitools/codegen/DefaultGenerator.java*
 - *modules/openapi-generator-corelib/src/main/java/org/openapitools/codegen/utils/ModelUtils.java*

7 Sugerencias de mejora

Basado en nuestro análisis, sugerimos implementar las siguientes mejoras.

1. **Documentación detallada de la ejecución y explicación del proyecto desde SpringBoot.** Debido a la falta de documentación, sugerimos mejorar la existente respecto al uso del proyecto. La documentación actual indica que hay diferentes formas de uso pero no las detalla, mostrando la más usada. Hemos creído conveniente aportar dicha documentación a través de nuestra propia experiencia usando OpenApi Generator.

[REQ] Contributed document with one spring boot app #9230

Open

jaortiz1 opened this issue now · 0 comments



jaortiz1 commented now

😊 ...

We have observed the lack of documentation of a spring boot application using open-api-generator, we send it here for you to add it.

thanks and greetings.

[springBootOpenApiExample.pdf](#)



jaortiz1 added the **Enhancement: Feature** label now

2. **Desarrollo de tests.** Debido a la gran falta de cobertura en el proyecto, creemos conveniente el desarrollo de más tests unitarios, ya que gran cantidad de puntos calientes detectados en el proyecto se deben a que no hay tests que los cubran.
3. **Incluir idioma castellano:** ampliar los archivos .json de la carpeta i18n de traducciones de texto, debido a que el idioma castellano es uno de los más usados y con más enriquecimiento léxico.



jaortiz1 commented now

😊 ...

We are a software team of the university of Seville and we think firmly that this project should add spanish language in i18n folder.

More info: [(<https://www.babbel.com/en/magazine/the-10-most-spoken-languages-in-the-world>)]

8 Contribuciones al proyecto

Documentación detallada de la ejecución y explicación del proyecto desde SpringBoot y Java 8

1. Instalación de NPM mediante la herramienta NodeJs

<https://nodejs.org/en/>

2. Abrimos la consola de comandos y llamamos al instalador de paquetes NPM para que nos instale MAVEN


npm install maven

3. Instalamos OpenApi generator cli para ejecutar comandos de openapi

4. `npm install @openapitools/openapi-generator-cli -g`

5. Abrimos powershell si estamos en windows 10 como administrador y realizmos el siguiente comando:

set-executionpolicy unrestricted -force

 Administrador: Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Windows\system32> set-executionpolicy unrestricted -force
PS C:\Windows\system32>
```

6. Nos descargamos el siguiente fichero de ejemplo para darle la información necesaria a OpenApi generator sobre nuestra API. (En nuestro caso hemos usado un fichero petstore.yaml que se encuentra en la documentación oficial de la app.



petstore.yaml

https://raw.githubusercontent.com/openapitools/openapi-generator/master/modules/openapi-generator/src/test/resources/3_0/petstore.yaml

7. Creamos una carpeta donde se vaya a alojar el proyecto con el archivo petstore.yaml contenido en ella.
8. Ejecutamos el siguiente comando dentro del directorio creado:
(unicamente modificamos el nombre del fichero .yaml a petstore.yaml o el que hayamos elegido)

```

openapi-generator-cli generate -g spring --library
spring-boot -i petstore.yaml -o ${PWD} -p
groupId=com.redhat -p artifactId=todo -p
artifactVersion=1.0.0-SNAPSHOT -p
basePackage=com.redhat.todo -p
configPackage=com.redhat.todo.config -p
apiPackage=com.redhat.todo.api -p
modelPackage=com.redhat.todo.model -p
sourceFolder=src/main/gen -p dateLibrary=java8 -p
java8=true

```

9. Añadimos el siguiente plugin a nuestro pom.xml

```

10. <plugin>
11.   <groupId>org.codehaus.mojo</groupId>
12.   <artifactId>build-helper-maven-plugin</artifactId>
13.   <version>3.1.0</version>
14.   <executions>
15.     <execution>
16.       <phase>generate-sources</phase>
17.       <goals>
18.         <goal>add-source</goal>
19.       </goals>
20.       <configuration>
21.         <sources>src/main/gen</sources>
22.       </configuration>
23.     </execution>
24.   </executions>
25. </plugin>

