Artifact Abstract: Deployment of APIs on Android Mobile Devices and Microcontrollers

Sergio Laso*, Marino Linaje*, Jose Garcia-Alonso*, Juan M. Murillo* and Javier Berrocal*
*University of Extremadura
Caceres, Spain
Email:{slasom, mlinaje, jgaralo, juanmamu, jberolm}@unex.es

***Index Terms*—Microservices, Android, Microcontroller, OpenAPI, Edge Computing**

I. Introduction

This artifact is a guideline for the generation of APIs through the APIGEND (API Generator for End Devices) tool. This tool is an extension of the OpenAPI Generator [1]. It originally allows developers to create both the client and server side through an OpenAPI Specification with a Server-Centric style in different languages. The extension developed also allows one to generate APIs for end devices, specifically for Android devices and ESP32 Microcontrollers, making the application of the Edge [2] and Mobile-Centric [3] paradigms easier.

II. Requirements and Deployment

APIGEND is a web application developed with Spring[1]and uses Mustache's templates[2] for code generation. APIGEND can be deployed locally using the development environment by running the *openapi-generator-online* module as shown in Figure 1, on a Docker[3] container as it has a DockerFile or directly on a cloud environment. This last option has been followed for deploying APIGEND on Heroku[4] and which can be accessed through the following link:

https://openapi-generator-spilab.herokuapp.com

The source code can be dowloaded from the following public Bitbucket repository:

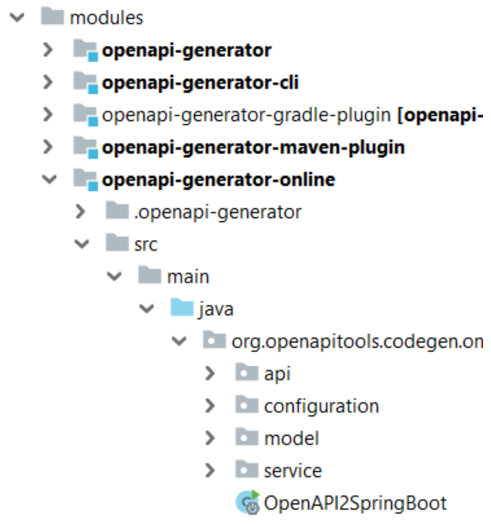
https://bitbucket.org/spilab/openapigenerator

III. Generating APIs

This section shows the complete process to generate APIs, specifically for Android devices with MQTT [4] as communication protocol, although to generate APIs with other languages, the procedure is practically the same.

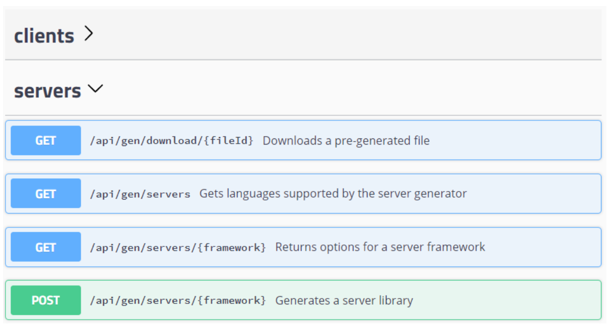
When we access the web interface we have three sections (Gen-Api-Controller , Clients and Servers). We will have to access the Servers section and the endpoint called *"Generates a server library"* as shown in Figure 2.

[1]https://spring.io
[2]https://mustache.github.io
[3]https://www.docker.com
[4]https://heroku.com



```
modules
  openapi-generator
  openapi-generator-cli
  openapi-generator-gradle-plugin [openapi-
  openapi-generator-maven-plugin
  openapi-generator-online
    .openapi-generator
    src
      main
        java
          org.openapitools.codegen.on
            api
            configuration
            model
            service
            OpenAPI2SpringBoot
```

Fig. 1. Module Package.



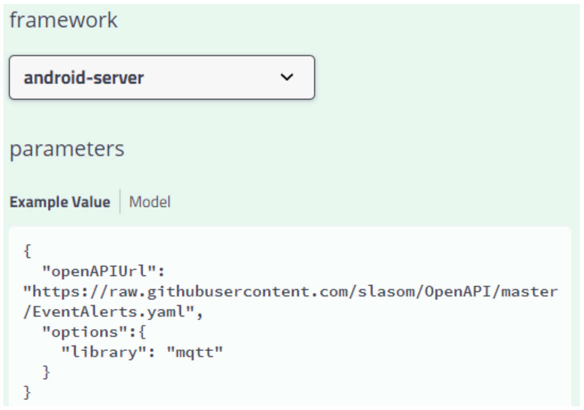
clients
servers

GET	/api/gen/download/{fileId}	Downloads a pre-generated file
GET	/api/gen/servers	Gets languages supported by the server generator
GET	/api/gen/servers/{framework}	Returns options for a server framework
POST	/api/gen/servers/{framework}	Generates a server library

Fig. 2. Web Interface.

