NORMA ST. 90

RECOMENDACIONES PARA EL TRATAMIENTO Y LA COMUNICACIÓN DE DATOS DE PROPIEDAD INTELECTUAL MEDIANTE INTERFACES DE PROGRAMACIÓN DE APLICACIONES (API) PARA SERVICIOS WEB

Versión 1.1

Revisión aprobada por el Comité de Normas Técnicas de la OMPI (CWS) en su décima sesión, celebrada el 25 de noviembre de 2022

ÍNDICE

ANEXOS

NORMA ST.90

RECOMENDACIONES PARA EL TRATAMIENTO Y LA COMUNICACIÓN DE DATOS DE PROPIEDAD INTELECTUAL MEDIANTE INTERFACES DE PROGRAMACIÓN DE APLICACIONES (API) PARA SERVICIOS WEB

Versión 1.1

Revisión aprobada por el Comité de Normas Técnicas de la OMPI (CWS) en su décima sesión, celebrada el 25 de noviembre de 2022

INTRODUCCIÓN

1. En la presente norma se formulan recomendaciones sobre las interfaces de programación de aplicaciones (API) para facilitar el tratamiento y el intercambio de datos de propiedad intelectual (datos de PI) de manera armonizada a través de Internet.

2. La norma tiene como finalidad:

- garantizar la coherencia mediante el establecimiento de principios uniformes de diseño de servicios web;
- mejorar la interoperabilidad de los datos entre los socios de servicios web;
- fomentar la reutilización a través de un diseño unificado;
- promover la flexibilidad en la asignación de nombres a los datos en todas las unidades operativas por medio de una política de espacio de nombres claramente definida en los recursos de lenguaje extensible de marcado (XML) asociados;
- promover un intercambio de información seguro;
- ofrecer procesos operativos internos apropiados como servicios de valor añadido que puedan ser utilizados por otras organizaciones; e
- integrar sus procesos operativos internos y vincularlos dinámicamente con sus asociados institucionales.

DEFINICIONES Y TERMINOLOGÍA

3. A efectos de lo previsto en la presente norma:

- se entenderá por "protocolo de transferencia de hipertexto (HTTP)" un protocolo de aplicación para sistemas de información distribuidos, colaborativos e hipermedia. HTTP es la base de la comunicación de datos para la *World Wide Web* (WWW), y funciona como un protocolo de petición-respuesta en el modelo de computación orientado al servicio;
- se entenderá por "interfaz de programación de aplicaciones (API)" un conjunto de componentes de software que proporcionan una interfaz reutilizable entre diferentes aplicaciones que pueden comunicarse fácilmente para intercambiar datos;
- se entenderá por "transferencia de estado representacional (REST)" un conjunto de principios de arquitectura en virtud de los cuales se pueden transmitir datos a través de una interfaz normalizada, a saber, HTTP. REST no contiene una capa adicional de mensajería y se centra en las normas de diseño para crear servicios sin estado;
- se entenderá por "protocolo simple de acceso a objetos (SOAP)" un protocolo para enviar y recibir mensajes entre aplicaciones sin tener que afrontar problemas de interoperabilidad. SOAP establece una especificación de protocolo de comunicación estándar (conjunto de normas) para el intercambio de mensajes basado en XML. SOAP utiliza diferentes protocolos de transferencia, como el HTTP y el protocolo simple de transferencia de correo (SMTP). El protocolo estándar HTTP facilita al modelo SOAP la creación de túneles en cortafuegos y *proxies* sin necesidad de modificar el protocolo SOAP;
- se entenderá por "servicio web" un método de comunicación entre dos aplicaciones o dispositivos electrónicos a través de la WWW. Los servicios web son de dos tipos: REST y SOAP;
- se entenderá por "API web RESTful" un conjunto de servicios web basados en el paradigma de arquitectura REST, que suelen utilizar JSON o XML para la transmisión de datos;
- se entenderá por "API web SOAP" un conjunto de servicios web SOAP, que se basan en SOAP y requieren el uso de XML como formato de carga útil;

