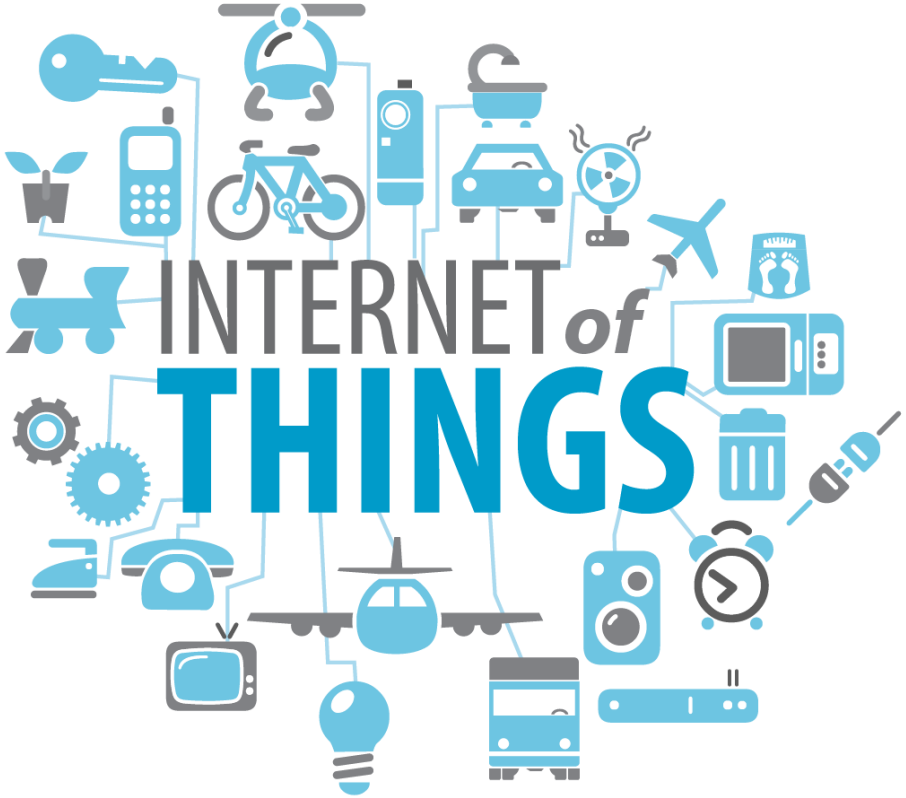
Dispositivos e Infraestructuras para Sistemas Multimedia

Tema 5. Internet of Things



Diseño APIs

- **Diseño de APIs en servicios Rest**
  - Muchas veces por las necesidades (urgentes) se exponen componentes internos de nuestras aplicaciones en servicios REST públicos.
  - La inercia suele llevar a ir creando este tipo de APIs sin un diseño previo.
    - Eso conlleva problemas por la falta de planificación.
    - E inconsistencia entre los objetos y métodos, sin hablar de agujeros de seguridad.
  - La tendencia cambia (SOA):
    - **Cada vez toma más importancia el diseño previo de APIs** utilizando herramientas que tengan en cuenta:
      - La usabilidad.
      - Las necesidades de los consumidores/aplicaciones que vayan utilizar los servicios,
      - Permitir realizar mocks testeables.
      - Posibilitar el versionado.
      - Y, por supuesto, **crear de forma conjunta al desarrollo de la documentación**.

IoT Tecnologías - Diseño APIs

- **Gestión**

```
#%RAML 0.8
---
title: Citizens Location Service
baseUri: http://www.dtic.ua.es/v1.0
version: v1.0

/citizenlocations:
  post:
    description: create locations read from RFID smart sensors
    protocols: [HTTPS]
    body:
      application/json:
        example: |
          {
            "idrfid":1,
            "location":[
                 "lat":38.384993156837425,
                 "lng":-0.5133978999999727
              ],
            "locations":[
                 "citizen":["cid": 101010101,"pw": 64,"ts":1443723690],
                 "citizen":["cid": 101010101,"pw": 62,"ts":1443723695],
                 "citizen":["cid": 203330107,"pw": 62,"ts":1443723695]
              ]
          }
    responses:
      201:
        description: Locations have been successfully created.
        body:
          application/json:
            example: |
              {
                "messeage": "Locations have been successfully created."
              }
      400:
        description: Locations have not been created.
        body:
          application/json:
            example: |
              {
                "message": "Locations have not been created."
              }
```

