Guía para el diseño de APIs

Oficina de Calidad y Arquitectura

Versión: 1.3.1

03/09/2021

Control de versiones

Versión	Fecha	Autor	Descripción / Comentarios
V01_v01	08/05/2019	Oficina de Calidad	Primera versión
V01_v02	02/09/2019	Oficina de Calidad	Eliminación del apartado “5. ROLES EN GESTIÓN DE APIS”
1.3	29/04/2021	Oficina de Calidad y Arquitectura	Modificación de los apartados “3.3.2.1 Identificadores de recursos” y “3.4. DOCUMENTACIÓN DE APIS”
1.3.1	03/09/2021	Oficina de Calidad y Arquitectura	Cambio de nombre del archivo para preparación para Publicación en el metodología.

Índice

1. INTRODUCCIÓN

1.1. OBJETO

Este documento normaliza el diseño de servicios conforme al estilo de arquitectura REST.

Esta normalización persigue los siguientes objetivos:

Homogeneizar el aspecto de todos los Servicios con el objeto de que puedan formar parte del API público del MIR. Los servicios de este API deben verse como un API único y, para ello, deben reunir un mismo conjunto de características que faciliten su consumo a los posibles usuarios.

Facilitar el consumo interno de las distintas áreas de negocio. El aspecto homogéneo agiliza la comprensión del API y simplifica su integración.

Agiliza su diseño al disponer de un conjunto de características predefinidas.

Simplifica su publicación a consumidores, tanto internos como externos al MIR.

Este estilo fue descrito originariamente por Roy Thomas Fielding en su Tesis denominada “Architectural Styles and the Design of Network-based Software Architectures”

https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

1.2. ALCANCE

Este documento está dirigido a las Áreas de Desarrollo y Negocio del Ministerio del Interior.

El documento también es informativo tanto para el Departamento de Calidad como de Sistemas en la iniciativa de maduración del desarrollo de Servicios REST como mecanismo de publicación de API.

El documento presupone que el lector tiene un conocimiento acerca del diseño e implementación de un API REST.

1.3. GLOSARIO DE TÉRMINOS

En este apartado se incluyen, organizados por orden alfabético, los términos que se utilizan en el propio documento para facilitar su lectura y comprensión.

Término	Descripción
API	Application programming interface
CORS	Cross-Origin Resource Sharing
MIR	Ministerio del Interior Gobierno de España
REST	Representational State Transfer
SGTIC	Subdirección General de Tecnologías de la información y las comunicaciones
URL	Localizador Uniforme de Recursos
URI	Identificador Uniforme de Recurso

Tabla 1: Términos y acrónimos

1.4. DOCUMENTOS RELACIONADOS

/Ubicación/Documento	Descripción
<documento1>	
<documento2>	

2. ¿QUÉ ES REST?

2.1. INTRODUCCIÓN

REST es un estilo de arquitectura de software que se puede utilizar para la definición de APIs accesibles en red.

Si se quiere construir un API basado en red es fundamental que se respete la semántica del protocolo de transporte. En este sentido, REST se definió pensando en respetar la semántica proporcionada por el protocolo HTTP/1.1. De esta forma, los agentes e intermediarios involucrados en una comunicación HTTP pueden entender el API sin necesidad de tener conocimiento de la aplicación.

Así, por ejemplo, si se sigue la semántica HTTP existente para un método GET se dispone de los mecanismos de cacheado provistos por el protocolo HTTP. De esta forma, es fundamental tener presente la semántica HTTP a la hora de diseñar nuestro API si se pretende que éste pueda considerarse un API REST.

Esto no excluye que REST pueda ser de aplicación con otros protocolos de transporte, pero si se quiere hacer un uso optimizado de las características proporcionadas por el protocolo, este último debe compartir en gran medida la semántica del protocolo HTTP/1.1.

2.2. CARACTERÍSTICAS DE UN API REST

El principio básico en el diseño de un API REST consiste en definir recursos, identificables mediante un nombre, que puedan ser manipulados usando un conjunto reducido de métodos.

En este sentido, la definición de una API REST implica:

- La identificación de recursos.
- La manipulación de recursos por medio de representaciones de los mismos.
- El intercambio de mensajes auto descriptivos y completos.
- El uso de hipermedia como ingeniería de estado de la aplicación.

2.2.1. Identificación de recursos

Un recurso puede ser cualquier información reconocible, que se puede identificar y reconocer por un nombre.

Se trata de un concepto que se corresponde con un conjunto de entidades. Los valores de dicho conjunto podrían ser representaciones de recursos o identificadores de recursos.

En ningún caso hay que identificar un recurso con el estado que puedan tener esas entidades en un momento dado.

REST usa un identificador de recurso para identificar un recurso involucrado en una interacción a través del API.

REST proporciona una interfaz genérica para acceder y manipular un recurso. Las acciones sobre un recurso se realizan usando una representación del mismo que captura el estado actual o deseado del recurso. Esta representación es intercambiada en la interacción con el API.

La representación es una secuencia de bytes junto con metadatos que describen dicha secuencia de bytes.

Una representación puede consistir en información, metadatos describiendo esa información, y, en ocasiones, metadatos describiendo, a su vez, a los metadatos. Estos últimos suelen tener como funcionalidad la verificación de la integridad del mensaje.

Los datos de control definen el propósito de un mensaje, tal como la acción requerida o el significado de la respuesta. Así, dependiendo de los datos de control, una representación dada podría corresponderse con el estado actual de un recurso, con el estado deseado o, incluso, el valor de algún otro recurso como sucede en una consulta con filtro.