- se entenderá por "lenguaje de descripción de servicios web (WSDL)" una norma del Consorcio *World Wide Web* (W3C) que se utiliza con el protocolo SOAP para proporcionar una descripción de un servicio web. Incluye los métodos que usa el servicio web, sus parámetros y sus medios de localización, entre otras cosas;
- se entenderá por "lenguaje de modelado de API RESTful (RAML)" un lenguaje que permite a los desarrolladores proporcionar una especificación de su API;
- se entenderá por "especificación de OpenAPI (OAS)" un lenguaje que permite a los desarrolladores proporcionar una especificación de su API;
- se entenderá por "contrato de servicio" (o contrato de servicio web) un documento en el que se describe la forma en que el servicio expone sus capacidades en términos de las funciones y los recursos ofrecidos a otros programas de *software* mediante una API publicada por el servicio; el término "documentación de la API REST" se utiliza de forma intercambiable con "contrato de servicio" para las API web RESTful;
- se entenderá por "proveedor de servicios" un *software* de servicios web que expone un servicio web;
- se entenderá por "consumidor de servicios" la función en tiempo de ejecución que asume un programa informático cuando accede a un servicio y lo invoca. Una vez que el programa envía un mensaje a una capacidad de servicio especificada en el contrato de servicio, al recibir la petición, el servicio comienza a procesarla y puede o no devolver un mensaje de respuesta correspondiente al consumidor de servicios;
- "*camel case*" hace referencia a la convención de nombres *lower camel case* (por ejemplo, applicantName) o a la *upper camel case* (por ejemplo, ApplicantName);
- "*kebab case*" se refiere a la convención de nombres en la que todas las palabras se escriben en minúsculas y se separan con un guion (por ejemplo, a-b-c);
- las "normas abiertas" son normas puestas a disposición del público en general, que se desarrollan (o aprueban) y se mantienen mediante un proceso de colaboración basado en el consenso. Las normas abiertas facilitan la interoperabilidad y el intercambio de datos entre los diferentes productos de servicios y son concebidas para ser adoptadas de forma generalizada;
- el identificador uniforme de recursos (URI) sirve para identificar un recurso y el localizador uniforme de recursos (URL) es un tipo de URI que incluye la dirección en Internet del recurso;
- una "etiqueta de entidad (ETag)" es un identificador opaco asignado por un servidor web a una versión específica de un recurso que se encuentra en un URL. Si la representación del recurso en ese URL cambia, se le asigna un ETag diferente. Los ETag se pueden comparar de forma rápida para determinar si dos representaciones de un recurso son iguales;
- se entenderá por "registro del servicio" un directorio basado en la red que contiene los servicios disponibles;
- "RMM" se refiere al modelo de madurez de Richardson, que mide la madurez de una API REST con una escala de 0 a 3; y
- se entenderá por "versionado semántico" un esquema de versionado en el que el número de versión está compuesto por MAYOR.MENOR.PARCHE. Se podrá lanzar una:

    - versión MAYOR, cuando se realizan cambios de API incompatibles con la versión anterior;
    - versión MENOR, cuando se agrega una funcionalidad compatible con versiones anteriores; o
    - versión PARCHE, cuando se realizan correcciones de errores compatibles con versiones anteriores.

4. De conformidad con las normas de diseño, las palabras clave que figuran a continuación deben interpretarse según lo dispuesto en el párrafo 8 de la Norma ST.96[1] de la OMPI, a saber:

- DEBE: o los términos equivalentes REQUERIDO o DEBERÁ, implica que la definición es una exigencia de la norma;
- NO DEBE: o el equivalente NO DEBERÁ, implica que la definición es una prohibición terminante de la norma;
- DEBERÍA: o el equivalente RECOMENDADO, indica que pueden existir razones válidas para pasar por alto un punto determinado, pero es preciso considerar plenamente las implicaciones que conlleva;
- NO DEBERÍA: o el equivalente NO RECOMENDADO, indica que pueden existir razones válidas para considerar como aceptable o incluso útil una actuación concreta, pero es preciso considerar cuidadosamente las implicaciones que conlleva; y
- PUEDE: o el equivalente OPCIONAL, indica que el punto en cuestión es realmente facultativo, y solo se ofrece como una opción entre muchas otras.