- **Gestión de APIs en servicios Rest**
  - **API Blueprint**, **RAML** y **Swagger** representan tres excelentes herramientas para diseñar APIs.
    - https://apiblueprint.org/
    - http://swagger.io/
    - http://raml.org/

  - Podemos diseñar sobre el papel antes de su implementación la definición de la API en formato JSON o usando markdown para describir la interfaz, estructura y el modelo de datos.

- **Gestión de APIs en servicios Rest**
  - **API Blueprint**
    - Con API Blueprint tenemos un amplio ecosistema entorno al desarrollo de APIs.
    - Tiene un lenguaje de markdown para escribir la definición y transformarla en JSON.
      - Esto mejora la legibilidad, pensado para humano y no maquinas.
    - Podemos usar Node.JS, .NET o Ruby directamente para realizar el binding con nuestra API.
    - También posibilita el uso de herramientas con Apiary.io para crear documentación interactiva, crear API mocks, validaciones, etc… combinado con Dredd para realizar testing.

- **Gestión de APIs en servicios Rest**


```
# GET /message
+ Response 200 (text/plain)

        Hello World!
```

```
# Blog Posts [/posts]

## Retrieve All Posts [GET]
+ Response 200 (application/json)
    + Attributes (array[Blog Post])
```

- **Gestión de APIs en servicios Rest**
  - **RAML**
    - RAML es la definición de **RESTful API Modeling Language**, el cual permite describir servicios REST de forma completa.
    - Destaca la capacidad de **reutilización** de componentes y patrones para aplicar en las definiciones como “best practices”.
    - Está construido a partir de estándares como YAML y JSON.
    - Cada API está definida con la versión de RAML que está usando junto a una serie de características en su descripción:
      - Título
      - Versión
      - base URI.
    - RAML permite definir patrones minimizando la repetición en las definiciones, usando:
      - *Traits*
      - *resourceTypes*
      - *SecuritySchemes*
    - Podemos definir las respuestas y ejemplos escritos en la definición como documentación.

IoT Tecnologías - Diseño APIs

For every API, start by defining which version of RAML you are using, and then document basic characteristics of your API - the title, baseURI, and version.

Create and pull in namespaced, reusable libraries containing data types, traits, resource types, schemas, examples, & more.

Annotations let you add vendor specifc functionality without compromising your spec

Traits and resourceTypes let you take advantage of code reuse and design patterns

Easily define resources and methods, then add as much detail as you want. Apply traits and other patterns, or add parameters and other details specific to each call.

Describe expected responses for multiple media types and specify data types or call in pre-defined schemas and examples. Schemas and examples can be defined via a data type, in-line, or externalized with !include.

Write human-readable, markdown-formatted descriptions throughout your RAML spec, or include entire markdown documentation sections at the root.

```yaml
#%RAML 1.0
title: World Music API
baseUri: http://example.api.com/{version}
version: v1

uses:
  Songs: !include libraries/songs.raml

annotationTypes:
  monitoringInterval:
   parameters:
     value: integer

traits:
  secured: !include secured/accessToken.raml

/songs:
  is: secured
  get:
    (monitoringInterval): 30
    queryParameters:
      genre:
        description: filter the songs by genre
  post:
  /{songId}:
    get:
      responses:
        200:
          body:
            application/json:
              type: Songs.Song
            application/xml:
              schema: !include schemas/songs.xml
              example: !include examples/songs.xml
```