El formato de la información en una representación es conocido como tipo de medio (media type).

La invocación es síncrona, pero tanto los parámetros de entrada como los de salida pueden ser proporcionados mediante 'streaming'.

2.2.2. Manipulación de recursos

Para la manipulación de un recurso se dispone de las acciones básicas tales como la consulta, creación, modificación, y borrado de un recurso.

Estas acciones tienen una correspondencia con los verbos propios del protocolo HTTP/1.1. Otras acciones que pudieran ser necesarias se definirán respetando la semántica proporcionada por el protocolo HTTP/1.1.

2.2.3. Intercambio de mensajes auto descriptivos y completos

El servidor no conserva estado de la interacción ('stateless').

La naturaleza sin estado propia de REST obliga a que los mensajes contengan toda la información necesaria para su interpretación.

El servidor, al recibir una petición, debe poder dar respuesta sin necesidad de disponer de información recopilada en peticiones previas.

2.2.4. Uso de hipermedia como ingeniería de estado de la aplicación

Hipermedia facilita el descubrimiento de recursos a partir de recursos raíz o relaciones entre recursos facilitando la navegación entre los mismos.

Junto con la información propia del recurso se proporciona información que facilita el descubrimiento de otros recursos, así como, los posibles sub-recursos (recursos contenidos en otro recurso) y operaciones disponibles.

3. DIRECTRICES SOBRE SERVICIOS REST

3.1. INTRODUCCIÓN

En este capítulo se establecen una serie de reglas que deben seguirse a la hora de aplicar el estilo de arquitectura REST.

Estas reglas cubren aspectos tan diversos como la denominación de recursos, la definición de esquemas para crear las representaciones de recursos o los verbos HTTP a emplear según la semántica.

3.2. NIVELES DE MADUREZ

En 2008, Leonard Richardson propuso el siguiente modelo de madurez para un API REST:

- **Nivel 0**: Define una única URI y todas las operaciones se implementan con el verbo HTTP POST.
- **Nivel 1**: Define URI independientes para cada recurso.
- **Nivel 2**: Uso de verbos HTTP para definir operaciones sobre recursos.
- **Nivel 3**: Uso de hipermedia (HATEOAS).

Este último nivel 3 se corresponde con un API Restful, tal y como lo define Fielding en su tesis.

Todo API que se publique en la plataforma de API Management del Ministerio del Interior, debe cumplir al menos el nivel 2 de madurez.

3.3. APIS Y VERSIONADO

Los API correspondientes a la organización deben agruparse en productos.

Un **producto** engloba un conjunto de recursos gestionados de forma conjunta. En este sentido, es el producto el que está sujeto a versionado.

Un producto será accesible mediante uno o varios puntos de acceso (‘endpoint’). Estos puntos de acceso tendrán información acerca de la versión del Producto publicado.

Puesto que el nombre completo de un Recurso contiene la denominación del Producto al que pertenece, los nombres de Producto deben ser definidos conforme a las reglas definidas para un segmento de URI (https://tools.ietf.org/html/rfc3986#appendix-A). Al mismo tiempo, el nombre debe componerse con el prefijo ‘api-’, como, por ejemplo, ‘api-apatridas’. Se recomienda no emplear nombres que hagan referencia a la estructura organizativa del MIR sino nombres descriptivos de la funcionalidad proporcionada.

Criterios a la hora de definir productos:

- **Afinidad funcional**. Es recomendable que las entidades de negocio referenciadas en un mismo contexto funcional estén accesibles a través de un mismo punto de acceso.
- **Gestión de cambios**. Es recomendable que las entidades fuertemente acopladas compartan un mismo ciclo de vida que facilite su versionado.

Como ejemplo de un producto se podría considerar el ámbito de los apátridas en el que se podrían encontrar recursos tales como “resolucion” o “instruccion”. El producto se podría llamar ‘api-apatridas’ y un hipotético recurso ‘instruccion’ se denominaría

‘api-apatridas/v1.0/instrucción’.

3.3.1. Versionado

El versionado de un API se realizará mediante el versionado del producto que lo engloba.

El versionado se realizará conforme a la recomendación 'Versionado Semántico 2.0.0' (https://semver.org/lang/es/).

En lo que respecta a esta recomendación, será necesario hacer uso de todo lo referido a la versión 'Major', dejando al propietario del producto la gestión de las versiones 'Minor' y 'Patch'.

3.3.2. Recursos

Los recursos son entidades del negocio.

El API debe estar organizado en torno a recursos. La representación de un recurso debe ser independiente de su implementación interna.

Las entidades deben agruparse en colecciones identificables como recursos.

Los recursos pueden contener sub-recursos, que a su vez pueden ser recursos o una colección.

En resumen, pueden existir tres tipos de recursos: colecciones, recursos (entidades concretas) y sub-recursos (entidades contenidas en otra entidad), todos ellos organizados de forma jerárquica.

3.3.2.1 Identificadores de recursos

Un recurso debe ser identificado mediante una URI (Identificador de Recurso Uniforme).

La URI debe contener el nombre del producto y la versión del mismo en distintos segmentos de la URI. Por ejemplo, 'api-apatridas/v1.0.1/instruccion'

Debe distinguirse entre el nombre del recurso cuando se hace referencia a aquel segmento de la URI que identifica a un recurso, y el identificador del recurso que se corresponde con la URI completa que identifica al recurso.