[1] Véase el capítulo *Material de referencia*.

NOTACIONES

Notaciones generales

5. A lo largo del presente documento se utilizan las siguientes notaciones:

- <>: indica un término descriptivo provisional que, en la aplicación, se sustituirá por un valor específico de instancia;
- " ": indica que el texto entre comillas debe usarse de manera literal en la aplicación;
- { }: indica elementos de aplicación facultativa; y
- El tipo de letra `Courier New` indica palabras clave o código fuente.

6. Los URL que figuran en la presente norma son simplemente ejemplos y no están activos.

Identificadores de normas

7. Todas las normas de diseño son normativas. Las normas de diseño se identifican mediante el prefijo [XX-nn] o [XXY-nn].

a) El valor "XX" es un prefijo para clasificar los tipos de normas de la siguiente forma:

- WS para las normas de diseño de las API web SOAP;
- RS para las normas de diseño de las API web RESTful; y
- CS para las normas de diseño de las API web SOAP y de las API web RESTful.

b) El valor "Y" se utiliza solo para las normas de diseño RESTful y proporciona una mayor granularidad en el tipo de respuesta con el que se relaciona la norma:

- G indica que se trata de una norma general aplicable tanto a los formatos de respuesta JSON como XML;
- J indica que es aplicable al formato de respuesta JSON; y
- X indica que corresponde a un formato de respuesta XML.

c) El valor "nn" indica el número siguiente disponible en la serie de un tipo específico de norma. El número no indica la posición de la norma, en particular para una norma nueva. Una norma nueva se colocará en el contexto pertinente. Por ejemplo, el identificador de norma [WS-4] identifica la cuarta norma de diseño de las API web SOAP. La norma [WS-4] puede ser posicionada entre las normas [WS-10] y [WS-11] en lugar de a continuación de [WS-3], si fuera esa la posición más adecuada para esa norma.

d) El identificador de norma de una norma eliminada se mantendrá, pero el texto de la norma será sustituido por "suprimida".

GRADO DE APLICACIÓN

8. La presente norma tiene por objeto orientar a las oficinas de propiedad intelectual (oficinas de PI) y a otras organizaciones que necesiten gestionar, almacenar, procesar, intercambiar y difundir datos de PI mediante una API web. Se pretende que la norma simplifique y agilice el desarrollo de las API web de forma armonizada y mejore la interoperabilidad entre ellas.

9. La norma también tiene por objeto abarcar las comunicaciones entre las oficinas de PI y sus solicitantes o usuarios de datos, así como las comunicaciones entre las propias oficinas de PI mediante conexiones entre dispositivos y entre dispositivos y aplicaciones de software.