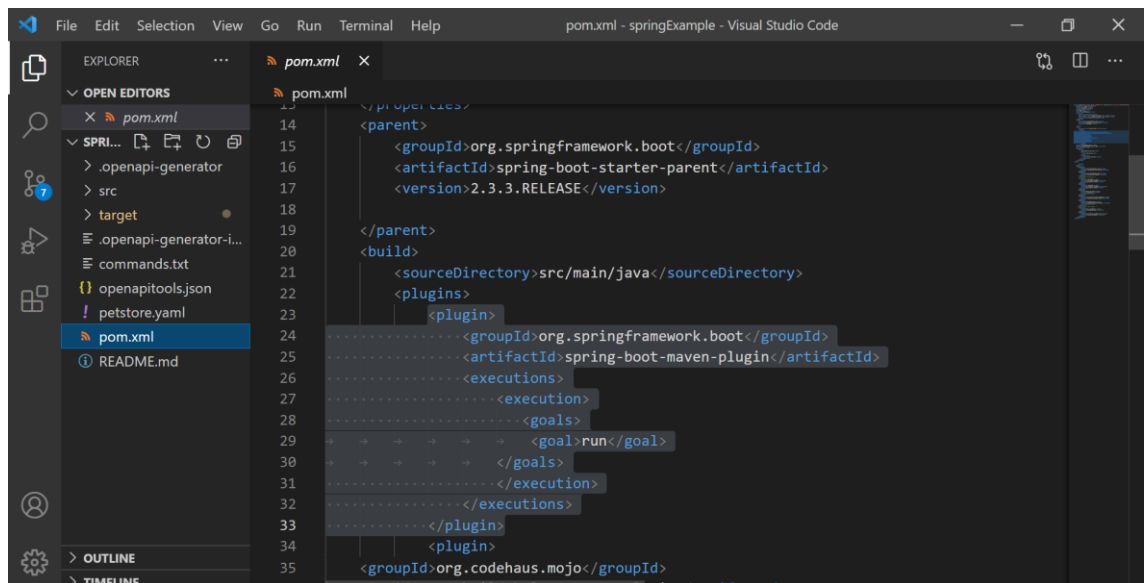
```

7. Añadiremos a plugin spring-boot-maven-plugin el siguiente apartado para que nos deje ejecutar la aplicación maven

```

<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>run</goal>
      </goals>
    </execution>
  </executions>
</plugin>

```



8. Abrimos la terminal y ejecutamos el siguiente comando dentro del directorio de nuestro proyecto para limpiar el proyecto e instalar todas sus dependencias modificadas:

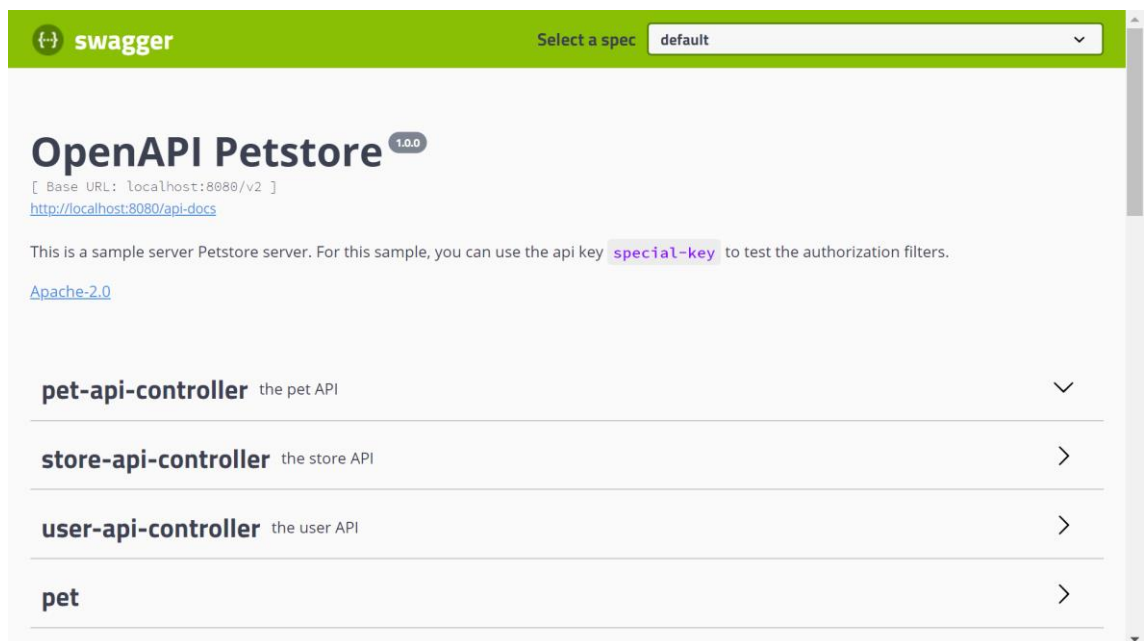
```
mvn clean install
```

9. No cierre la terminal ahora toca ejecutar el siguiente comando para que se quede la aplicación que hemos creado ejecutandose y poder acceder a ella desde cualquier navegador:

```
mvn spring:boot run
```

10. Abrimos el navegador (preferentemente firefox por el poco consumo de recursos y escribimos la siguiente URL):

<http://localhost:8080/swagger-ui.html#/>



11. Aquí nos encontraremos la documentación necesaria para hacer peticiones CRUD a nuestras apis

pet-api-controller

the pet API

>

store-api-controller

the store API

>

user-api-controller

the user API

>

pet

>

POST

/pet

Add a new pet to the store

🔒

PUT

/pet

Update an existing pet

🔒

GET

/pet/{petId}

Find pet by ID

🔒

POST

/pet/{petId}

Updates a pet in the store with form data

🔒

Inlusive se pueden ampliar y probar

pet

>

POST

/pet

Add a new pet to the store

🔒

Parameters

Try it out

Name

Description

pet required

Pet object that needs to be added to the store

(body)

Example Value | Model

<?xml version="1.0" encoding="UTF-8"?>
<Pet>
 <category>
 <id>0</id>
 <name>string</name>
 </category>
 <id>0</id>
 <name>doggie</name>
 <photoUrls>string</photoUrls>
 <status>available</status>
 <tags>
 <id>0</id>
 <name>string</name>
 </tags>
</Pet>

12. Modelos especificados:

Models

>

Category

>

description: A category for a pet

id integer(\$int64)

name string

}

File

>

absolute boolean

absoluteFile > {...}

absolutePath string

canonicalFile > {...}

canonicalPath string

directory boolean

executable boolean

file boolean

freeSpace integer(\$int64)

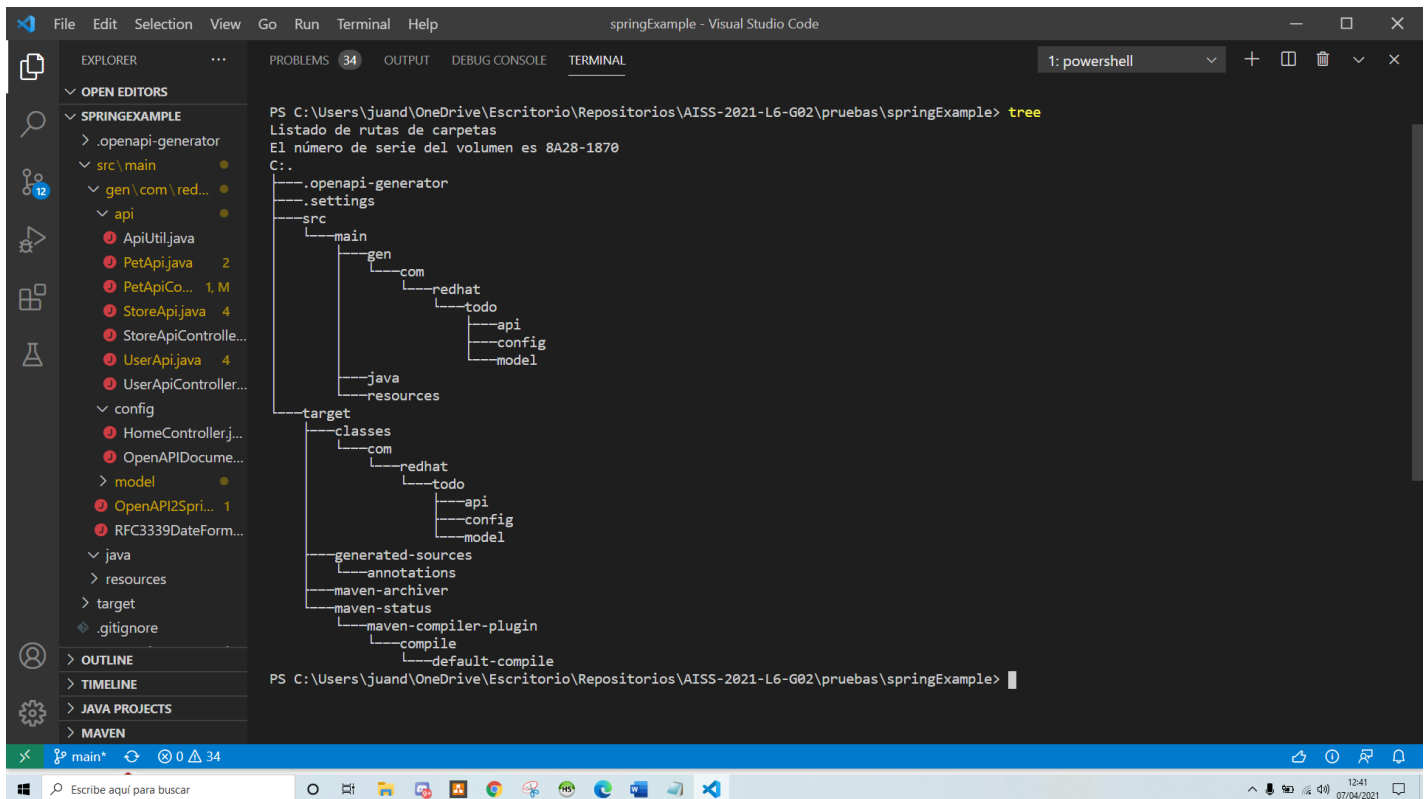
hidden boolean

lastModified integer(\$int64)

name string

IMPORTANTE: este proyecto usa [api key](#)

13. Estructura autogenerada de nuestro proyecto



The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left and the Terminal at the bottom. The Explorer sidebar shows the project structure for 'springExample', including folders like 'src/main', 'gen', 'config', 'java', 'resources', 'target', and 'maven'. The Terminal window shows the output of a 'tree' command executed in a PowerShell session, displaying the directory structure of the project.