- Un nombre de recurso debe construirse en base a nombres y no verbos. Los nombres deben ir en minúsculas. Los nombres compuestos deben separarse mediante guiones (Kebab case - https://en.wikipedia.org/wiki/Letter_case#Special_case_styles).
- Los nombres deben estar en español.
- Los nombres deben estar diseñados conforme a las reglas definidas para un segmento de URI (https://tools.ietf.org/html/rfc3986#appendix-A).
- En el caso en que una URI obedezca a un template de URI y maneje variables, éstas irán de acuerdo a las normas establecidas en la especificación https://tools.ietf.org/html/rfc6570#page-14.
  Observación: Esta especificación admite el guion bajo, "_", como separador, pero no admite el guion medio, "-".
- Los nombres de colecciones deben ser nombres en plural.
- El nombre del producto se denominará mediante el prefijo "api-", seguida de un sufijo que indicará el nombre de la api que deberá estar asociado a la funcionalidad genérica de la misma.
- Un identificador de recurso debe construirse en base a la estructura jerárquica en la que se engloba un recurso. Se recomienda que un identificador de recurso esté compuesto a lo sumo de tres niveles de anidamiento en la estructura jerárquica (colección/recurso/sub-recurso). Un nivel de anidamiento mayor genera a futuro problemas para mantener el API. Sería preferible el uso de referencias a recursos, aunque esto implique una petición extra para recuperar el recurso.

Como ejemplos de identificadores de recurso se podría tener algo similar a:

- api-expedientes/v1.0/instructores (colección)
- api-expedientes/v1.0/electronicos (colección)

- api-expedientes/v1.0/electronicos/exp000010 (recurso)
- api-expedientes/v1.0/fisicos/exp000010/apatrida (sub-recurso)
- api-expedientes /v1.0.0/instructores/333333/asignado (sub-recurso)
- api-apatridas/v1.0.1/instrucciones (colección)

Como ejemplos de templates de uri conforme a la especificación rfc6570 se podría tener algo similar a:

- solicitantes/{clave}/dois/{clave_doi}

3.3.2.2 Excepciones

En casos muy excepcionales, se podría considerar el diseño de determinadas operaciones que difícilmente pueden ser asociadas a recursos como verbos HTTP sin la semántica propia de un recurso. Así, por ejemplo, una operación para suma de dos números podría implementarse como un método HTTP GET con la siguiente URI:

- /suma?operando1=99&operando2=1

Referencia:

- no-recurso: (https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design)

3.3.3. Operaciones sobre recursos

Las operaciones sobre recursos se deben expresar mediante alguno de los verbos HTTP disponibles. En función de la semántica asociada a la operación se hará uso de uno u otro verbo.

Cabe mencionar dos características importantes a la hora de clasificar operaciones:

- **Segura**: se dice que una operación es segura si su ejecución no altera el estado del servidor.
- **Idempotente**: se dice que una operación es idempotente si produce los mismos resultados tanto si se ejecuta una vez como si se ejecuta varias veces.

Las operaciones que se pueden realizar sobre los recursos son las siguientes:

- Consulta
- Creación
- Actualización
- Borrado

Existiendo para cada caso diversas posibilidades que se describen a continuación.

3.3.3.1 Consulta

Las operaciones de consulta de un recurso deben implementarse con el verbo HTTP GET.

3.3.3.1.1 Con el verbo HTTP GET

Esta operación es segura e idempotente.

El cuerpo de la respuesta contiene la representación del recurso.

En el caso de recursos de tipo colección, la petición normalmente incluirá un filtro de consulta.

Los atributos usados para componer el filtro de la consulta deben proporcionarse por estos medios:

- Como parte del PATH de la URL.
- Como Parámetros de Query de la URL.

Códigos de respuesta:

- El código HTTP de respuesta correcta será un 200.
- Si el Recurso no existe el servidor devolverá un código HTTP 404(Not Found).

3.3.3.1.2 Excepción mediante el verbo HTTP POST

Como excepción a la regla de usar el verbo GET para realizar una consulta, en ocasiones, pueden encontrarse consultas complejas que requieren de muchos atributos en la definición del filtro de consulta.

La limitación en el tamaño de la URL del protocolo HTTP provoca que no se pueda usar el verbo GET para implementar este tipo de consultas. En estos casos excepcionales, se puede hacer uso del método POST con el objeto de incluir el filtro de la consulta en el cuerpo de la petición HTTP.

Esta circunstancia se considera excepcional y debe ser consultada con la Oficina de Calidad.

3.3.3.2 Creación

Las operaciones de creación de un recurso deben implementarse mediante el verbo POST.

Esta operación NO es ni segura NI idempotente.

3.3.3.2.1 Con identificador proporcionado por Servidor

El servidor asigna una URI al nuevo recurso y retorna esa URI al cliente. Esta operación se aplica sobre la colección a contener el recurso.

El cuerpo de la petición contiene una representación completa del recurso a dar de alta.

El código HTTP de respuesta correcta será un 201(created).

Junto con el código se debe devolver un identificador(URI) del recurso creado. Este identificador se devuelve en la cabecera HTTP "Location".

3.3.3.2.2 Con identificador proporcionado por Cliente

Las operaciones de creación de un recurso, cuando el identificador del Recurso (URI) es proporcionado por el Cliente en la petición HTTP, deben implementarse mediante el método HTTP PUT. Estas peticiones se realizan sobre recursos individuales (no colecciones).

El cuerpo de la petición contiene una representación completa del Recurso. Si un recurso con ese identificador(URI) ya existe, éste es actualizado. De no existir el recurso, éste es creado si el servidor admite esta forma de crear recursos.

El código HTTP de respuesta correcta será un 201(Created).