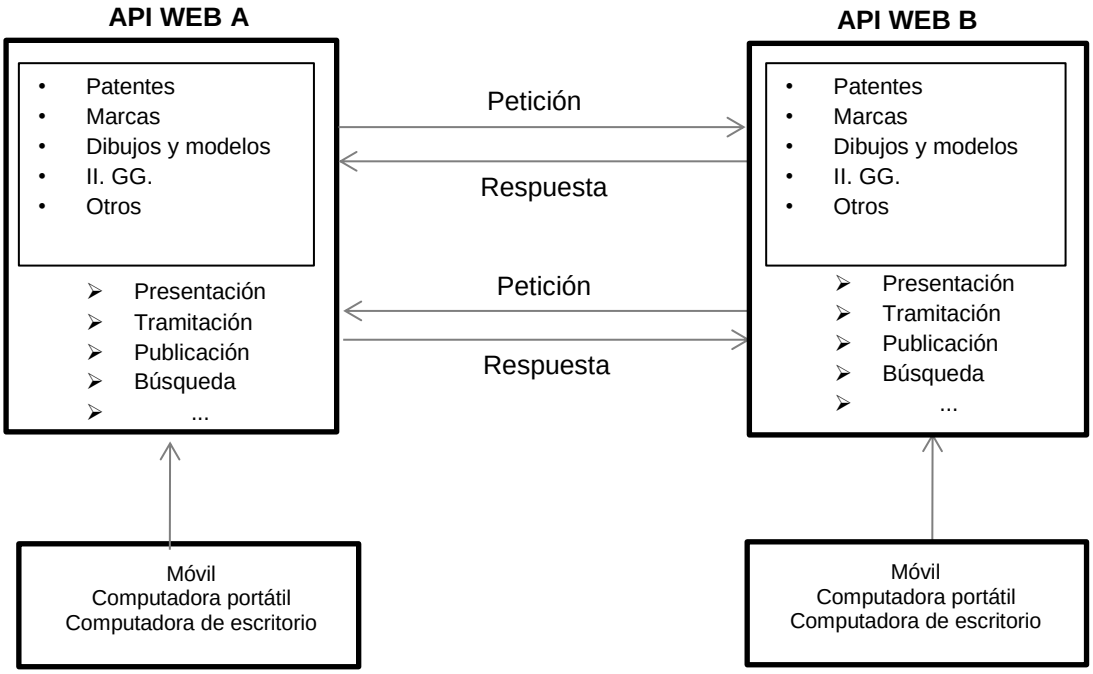


Fig. 1. Grado de aplicación de la norma

10. Asimismo, la presente norma tiene como finalidad proporcionar un conjunto de normas y convenciones de diseño para las API web RESTful y las API web SOAP; una lista de los recursos de datos de PI que se intercambiarán o expondrán; y un modelo de documentación o contrato de servicio de la API, que puede utilizarse para las adaptaciones, en el que se describe el formato de los mensajes, la estructura de datos y el diccionario de datos en formato JSON según la Norma ST.97 de la OMPI y/o XML conforme a la Norma ST.96 de la OMPI.

11. La norma incluye modelos de contratos de servicio para las API web SOAP, que utilizan WSDL, y para las API web RESTful, que utilizan RAML y OAS. En los contratos de servicio también se definen o se mencionan los tipos de datos para las interfaces (véase la sección sobre convenciones de tipos de datos más adelante). En esta norma se recomiendan tres tipos de interfaces: REST-XML (XSD), REST-JSON y SOAP-XML (XSD).

12. La presente norma no obliga a utilizar:

a) pilas tecnológicas de aplicación específicas y productos disponibles en el mercado (COTS);
b) diseños específicos de arquitectura (por ejemplo, arquitectura orientada a servicios (SOA) o arquitectura orientada a microservicios (MOA));
c) algoritmos específicos, como los algoritmos para el cálculo de ETag, es decir, el cálculo de un identificador único para una versión específica de un recurso (por ejemplo, utilizado para el almacenamiento en caché).

PRINCIPIOS DE DISEÑO DE UNA API PARA SERVICIOS WEB

13. Tanto las API web RESTful como las API web SOAP han demostrado que pueden satisfacer las demandas de las grandes organizaciones, así como dar servicio a las pequeñas aplicaciones integradas en la producción. A la hora de elegir entre RESTful y SOAP, se pueden tener en cuenta los siguientes aspectos:

- La seguridad. Por ejemplo, SOAP incluye un protocolo de seguridad en servicios web (WS-Security), mientras que REST no especifica ningún patrón de seguridad.
- Las transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad). Por ejemplo, SOAP cuenta con el soporte de transacción atómica de servicios web (WS-AT), mientras que REST no tiene un soporte adecuado para este tipo de transacciones.

- El tipo de arquitectura. Por ejemplo, la arquitectura de microservicios y sin servidores utiliza REST, mientras que SOA utiliza servicios web SOAP.
- La flexibilidad.
- Las restricciones de ancho de banda.
- La garantía de entrega. Por ejemplo, SOAP ofrece mensajería segura de servicios web (WS-RM), mientras que REST no satisface los requerimientos técnicos necesarios.