```
PS C:\Users\juand\OneDrive\Escritorio\Repositorios\AISS-2021-L6-G02\pruebas\springExample> tree
Listado de rutas de carpetas
El número de serie del volumen es 8A28-1870
C:.
|-- .openapi-generator
|-- .settings
|-- src
|   |-- main
|   |   |-- gen
|   |   |   |-- com
|   |   |   |   |-- redhat
|   |   |   |   |   |-- todo
|   |   |   |   |   |   |-- api
|   |   |   |   |   |   |-- config
|   |   |   |   |   |   |-- model
|   |   |-- java
|   |   |-- resources
|   |-- target
|   |   |-- classes
|   |   |   |-- com
|   |   |   |   |-- redhat
|   |   |   |   |   |-- todo
|   |   |   |   |   |   |-- api
|   |   |   |   |   |   |-- config
|   |   |   |   |   |   |-- model
|   |   |-- generated-sources
|   |   |-- annotations
|   |   |-- maven-archiver
|   |   |-- maven-status
|   |   |-- maven-compiler-plugin
|   |   |-- compile
|   |   |-- default-compile
|-- .gitignore
-- .gitignore
```

14. RECOMENDACIÓN TESTING:

Recomendamos encarecidamente el uso de postman para probar las apis.

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

9 Conclusiones

Este documento proporciona diversa información sobre el diseño arquitectónico del proyecto OpenApi Generator, un proyecto de código abierto utilizado para la generación automática de APIs REST y casos de prueba de las mismas. Este proyecto se ha desarrollado utilizando diseños modulares, mostrando un rendimiento aceptable a pesar de algunos atributos de calidad analizados. Sin embargo, siempre se puede mejorar la herramienta a través de diferentes aspectos como la documentación proporcionada o mayor cobertura de tests unitarios. La mayor limitación, debido a esto, es la gran probabilidad de encontrar bugs y la gran dependencia de Java y Maven, pudiendo causar problemas al intentar integrar el servicio de OpenApi en diferentes extensiones como la que ofrece Linode¹³. Hemos contribuido al proyecto aportando la documentación necesaria para poder ejecutar el proyecto mediante el uso de Springboot.

Referencias

- [1] “[Swagger](#)”
- [2] “[RESTful](#)”
- [3] “[Iniciativa OpenAPI](#)”
- [4] “[PKMST](#)”
- [5] “[Vert.x](#)”
- [6] “[Symfony](#)”
- [7] “[Zend Expressive](#)”
- [8] “[Finch](#)”
- [9] “[Lagom](#)”
- [10] “[Play](#)”
- [11] “[Apache2](#)”
- [12] “[GoDaddy](#)”
- [13] “[Linode](#)”
- [14] “[Allianz](#)”
- [15] “[Kubernetes](#)”
- [16] “[Twitter](#)”
- [17] “[Lista de usuarios](#)”
- [18] “[generator-rest](#)”
- [19] “[rest-hapi](#)”
- [20] “[AsyncAPI](#)”
- [21] “[jmustache](#)”
- [22] “[Pebble Templates](#)”
- [23] “[SonarCloud](#)”
- [24] “[Mockito](#)”
- [25] “[CVE details](#)”
- [26] “[redHat](#)”