To generate the API for an Android device with MQTT, we must select the option *Try it Out*. This will allow us to specify the parameters required for generating the API. Concretely, we have to specify in *framework* the *"android-server"* option. In the *parameters* section, we have to indicate in *"openAPIUrl"* the url of the API specification and in "options" we have to indicate as *library*, *"mqtt"* (if no library is indicated, by default it is generated using Firebase Cloud Messagging [5]). Figure 3 shows an example for the use case that will be followed during the demo.

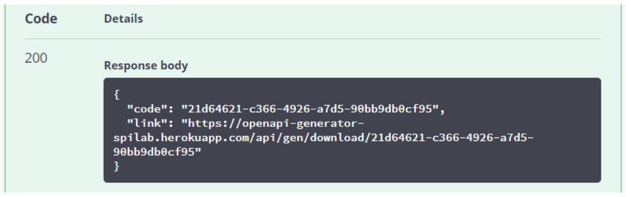
To generate the API, click on the *Execute* button and if



```
{
  "openAPIUrl":
"https://raw.githubusercontent.com/slasom/OpenAPI/master
/EventAlerts.yaml",
  "options":{
    "library": "mqtt"
  }
}
```

Fig. 3. Parameters to generate the Android application using the MQTT library.

there is no error in the specification, we will get as a result a JSON. Inside the JSON, the link option specify the URL for downloading the application as Figure 4 shows. If we copy and paste it into the browser bar, a *.zip* file with the API generated will be downloaded automatically.



```
{
  "code": "21d64621-c366-4926-a7d5-90bb9db0cf95",
  "link": "https://openapi-generator-
spilab.herokuapp.com/api/gen/download/21d64621-c366-4926-a7d5-
90bb9db0cf95"
}
```

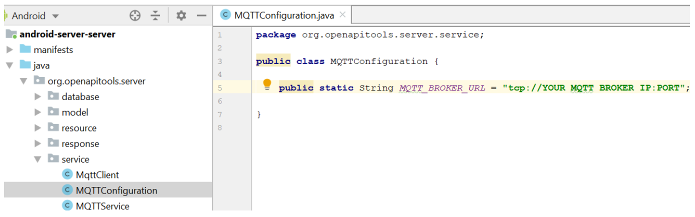
Fig. 4. Correct Generation.

IV. Deploying APIs on end devices

In this section, we will proceed to show how the deployment could be done for Android devices. Having downloaded the .zip file, it have to be unzipped in order to finish the implementation of the different endpoints defined in the specification and also to configure the MQTT communication protocol. To do this, it is necessary to access the *MQTTConfiguration.java* file located in the following path:

```
src/main/java/org/openapitools/server/
service/
```

In the file, we must introduce the IP and the port of the MQTT broker to which we want to connect the API for the reception of requests.



```
package org.openapitools.server.service;

public class MQTTConfiguration {

    public static String MQTT_BROKER_URL = "tcp://YOUR MQTT BROKER IP:PORT";

}
```

Fig. 5. Android Configuration.

V. Requests on end devices

This section explains the structure that has to be followed to invoke an endpoint employing MQTT. The Listing 1 shows the content of a request to one of the endpoints declared in the Event Alerts API[5]. This is the specification that will be used during the defined demo.

The different parameters that have to be specified are:

- **resource**: this parameter corresponds to the names of the *Tags* created in the specification. There are two in this specification, *Event* and *Location*.
- **method**: corresponds to the *operationId* of each endpoint in the specification.
- **sender**: this parameter is automatically included in the Response of the API (it is not necessary to indicate it in the specification), it can be useful to introduce, to which client we want to respond to. In this case, the sender's topic is specified to receive the reply.
- **params**: in this parameter are included the objects or parameters that go in the request. In this method, the specification has a request body with the *Event* object schema as a parameter.

The **MQTT topic** for sending requests to the API is the *title* of the specification without spaces. In this case, *EventAlerts*.

Listing 1. Content of an Event Alerts API request.

```
{
    "resource": "Event",
    "method": "postEvent",
    "sender": "client3248",
    "params": {
        "event": {
          "id": 1,
          "title": 'Football Match!',
          "description": 'Football match at 11:00 in CC',
          "location":{
                "latitude": 38.514377,
                "longitude":-6.844325,
                "radius": 200
                }
            }
        }
 }
```

Acknowledgment

This work was supported by the Innovation and Universities (MCIU) (RTI2018-094591-B-I00) (MCIU/AEI/FEDER, UE), by Government of Extremadura (GR18112, IB18030), and by the European Regional Development Fund.

References

[1] "Openapi Generator." [Online]. Available: https://github.com/OpenAPITools/openapi-generator
[2] K. Dolui and S. K. Datta, "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing," in *2017 Global Internet of Things Summit (GIoTS)*. IEEE, 2017, pp. 1–6.
[3] J. Guillén, J. Miranda, J. Berrocal, J. Garcia-Alonso, J. M. Murillo, and C. Canal, "People as a service: A mobile-centric model for providing collective sociological profiles," *IEEE Software*, vol. 31, no. 2, pp. 48–53, 2014.
[4] "MQTT." [Online]. Available: http://mqtt.org/
[5] "Firebase Cloud Messaging." [Online]. Available: https://firebase.google.com/docs/cloud-messaging

[5]https://raw.githubusercontent.com/slasom/OpenAPI/master/EventAlerts.yaml