14. Cuando se diseña una API web es necesario observar los siguientes principios de diseño orientado a servicios:

- Contrato de servicio estandarizado. La estandarización de los contratos de servicio es el principio de diseño más importante puesto que los contratos permiten regular y dar coherencia al diseño del servicio. Un contrato de servicio debe ser fácil de cumplir y de entender. Un contrato de servicio consta de metadatos que describen la forma en que el proveedor y el consumidor de servicios interactuarán, así como las condiciones en que ambas partes tendrán derecho a interactuar. Se recomienda que los contratos de servicio incluyan:
    - Requerimientos funcionales: información sobre la funcionalidad que proporciona el servicio y los datos que devuelve, o, por lo general, una combinación de ambos elementos.
    - Requerimientos no funcionales: información sobre la responsabilidad de los proveedores de prestar la funcionalidad y/o los datos del servicio, así como la responsabilidad prevista de los consumidores de esos datos y lo que tendrán que ofrecer a cambio. Por ejemplo, la disponibilidad del consumidor, la seguridad y otras consideraciones relativas a la calidad del servicio.
- Servicios débilmente acoplados. Los clientes y los servicios deben evolucionar de forma independiente. La aplicación de este principio de diseño requiere:
    - Versionado de servicios. Los consumidores vinculados a una versión de una API web no deben estar expuestos a interrupciones inesperadas debido a cambios incompatibles de la API.
    - Un contrato de servicio independiente de las especificaciones de la tecnología.
- Abstracción del servicio. Los detalles de implementación del servicio deben estar ocultos. El diseño de la API debe ser independiente de las estrategias compatibles con el servidor. Por ejemplo, para el servicio web REST, el modelo de recursos de la API debe estar desacoplado del modelo de entidad en la capa de persistencia.
- Servicios sin estado. Los servicios deben ser ampliables.
- Reutilización de servicios. Una API bien diseñada debe proporcionar servicios reutilizables con contratos genéricos. La presente norma prevé un modelo de contrato de servicio para ese propósito.
- Autonomía de servicios. Los límites funcionales del servicio deben estar bien definidos.
- Descubrimiento de servicios. Los servicios deben ser descubiertos e interpretados eficazmente.
- Composición de servicios. Los servicios pueden ser parte de otros servicios.
- Utilización de normas como fundamento. Las API deben seguir las normas del mercado (como IETF, ISO y OASIS) según corresponda, y darles prioridad por defecto con respecto a las soluciones optimizadas a nivel local.
- Principio de selección cuidadosa. No es necesario aplicar todas las normas de diseño de la API. Deben elegirse en función de cada caso concreto.

15. Además, sobre todo en lo que respecta a las API web RESTful, deben observarse los siguientes principios:

a) Cualidad de poder ser almacenadas en memoria caché. Las respuestas indican explícitamente si pueden ser almacenadas en memoria caché.
b) Identificación de recursos en las peticiones. Los recursos individuales se identifican en las peticiones, por ejemplo, mediante el uso de los URI en los servicios web REST. Los propios recursos son conceptualmente separados de las representaciones que se devuelven al cliente.
c) Hipermedia como el motor del estado de la aplicación (HATEOAS). Tras haber accedido a un URI inicial para la aplicación REST, como cuando una persona accede a la página de inicio de un sitio web, un cliente de REST debería poder utilizar los enlaces proporcionados por el servidor de forma dinámica para descubrir todas las acciones y los recursos de que dispone.