Junto con el código se debe devolver un identificador(URI) del recurso creado. Este identificador se devuelve en la cabecera HTTP "Location".

Esta implementación de la operación habilita la semántica de creación o actualización en una misma petición (upsert). El código HTTP de retorno permite conocer si el recurso ha sido creado o actualizado.

Si no se desea habilitar tal semántica, se debería implementar la creación mediante el verbo POST incluyendo el identificador(URI) del recurso como un atributo de la representación enviada en el cuerpo de la petición POST. De esta forma se permite que el cliente pueda evitar la creación de recursos donde sólo se desean actualizar los mismos.

3.3.3.3 Actualización

En este contexto se puede distinguir entre la actualización completa de un recurso o la actualización de una parte del mismo.

3.3.3.3.1 Actualización completa

Las operaciones de actualización de todo el contenido de un recurso deben implementarse mediante el verbo HTTP PUT.

Esta operación NO es segura, pero SI idempotente.

Estas peticiones se realizan, salvo excepciones, sobre un recurso individual a actualizar (no colecciones). Como excepción, cabría pensar, por ejemplo, de operación por lote que actualizan un conjunto de recursos con una única operación.

Como resultado, el recurso es sustituido enteramente por el contenido nuevo proporcionado en la petición.

Códigos de respuesta:

- El código HTTP de respuesta correcta será un 200 si la respuesta incluye una representación del Recurso en su cuerpo, o bien, 204(no-content) si la respuesta no incluye representación alguna en su cuerpo (respuesta sin cuerpo).
- Si no es posible realizar la actualización del Recurso, el código HTTP de respuesta debe ser 409(Conflict).

3.3.3.3.2 Actualización parcial

Las operaciones de actualización parcial de un Recurso deben implementarse mediante el verbo HTTP POST. Esta operación es de aplicación sobre recursos que no son de tipo colección.

La limitación existente en el Gateway Corporativo hace inviable el uso del verbo PATCH para implementar este tipo de operaciones.

Esta operación NO es ni segura NI idempotente(atributos no atómicos).

Códigos de respuesta:

- El código HTTP de respuesta correcta será un 200 si la respuesta incluye una representación del Recurso en su cuerpo, o bien, 204(no-content) si la respuesta no incluye representación alguna en su cuerpo (respuesta sin cuerpo).
- Si la petición, aun siendo una petición válida, no puede realizarse se debe devolver el código HTTP 422 (Unprocessable Entity). Este sería el caso, por ejemplo, en el que la aplicación del cambio supusiera dejar el recurso en un estado incorrecto.

3.3.3.4 Borrado

Las operaciones de borrado de un recurso deben implementarse mediante el verbo HTTP DELETE.

Esta operación NO es segura, pero SI idempotente.

Estas peticiones se realizan sobre el recurso individual (recurso o sub-recurso) a borrar. Esta operación no aplicará, salvo en casos excepcionales, sobre un recurso de tipo colección.

El código HTTP de respuesta correcta será 204(no-content). Si no existe el Recurso, el código HTTP de respuesta debe ser 404(Not Found).

Sería recomendable devolver, junto con el código, el identificador(URI) del Recurso. Este identificador se devuelve en la cabecera HTTP "Location".

3.3.3.5 Operaciones asíncronas

Cualquiera de las operaciones antes mencionadas podría implementarse de forma asíncrona si se considera procedente. En estos casos, es la respuesta la que difiere de lo descrito para cada una de estas operaciones.

El código HTTP de respuesta para estas peticiones asíncronas es 202 (Accepted).

Junto con el código se debe devolver un identificador(URI) de un Recurso que permita consultar el estado de la petición asíncrona. Este identificador se devuelve en la cabecera HTTP "Location".

Por ejemplo, el resultado de una petición de borrado podría ser el siguiente:

```
HTTP/1.1 202 Accepted
Location: /api/status/12345
```

Interior HTTP Ejemplo 1. Respuesta DELETE

y una consulta a ese recurso(/api/status/12345):

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "status":"In progress",
    "link": { "rel":"cancel",
"method":"delete",
"href":"/api/status/12345" }
}
```

Interior HTTP Ejemplo 2. Respuesta estado de la petición

3.3.3.6 Otras operaciones

Aquellas operaciones cuya semántica no encaje con la casuística previamente descrita se recomienda seguir los siguientes principios para determinar el verbo HTTP a usar:

- GET: Para operaciones que no impliquen un cambio de estado en el servidor.
- PUT/POST: Para operaciones que impliquen un cambio, si son operaciones idempotentes se implementarán mediante PUT. En caso contrario, se implementarán mediante POST.

3.3.3.7 Negociación de operaciones

El protocolo HTTP habilita un mecanismo para que un cliente pueda determinar los verbos HTTP disponibles para un recurso determinado. Para ello, habilita el verbo HTTP OPTIONS.

El comportamiento habitual de este método consiste en devolver una cabecera 'Allow' con la lista de verbos HTTP disponibles para el recurso especificado en la petición.

Ejemplo de petición:

```
OPTIONS /api/user HTTP/1.1
Host: example.org
```

Interior HTTP Ejemplo 3. Petición de negociación de operaciones

Ejemplo de respuesta:

```
HTTP/1.1 200 OK
Allow: GET, POST
```

Interior HTTP Ejemplo 4. Respuesta de negociación de operaciones

3.3.3.7.1 CORS

Siempre que se habilite CORS para una aplicación, será necesario que se implemente esta operación ya que es invocada por los navegadores como paso previo a invocar un recurso en un dominio distinto al original.

La recomendación de seguridad es no habilitar CORS salvo que sea estrictamente necesario.

La mayor parte de los frameworks de desarrollo habilitan esta operación por configuración sin necesidad de implementarla.

3.3.4. Representación de recursos

La representación de un recurso puede contener el contenido, metadatos y metadatos sobre los metadatos. Desde una misma URI se pueden devolver distintas representaciones de un recurso.

Salvo para determinados recursos que contienen contenido, la mayor parte de ellos sólo contienen metadatos y metadatos sobre los metadatos.

En aquellos casos que contienen contenido, se puede usar el formato o tipo media que sea recomendable cuando se opere sobre una representación que contenga el contenido.

Aquellos recursos cuya representación sólo contiene metadatos y metadatos sobre metadatos, se debe usar JSON (application/json) como formato de representación.

El tipo media debe ser informado mediante la cabecera HTTP 'Content-Type' (Content-Type: application/json). Esto aplica tanto para el formato de las peticiones del cliente como para las respuestas del servidor.

Si el servidor no admite el tipo media usado en la petición, debe devolver el código HTTP 415 (Unsupported Media Type).

3.3.4.1 Negociación de contenido

El protocolo habilita un mecanismo de negociación de formato de contenido que permite a un cliente indicar los formatos de contenido que es capaz de interpretar.

Cuando el cliente envía una petición proporciona una lista priorizada de tipos media como los formatos de contenidos que puede interpretar. El servidor responde usando el primer formato de esa lista que se encuentre en la lista de formatos que el servidor es capaz de generar. Si el servidor no puede generar ninguno de esos formatos, debe devolver el código HTTP 406 Not Acceptable. La lista priorizada es proporcionada por el cliente mediante la cabecera HTTP 'Accept' (Accept: application/json).

La ingeniería REST es la encargada de realizar esta negociación de contenidos. En base a los métodos definidos donde se especifican los formatos de entrada y salida, la ingeniería determina el formato a proporcionar e invoca al método apropiado para generar dicho formato.

3.3.4.2 Códigos de estado de respuesta

Toda respuesta debe proporcionar un código de Estado HTTP informativo del resultado de la petición.

El protocolo HTTP define un catálogo de posibles códigos de estado. Estos códigos se engloban en distintas categorías identificadas por su primer dígito. Así, tenemos categorías de códigos de respuesta correcta (2xx), redireccionamiento (3xx), error de cliente (4xx) y error del servidor (5xx).

Al describir las operaciones sobre recursos se especificaron algunas reglas que definían cuando utilizar algunos de estos códigos. No obstante, para una descripción detallada de todos los códigos de estado HTTP disponibles y su significado, se dispone del siguiente enlace:

https://httpstatuses.com/

3.3.4.3 Tratamiento de errores

Cuando se produce un error de los englobados en las categorías 4xx o 5xx, el cuerpo de la respuesta debe consistir en un **documento** de tipo application/problem+json

(https://tools.ietf.org/html/rfc7807).

La cabecera HTTP 'Content-Type' debe contener el valor indicado: application/problem+json.

El documento de respuesta debe incluir:

- **type**: referencia URI que identifica el tipo de problema. Podría ser publicada como un recurso que describa el tipo de error.
- **title**: un sumario del tipo de error. No debería cambiar entre ocurrencias de un mismo tipo de error salvo los cambios derivados de aplicar la adaptación local (localization) al texto.
- **status**: código de Estado HTTP (número).

Opcionalmente, este documento también puede incluir:

- **detail**: una descripción detallada y legible por el usuario del error específico.
- **instance**: referencia URI que identifica la ocurrencia específica del error.

El documento puede incluir otros metadatos que proporcionen información añadida acerca del error.

Por ejemplo:

```
{
 "type":
"https://example.net/validation-
error",
 "title": "Parámetros
incorrectos.",
 "status":  422
 "invalid-params": [{
  "name": "edad",
  "reason": "debe ser un entero
positivo"
 }, {
  "name": "color",
  "reason": "debe ser rojo o
negro"
 }]
}
```

Interior HTTP Ejemplo 5. Tratamiento de errores, validación de campos

3.3.4.4 Esquemas asociados a la representación de un recurso

3.3.4.4.1 Nombres de campos

Los nombres de campos deben cumplir las siguientes reglas:

- Se debe usar minúsculas.

- Palabras compuestas o frases deben codificarse en formato 'snake case'. Esto es, debe separarse cada palabra simple por un guion bajo ('_'). Por ejemplo, "nombre_completo", "dia_semana".

3.3.4.4.2 Tipos de campos

Se deben seguir las reglas del estándar RFC4627 para valores de tipo primitivo.

(https://tools.ietf.org/html/rfc4627).

Se deberían excluir del documento los **campos nulos** salvo que la semántica del campo requiera distinguir entre un campo no informado y un campo con un valor nulo.

Las **cadenas vacías** no deberían considerarse como campos nulos. Su valor sería "".

En una actualización parcial('patch') podría ser necesario indicar el campo con valor nulo para indicar que se desea borrar el contenido de dicho campo.

3.3.4.4.3 Fechas e instantes de tiempos

Las fechas e instantes de tiempo deben definirse conforme a RFC3339

(http://xml2rfc.ietf.org/public/rfc/html/rfc3339).

Los formatos se corresponden con el estándar ISO 8601.

Para las fechas se debe usar el formato 'ISO8601 full-date' y para instantes de tiempo se debe usar el formato 'ISO8601 date-time' (https://xml2rfc.tools.ietf.org/public/rfc/html/rfc3339.html#anchor14).

	Representación
Fecha	**YYYY-MM-DD**
Instante de Tiempo	**YYYY-MM-DDTHH:MM:SSZ**

Tabla 2: Representación de fechas e instantes de tiempo

En casos excepcionales, se podría hacer uso de la representación con fracción decimal: 'HH:MM:SS,ssZ'

Ejemplos de fechas e instantes de tiempo serían:

- 2018-08-23
- 2018-08-22T12:00:00Z (Hora UTC)
- 2018-08-24T23:30:59+0002

3.3.4.4.4 Duración a intervalos de tiempo

Para describir la duración de un evento se recomienda el uso del estándar ISO 8601. Este estándar establece el siguiente formato:

P[n]Y[n]M[n]DT[n]H[n]M[n]S donde cada elemento tiene la siguiente significación:

- **P**: indicador de duración. Permite identificar el campo como una duración y, así, diferenciarlo de una fecha, instante de tiempo o intervalo.
- **Y**: indicador de Año.
- **M**: indicador de Mes.
- **W**: indicador de Semana.

- **D**: indicador de Día.
- **T**: indicador de Tiempo.
- **H**: indicador de Hora.
- **M**: indicador de Minutos.
- **S**: indicador de Segundos.

Así, por ejemplo, "P3Y6M4DT12H30M5S" representa una duración de 3 años, 6 meses, 4 días, 20 horas, 30 minutos y 5 segundos.

Por otro lado, para describir un intervalo de tiempo se recomienda también el uso del estándar ISO 8601

Este estándar contempla varias opciones:

- **Fecha/Instante inicial y Fecha/Instante Final**: "2007-03-01T13:00:00Z/2008-05-11T15:30:00Z"
- **Fecha/Instante inicial y Duración**: "2007-03-01T13:00:00Z/P1Y2M10DT2H30M"
- **Duración y Fecha/Instante Final**: "P1Y2M10DT2H30M/2008-05-11T15:30:00Z"
- **Duración** solamente. El contexto proporciona la información adicional para determinar el intervalo: "P1Y2M10DT2H30M,"

3.3.4.4.5 Otras recomendaciones

Se recomienda:

- Evitar el uso de identificadores fuertemente acoplados a la implementación, como claves primarias de tablas de BBDD.
- Evitar el uso de abreviaturas salvo que estas sean de uso muy común y, en consecuencia, sean fácilmente reconocibles.

3.3.4.4.6 Denominación de campos más comunes

Esta lista pretende ser una recopilación de nombres de campos a emplear para referenciar conceptos comunes que suelen aparecer de forma recurrente en la definición de los esquemas de un API.

El objeto de recopilar estas denominaciones comunes es contribuir a una homogeneización de los nombres de campos empleados en el diseño de APIs.

Se recomienda el empleo de estos nombres en aquellos esquemas que incorporen el concepto correspondiente.

NOMBRE	NOMBRE DEL RECURSO. Palabra única.
titulo	Nombre del recurso. Puede contener espacios.
descripcion	Descripción breve del recurso.
fecha_creacion	Fecha de Creación del recurso. Es una fecha.
instante_creacion	Instante en el que se creó el recurso. Incluye fecha y hora.
fecha_modificacion	Fecha de última modificación del recurso. Es una fecha.
instante_modificacion	Instante en el que se modificó el recurso por última vez. Incluye fecha y hora.

fecha_baja	**Fecha de eliminación o borrado del recurso. Es una fecha.**
instante_baja	**Instante en el que se eliminó o borró el recurso. Incluye fecha y hora.**

Tabla 3: Denominación de campos más comunes

3.3.5. Búsquedas, filtros y paginación

Las consultas sobre colecciones suelen incluir mecanismos de selección de recursos mediante filtros de consulta y mecanismos de paginación y ordenación del conjunto resultante.

3.3.5.1 Filtros

Las operaciones de consulta definen un filtro a aplicar sobre el conjunto total de recursos que forman parte de la colección.

Este filtro se puede define de dos formas:

- **Mediante segmentos añadidos al 'Path' de la URL**

  Por ejemplo, esta URI: api-apatridas/v1.0.1/resoluciones/tipo/pendientes implementaría una consulta sobre la colección resoluciones aplicando un filtro de forma que la respuesta sólo contendría las resoluciones de tipo 'pendientes'.

  El formato de la URL debe cumplir los requisitos descritos para la definición de un recurso.

- **Mediante parámetros de query de la URL.**

  Por ejemplo, api-apatridas/v1.0.1/resoluciones?tipo=pendientes se comportaría como en el ejemplo definido mediante segmentos del 'Path'. Los campos definidos como parámetros deben seguir el mismo formato que se aplica en la definición de campos de un esquema.

3.3.5.2 Consultas más comunes mediante alias

Aquellas consultas de uso frecuente, es recomendable que sean definidas mediante alias accesibles como un recurso.

Así, por ejemplo, la consulta descrita anteriormente para obtener las resoluciones de los expedientes apatridas pendientes, podría ser implementada definiendo el siguiente recurso 'api-apatridas/v1.0.1/resolucionespend'.

3.3.5.3 Paginación

Las consultas sobre colecciones deben incluir mecanismos de paginación. La única excepción a la paginación sería para colecciones con un número reducido de elementos durante todo su ciclo de vida.

La implementación debe garantizar que cada respuesta contenga como máximo un número de elementos. La implementación debe proporcionar el siguiente mecanismo de configuración de la Paginación:

- Disponer del parámetro de query cuyo nombre es **'limite'**. Este parámetro permitirá, a la hora de realizar una petición, definir el tamaño de página a usar. Este parámetro no podrá tener un valor superior al máximo previamente descrito. Sería recomendable que su valor tuviese también un límite inferior.
- Disponer de un tamaño de página por defecto que aplicará cuando no se informe en la petición el parámetro 'limite'.
- Unido al tamaño de página, la implementación debe proporcionar un mecanismo para solicitar una determinada página del conjunto resultante de la consulta. Para ello, se dispone de dos mecanismos mutuamente excluyentes:

- Un parámetro de query cuyo nombre es **'pagina'** que permite indicar la página solicitada.
- Un parámetro de query cuyo nombre es **'avance'** que permite indicar un número de recursos, tomados desde el inicio, a ignorar previamente a componer la respuesta. Este atributo es conocido normalmente como 'offset'.

3.3.5.4 Ordenación

En consultas sobre colecciones no es obligatorio implementar mecanismos que permitan definir una ordenación específica para el conjunto resultante.

Siendo recomendación, en caso de implementarse este mecanismo, deberían seguirse las siguientes consideraciones:

- Emplear un parámetro de query con nombre **'orden'** para definir en la petición la ordenación deseada.
- Emplear un DSL simple que pueda ser codificado en el parámetro antes referenciado.

El framework Spring, por ejemplo, facilita un mecanismo para proporcionar una lista ordenada de campos por los que ordenar, así como, si el orden requerido para cada campo es ascendente o descendente. Un ejemplo de definición de una ordenación sería:

```
sort=nombre,asc&sort=edad,desc
```

De esta forma se ordenaría por nombre y edad, los nombres de forma ascendente y la edad descendentemente.

3.4. DOCUMENTACIÓN DE APIS

La documentación debe ser conforme a la especificación OpenAPI. La versión que se está utilizando de dicha especificación es la 3.0. Es necesario establecer, de forma consensuada con el área de Arquitectura de la Oficina de Calidad la versión aplicable en cada caso.

La documentación debe incluir los siguientes:

- Información acerca del API (Info Object).
- Recursos y Operaciones (Path Objects, Parameters Definitions Object).
- Códigos de Respuesta, cabeceras y esquemas usados para las Respuestas (Responses Definitions Object).
- Esquemas usados en el API (Definition Objects).
- Requisitos de Seguridad (Security Requirement Object, Security Definitions Object)
- Ejemplos de los valores que puedan tomar los parámetros.
- Ejemplos de los esquemas usados, en peticiones y respuestas.

Se recomienda el uso de elementos reutilizables para definir parámetros, esquemas de objetos y respuestas.

3.4.1. Etiquetado de operaciones

Las operaciones deben disponer de al menos una agrupación funcional que permita la identificación de todas las operaciones pertenecientes a un recurso. El mecanismo proporcionado para este cometido, son las etiquetas (“tags”) definidas a nivel de operación.

Referencia: https://swagger.io/docs/specification/grouping-operations-with-tags/

4. SEGURIDAD

4.1. NIVELES DE SEGURIDAD

Con el objetivo de definir la seguridad a aplicar en el acceso a un API se han de evaluar distintos criterios que permitirán determinar el Nivel de Seguridad requerido para nuestro API.

El Nivel de Seguridad es una categorización de los requisitos de securización del acceso al API. Cada Nivel de Seguridad se implementará mediante una Política de Seguridad predefinida.

A continuación, se describen los criterios que permitirán establecer el Nivel de Seguridad a aplicar en el acceso al API.

4.1.1. Clientes confidenciales o públicos

El tipo de Cliente del API determinará, junto con otros criterios, la seguridad a aplicar a la hora de publicar el API. En este sentido, cabe distinguir aquellos clientes considerados confidenciales de aquellos que son considerados públicos.

- **Cliente confidencial**: es aquel capaz de mantener las claves del Cliente de forma segura. Estos clientes disponen, junto con el identificador de cliente (Client ID) una clave (Client Secret) que les permite autenticarse de cara al proveedor del API. En esta categoría se engloban aquellos clientes que residen en un servidor protegido.
- **Cliente público**: es aquel cliente sin garantía de poder custodiar de forma segura una clave. En esta categoría se engloban aquellos clientes que residen en un entorno no securizado como las aplicaciones de móviles o las aplicaciones que se ejecutan en un navegador (javascript).

4.1.2. Aplicaciones propias vs aplicaciones de terceros

La naturaleza de las aplicaciones clientes del API determinará la interacción de éstas con el API. Para aplicaciones propias y de confianza podría habilitarse un modelo delegado de autorización. En este modelo, la autenticación de la aplicación sería suficiente para facilitar el acceso al API. La aplicación garantiza la autenticación y autorización del usuario final.

Serán necesarios nuevos casos de uso, con las vistas puestas a futuro, en el que intervengan aplicaciones de terceros no confiables que requerirán de una autorización de usuario no delegada sino controlada por los propios mecanismos de seguridad del API.

4.1.3. Tipo de aplicación cliente

El tipo de implementación del cliente condiciona el grado de seguridad que este puede ofrecer en su interacción con el servidor del API. Así, puede haber aplicaciones WEB, aplicaciones móviles nativas, aplicaciones de escritorio o aplicaciones basadas en agente usuario (browser).

Aquellos clientes pensados para ejecutarse en entornos más vulnerables como pueden ser las aplicaciones móviles deben estar sujetos a mecanismos de autenticación y autorización más seguros que aquellos ubicados en entornos protegidos.

4.1.4. Identidad de usuario final o de la aplicación

En función del caso de uso será necesario disponer de la identidad del usuario final a la hora de acceder al API o, en algunos casos, tan sólo disponer de la identidad de la aplicación.

La naturaleza de la información accesible o, el uso o no de un modelo de autorización delegado determinará la necesidad de identificación del usuario final.

4.1.5. Naturaleza de la información

En este sentido, se podrá disponer de:

- Información pública accesible a cualquier usuario.
- Información accesible por organismos de confianza.
- Información accesible de forma particularizada para cada usuario.