Songs Library

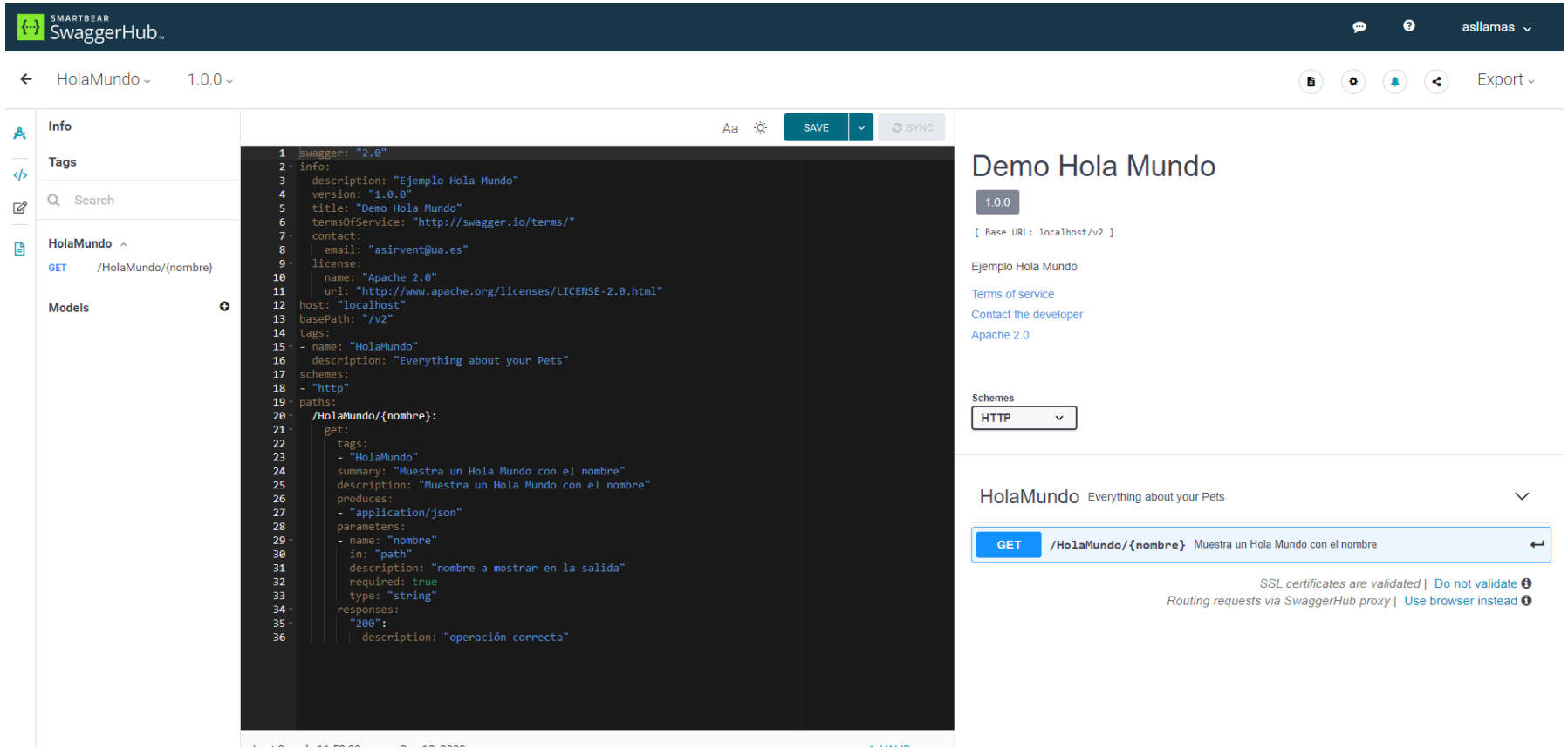
```yaml
#%RAML 1.0 Library
types:
  Song:
    properties:
      title: string
      length: number
  Album:
    properties:
      title: string
      songs: Song[]
  Musician:
    properties:
      name: string
      discography: (Song | Album)[]
```

songs.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
      <xs:element name="song">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="title" type="xs:string">
                        </xs:element>
                        <xs:element name="artist" type="xs:string">
                        </xs:element>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

- **Gestión de APIs en servicios Rest**
  - **swagger**
    - Swagger es un conjunto de herramientas de software de código abierto para diseñar, construir, documentar, y utilizar servicios web RESTful.
    - Las herramientas Swagger como **Swagger Editor** y **SwaggerHub** proporcionan un editor YAML con un panel de visualización para que los desarrolladores trabajen y vean cómo se verá y se comportará la API para el consumidor final.