d) Manipulación de recursos a través de representaciones. Cuando un cliente tiene una representación de un recurso, incluido cualquier metadato adjunto, dispone de suficiente información para modificar o eliminar el recurso.
e) Mensajes autodescriptivos. Cada mensaje incluye metadatos suficientes para describir cómo procesar su contenido.
f) Las API web deben seguir la semántica de HTTP, incluidos sus métodos y errores.
g) Disponible al público. La API se diseña para que sea posible acceder a ella desde la Internet pública si más adelante se desea, aunque no sea la intención inicial.
h) Autenticación común. Conviene utilizar un patrón común de autenticación y autorización, preferiblemente basado en los componentes de seguridad existentes, a fin de no tener que buscar una solución a medida para cada API.
i) Mínimo privilegio. Debe asignarse a los consumidores de la API el mínimo grado de acceso y autorización necesario para realizar las funciones solicitadas.
j) Máxima entropía. Conviene que la aleatoriedad de las credenciales de seguridad sea la máxima posible mediante el uso de claves de API en lugar de nombres de usuario y contraseñas a efectos de autorización de la API, ya que las claves de API proporcionan una superficie de ataque más desafiante para los posibles ciberdelincuentes.
k) Rendimiento frente a seguridad. Conviene alcanzar un equilibrio entre el rendimiento y la seguridad en lo que respecta a la vida útil de las claves y los gastos generales de cifrado y descifrado.

API WEB RESTFUL

16. Una API web RESTful permite a los sistemas de petición acceder y manipular representaciones textuales de los recursos web mediante un conjunto uniforme y predefinido de operaciones sin estado.

Componentes de los URI

17. Las API web RESTful utilizan los URI para direccionar los recursos. De conformidad con la Norma RFC 3986, la sintaxis de los URI es la siguiente:

URI = <esquema> "://" <autoridad> "/" <ruta> {"?" consulta}

autoridad = {infousuario@}anfitrión{:puerto}

Por ejemplo, https://wipo.int/api/v1/patents?sort=id&offset=10

```
  _____/_____/___________/__________________/
    |      |         |          |
esquema autoridad ruta parámetros de consulta
```

18. En la ruta del URI se utiliza la barra, "/", para indicar la relación jerárquica entre los recursos. No obstante, la ruta no debe terminar con una barra, ya que esta no proporciona ningún valor semántico y puede llevar a confusión.

[RSG-01] DEBE usarse la barra, "/", en la ruta del URI para indicar la relación jerárquica entre los recursos, pero la ruta NO DEBE terminar con una barra.

19. En las URI se distingue entre mayúsculas y minúsculas, excepto en el caso del esquema y el anfitrión. Por ejemplo, mientras que https://wipo.int/api/my-resources/uniqueId y https://wipo.INT/api/my-resources/uniqueId son el mismo URI, https://wipo.int/api/my-resources/uniqueid es diferente. En cuanto a los nombres de los recursos, las convenciones *kebab case* y *lower camel case* garantizan una buena legibilidad y asignan los nombres a las entidades en los lenguajes de programación con una simple transformación. Para los parámetros de consulta debe usarse la convención *lower camel case*. Por ejemplo, https://wipo.int/api/v1/inventors?firstName=John. Los nombres de los recursos y los parámetros de consulta distinguen mayúsculas de minúsculas y pueden abreviarse.

20. Una API web RESTful puede tener argumentos:

- en el parámetro 'consulta'. Por ejemplo, /inventors?id=1;
- en el parámetro 'ruta' del URI. Por ejemplo, /inventors/1; y
- en la carga útil de una petición JSON.

21. Salvo los tipos de argumento referidos, que forman parte del URI, un argumento puede formar parte también de la carga útil de la petición.

[RSG-02] Los nombres de los recursos DEBEN seguir un patrón de nomenclatura coherente.

[RSG-03] DEBERÍA utilizarse la convención *kebab case* para los nombres de los recursos en la petición, que PUEDEN ser abreviados.

[RSG-04] Los parámetros de consulta DEBEN seguir un patrón de nomenclatura coherente.

[RSG-05] DEBERÍA utilizarse la convención *lower camel case* para los parámetros de consulta, que PUEDEN ser abreviados.