En función de la naturaleza de la información que maneje el API, se deberá optar por el mecanismo de seguridad apropiado.

4.2. MECANISMOS DE SEGURIDAD SOPORTADOS POR API MANAGER MIR

A continuación, se describen los mecanismos de seguridad soportados por el API Manager instalado en el MIR y los escenarios para su uso.

Independientemente de estos mecanismos, es obligatorio la protección de la capa de transporte mediante certificados (TLS) y opcionalmente, dependiendo de la sensibilidad de la información, se puede añadir un mecanismo de protección a la API mediante autenticación con certificado cliente

4.2.1. Usuario / password

Este mecanismo soportado por el API Manager del MIR, no estará permitido salvo petición por necesidades de Negocio, y deberá ser previamente valorado por la Oficina de Calidad.

4.2.2. API / Keys

El uso de API Keys estará habilitado para los siguientes escenarios:

- Cliente Confidencial
- Aplicaciones Propias o de terceros cuya naturaleza de la información sea pública y accesible a cualquier usuario o accesible por organismos de confianza (B2B).
- No es necesario conocer con seguridad la identidad del usuario final.

4.2.3. oAuth

El uso del protocolo oAuth 2.0 será de uso obligado en los siguientes escenarios:

- Cliente Público
- Aplicaciones de terceros cuya naturaleza de la información sea accesible de forma particularizada para cada usuario.
- Es necesario conocer con seguridad la identidad del usuario final.

5. NOMENCLATURA, CÓDIGOS Y CABECERAS

5.1. NOMENCLATURA DE RECURSOS

Será la siguiente.

ELEMENTO	NOMENCLATURA
recurso-completo	'/' producto '/' version '/' recurso
producto	'api-' Segmento de URI
versión	major.minor.patch (https://semver.org/lang/es/)
recurso	colección | recurso-simple | subrecurso
colección	Segmento de URI[1]
recurso-simple	colección '/' nombre-recurso
subrecurso	colección '/' nombre-recurso '/' nombre-subrecurso
nombre-recurso	Segmento de URI
nombre-subrecurso	Segmento de URI

Tabla 4. Cuadro resumen de la nomenclatura de recursos

Cabe mencionar dos características importantes a la hora de clasificar operaciones:

- **Segura**: se dice que una operación es segura si su ejecución no altera el estado del servidor.
- **Idempotente**: se dice que una operación es idempotente si produce los mismos resultados tanto si se ejecuta una como si se ejecuta varias veces.

VERBO HTTP	SEGURO	IDEMPOTENTE
GET	SI	SI
POST	NO	NO
PUT	NO	SI
DELETE	NO	SI

Tabla 5. Características de los verbos http

5.2. CÓDIGOS DE RESPUESTA HTTP COMUNES

Son los siguientes.

CÓDIGO	DESCRIPCIÓN	OBSERVACIONES
200	Ok (CORRECTO)	
201	*Created* (CREADO)	Operaciones de creación de recursos
202	*Acepted* (Aceptada)	Operaciones asíncronas.
204	*No Content* (Sin Contenido)	Respuesta correcta pero no devuelve contenido. Operaciones ejecutadas

[1] Segmento de URI: definido en RFC3986 (https://tools.ietf.org/html/rfc3986#appendix-A)

		correctamente y que no devuelven ninguna representación.
400	***Bad Request* (petición incorrecta)**	**La petición es incorrecta: json malformado, objeto no es correcto, parámetros incorrectos, etc.**
401	***Unauthorized* (No autorizado)**	**No dispone de credenciales válidas para ejecutar la operación. Por ejemplo, no está autenticado o credenciales caducadas.**
403	***Forbidden* (Prohibido)**	**No tiene acceso al recurso o no tiene permisos para realizar la operación sobre el recurso.**
404	**Not found (No encontrado)**	**Recurso no existe.**
405	***Method Not Allowed* (Método no permitido)**	**El recurso no implementa dicho método o operación.**
406	***NOT ACCEPTABLE* (no aceptable)**	**El recurso no puede proporcionar una representación válida para el cliente (punto 3.2.4.1 Negociación de contenido)**
409	***Conflict* (conflicto)**	**El estado actual del recurso impide ejecutar la operación.**
500	***Internal Server Error* (Error interno del servidor)**	**Servidor no disponible, error no recuperable.**

Tabla 6. Códigos de respuesta http más comunes

6. REFERENCIAS

- https://www.restapitutorial.com/
- https://martinfowler.com/articles/richardsonMaturityModel.html
- https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
- https://apigility.org/documentation/api-primer/content-negotiation
- https://tools.ietf.org/html/rfc7231#section-4.3.7
- https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- https://apigility.org/documentation/api-primer/content-negotiation
- http://xml2rfc.ietf.org/public/rfc/html/rfc3339
- http://apiux.com/2013/03/20/5-laws-api-dates-and-times/
- http://restcookbook.com/Resources/pagination/
- https://tools.ietf.org/html/rfc5988#section-5.1
- https://swagger.io/docs/specification/2-0
- https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md
- https://semver.org/lang/es/