- **Gestión de APIs en servicios Rest**
  - **swagger**



IoT OpenApi

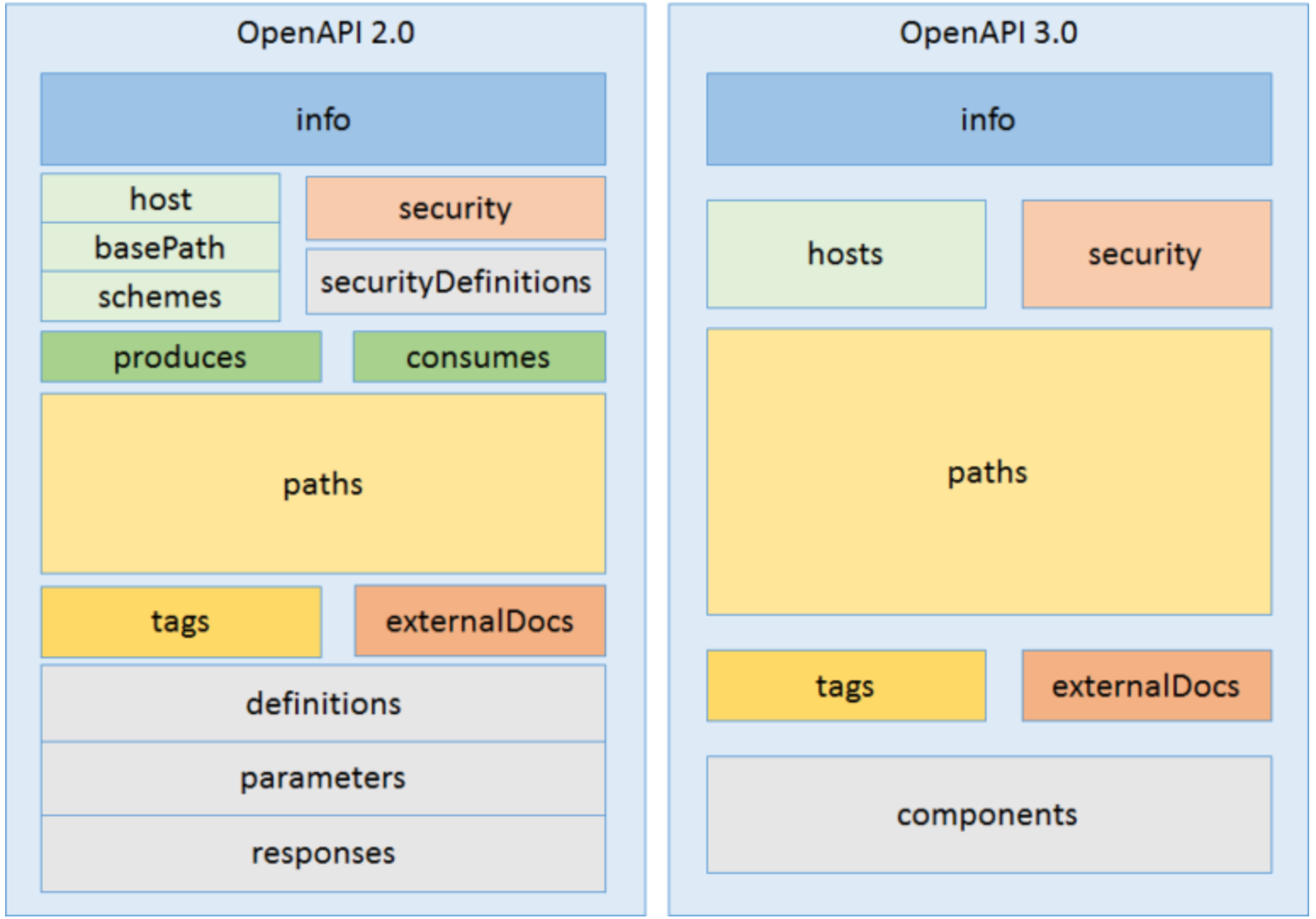
- ¿Cómo debo diseñar mi API?
- ¿Cómo voy a exponerla a distintos desarrolladores?
- Todos los productos necesitan una documentación y las APIs no son la excepción.
- OpenAPI, es un estándar para crear esa documentación para nuestra API.
- OpenAPI se creó para "**Crear un formato de descripción abierto para los servicios API** que sea neutral, portátil y abierto, para acelerar la visión de un mundo verdaderamente conectado."

IoT OpenApi

- Swagger vs OpenAPI
  - OpenAPI = Especificación
    - https://www.openapis.org/
    - El desarrollo de la especificación es fomentado por la Iniciativa OpenAPI (Microsoft, Google, IBM y CapitalOne…)
    - https://swagger.io/specification/
  - Swagger = Herramientas para implementar la especificación
    - https://swagger.io/
    - El conjunto de herramientas Swagger incluye una combinación de herramientas comerciales, gratuitas y de código abierto, que se pueden utilizar en diferentes etapas del ciclo de vida de la API.

- Swagger Herramientas
  - **Swagger Editor :** Swagger Editor le permite editar especificaciones de OpenAPI en YAML dentro de su navegador y obtener una vista previa de la documentación en tiempo real.
  - **Swagger UI :** Swagger UI es una colección de activos HTML, Javascript y CSS que generan dinámicamente una hermosa documentación a partir de una API compatible con OAS.
  - **Swagger Codegen :** permite la generación de bibliotecas cliente API (generación de SDK), stubs de servidor y documentación de forma automática dada una especificación OpenAPI.
  - **Swagger Parser :** biblioteca independiente para analizar definiciones de OpenAPI desde Java.
  - **Swagger Core :** bibliotecas relacionadas con Java para crear, consumir y trabajar con definiciones de OpenAPI.
  - **Swagger Inspector (gratis) :** herramienta de prueba de API que le permite validar sus API y generar definiciones de OpenAPI a partir de una API existente.
  - **SwaggerHub (gratuito y comercial) :** diseño y documentación de API, creado para equipos que trabajan con OpenAPI.

- Swagger 2.0 vs OpenApi 3.0



- OpenApi 3.0 Tutorial
  - Las definiciones de OpenAPI se pueden escribir en JSON o YAML.
    - YAML es más fácil de leer y escribir.
  - Una especificación simple de OpenAPI 3.0:

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Sample API
  description: A sample API to illustrate OpenAPI concepts
paths:
  /list:
    get:
      description: Returns a list of stuff
      responses:
        '200':
          description: Successful response
```

- OpenApi 3.0 Tutorial
  - Diseñaremos una API para un sello discográfico.
  - Partiremos de que el sello discográfico tiene una base de datos de artistas con la siguiente información:
    - Nombre del artista.
    - Género de artista.
    - Número de álbumes publicados bajo la etiqueta.
    - Nombre de usuario del artista.
  - La API permitirá a los consumidores obtener la lista de artistas almacenada en la base de datos y agregar un nuevo artista a la base de datos.

- OpenApi 3.0 Tutorial
  - Una API definida por la especificación OpenAPI se divide en 3 categorías:
    - Meta información
    - Elementos de ruta (endpoints):
      - Parámetros
      - Órganos de solicitud
      - Respuestas
    - Componentes reutilizables:
      - Esquemas (modelos de datos)
      - Parámetros
      - Respuestas
      - Otros componentes

OpenApi 3.0 Tutorial (Meta información)

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple Artist API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

# Basic authentication
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
security:
  - BasicAuth: []

paths: {}
```

OpenApi 3.0 Tutorial (paths)

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
security:
  - BasicAuth: []

#  ----- Added lines  ----------------------------------------
paths:
  /artists:
    get:
     description: Returns a list of artists
#  ---- /Added lines  ----------------------------------------
```

Se define /artists como endpoint y el método GET.
Un cliente usará:
GET https://example.io/v1/artists para obtener una lista de artistas.

OpenApi 3.0 Tutorial (responses)

```
…
paths:
  /artists:
    get:
      description: Returns a list of artists
      #  ----- Added lines  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of artists
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  required:
                    - username
                  properties:
                    artist_name:
                      type: string
                    artist_genre:
                      type: string
                    albums_recorded:
                      type: integer
                    username:
                      type: string

#  ---- /Added lines  ----------------------------------------
```

```
…

        '400':
          description: Invalid request
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
      #  ---- /Added lines  --------------------------------
-------
```

OpenApi 3.0 Tutorial (parameters)

```
…
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple API
  description: A simple API to illustrate OpenAPI concepts



servers:
  - url: https://example.io/v1

components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
security:
  - BasicAuth: []

paths:
  /artists:
    get:
      description: Returns a list of artists
```

```
…

 #  ----- Added lines  ----------------------------------------
      parameters:
        - name: limit
          in: query
          description: Limits the number of items on a page
          schema:
            type: integer
        - name: offset
          in: query
          description: Specifies the page number of the artists to be displayed
          schema:
            type: integer
      #  ---- /Added lines  ----------------------------------------
```

Query Params:
GET https://example.io/v1/artists?limit=20&offset=3

IoT OpenApi

- OpenApi 3.0 Tutorial (parameters)

```
…
paths:
  /artists:
    #  ----- Added lines  ----------------------------------------
  /artists/{username}:
    get:
      description: Obtain information about an artist from his or her unique username
      parameters:
        - name: username
          in: path
          required: true
          schema:
            type: string

      responses:
        '200':
          description: Successfully returned an artist
          content:
            application/json:
              schema:
                type: object
                properties:
                  artist_name:
                    type: string
                  artist_genre:
                    type: string
                  albums_recorded:
                    type: integer
```

```
…
'400':
          description: Invalid request
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
  #  ---- /Added lines  ----------------------------------------
```

- OpenApi 3.0 Tutorial (componentes reutilizables)
  - La Especificación define varios tipos de componentes reutilizables:
    - Schemas (modelos de datos)
    - Parameters
    - Request bodies
    - Responses
    - Response headers
    - Examples
    - Links
    - Callbacks

OpenApi 3.0 Tutorial (Schemas)

```
…
 /artists:
    get:
…
      responses:
        '200':
          description: Successfully returned a list of artists
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ----------------------------------------
                  $ref: '#/components/schemas/Artist'
                  #  ---- /Added line  ----------------------------------------
…
 post:
      description: Lets a user post a new artist
      requestBody:
        required: true
        content:
          application/json:
            schema:
              #  ----- Added line  ----------------------------------------
              $ref: '#/components/schemas/Artist'
              #  ---- /Added line  ----------------------------------------
```

- OpenApi 3.0 Tutorial (Schemas)

```
…
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
 #  ----- Added lines  ----------------------------------------
  schemas:
    Artist:
      type: object
      required:
        - username
      properties:
        artist_name:
          type: string
        artist_genre:
            type: string
        albums_recorded:
            type: integer
        username:
            type: string
  #  ---- /Added lines  ----------------------------------------
```

- OpenApi 3.0 Tutorial (Parametros y Respuestas)
  - La sección ***components*** también tiene subsecciones para almacenar parámetros y respuestas reutilizables.
  - Definimos los parámetros de consulta reutilizables:
    - **Offset y limit** y luego hacemos referencia a ellos desde el endpoint /artists.
    - **400Error** respuesta reutilizable , a la que luego hacemos referencia desde todos los endpoints.

IoT OpenApi

- OpenApi 3.0 Tutorial (Parametros y Respuestas)

```
…
 #  ----- Added lines  ----------------------------------------
  parameters:
    PageLimit:
      name: limit
      in: query
      description: Limits the number of items on a page
      schema:
        type: integer

    PageOffset:
      name: offset
      in: query
      description: Specifies the page number of the artists to be displayed
      schema:
        type: integer

  responses:
    400Error:
      description: Invalid request
      content:
        application/json:
          schema:
            type: object
            properties:
              message:
                type: string
  #  ---- /Added lines  ------------
```

```
…
parameters:
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
…
 '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400Error'
          #  ---- /Added line  ----------------------------------------
…
 items:
                  #  ----- Added line  --------------------------------
                  $ref: '#/components/schemas/Artist'
                  #  ---- /Added line  --------------------------------
```

- OpenApi 3.0 Tutorial
  - https://app.swaggerhub.com/help/tutorials/index
  - https://swagger.io/specification/