22. Un punto final de una API web debe ajustarse a la Norma RFC 3986 del Grupo de Tareas sobre Ingeniería de Internet (IETF) y evitar posibles conflictos con los URL de las páginas del sitio web alojado en el dominio raíz. Una API web requiere un punto de entrada exacto para consolidar todas las peticiones. En general, hay dos patrones para definir los puntos finales:

- como el primer segmento de la ruta del URI. Por ejemplo: https://wipo.int/api/v1/; y
- como subdominio. Por ejemplo: https://api.wipo.int/v1/.

[RSG-06] El patrón del URL para una API web DEBE contener la palabra ‘api’ en el URI.

23. Los parámetros matriciales son indicativos de una API compleja con múltiples niveles de recursos y subrecursos, lo que se opone a los principios de diseño orientado a servicios anteriormente definidos. Además, los parámetros matriciales no son estándares, ya que se aplican a un elemento concreto de la ruta, mientras que los parámetros de consulta afectan a la petición en su conjunto. Un ejemplo de parámetros matriciales es el siguiente: `https://api.wipo.int/v1/path;param1=value1;param2=value2`.

[RSG-07] NO DEBEN utilizarse parámetros matriciales.

Códigos de estado

24. Una API web debe aplicar sistemáticamente códigos de estado HTTP según se describe en los documentos RFC del IETF. Deben utilizarse los códigos de estado HTTP que figuran en la lista de códigos de estado HTTP estándares (Norma RFC 7807) incluida en el anexo V.

[RSG-08] Una API web DEBE aplicar sistemáticamente códigos de estado HTTP según se describe en los documentos RFC del IETF.

[RSG-09] Las API web DEBERÍAN utilizar los códigos recomendados en el Anexo V para clasificar los errores.

Principio de selección cuidadosa

25. Un contrato de servicio debería ser flexible con respecto a parámetros inesperados (en la petición, mediante los parámetros de consulta), y generar un error en caso de que los valores de los parámetros esperados no estén bien definidos.

[RSG-10] Si la API detecta valores de entrada incorrectos, DEBE devolver el código de estado HTTP `400 Bad Request`. La carga útil de error DEBE indicar el valor incorrecto.

[RSG-11] Si la API detecta nombres sintácticamente correctos de argumentos no esperados (en la petición o en los parámetros de consulta), DEBERÍA ignorarlos.

[RSG-12] Si la API detecta valores válidos que requieren funcionalidades no implementadas, DEBE devolver el código de estado HTTP `501 Not Implemented`. La carga útil de error DEBE indicar el valor sin implementar.

Modelo de recursos

26. Un modelo de datos de PI debe dividirse en contextos delimitados conforme a un enfoque de diseño basado en el dominio. Cada contexto delimitado debe ser asignado a un recurso. De acuerdo con los principios de diseño, el modelo de recursos de una API web debe estar desvinculado del modelo de datos. Una API web debe diseñarse como una jerarquía de recursos destinada a aprovechar la naturaleza jerárquica del URI para establecer una estructura (asociación, composición o agregación), en la que cada nodo sea un recurso simple (único) o un conjunto de recursos.

27. En ese modelo jerárquico de recursos, los nodos de la raíz se denominan 'nodos de nivel superior' y los recursos anidados 'subrecursos'. Los subrecursos deben utilizarse únicamente para composiciones, es decir, recursos que no pueden ser de nivel superior, ya que, de lo contrario, habría múltiples formas de recuperar las mismas entidades. Esos subrecursos, que implican asociación, se denominan subcolecciones. Las demás estructuras jerárquicas, a saber, la asociación y la agregación, deberían evitarse a fin de que no se creen API complejas ni se dupliquen las funciones.

28. El punto final siempre determina el tipo de respuesta. Por ejemplo, el punto final `https://wipo.int/api/v1/patents` siempre devuelve respuestas relativas a los recursos sobre patentes. El punto final `https://wipo.int/api/v1/patents/1/inventor` siempre devuelve respuestas relativas a los recursos del inventor. Sin embargo, el punto final `https://wipo.int/api/v1/inventors` no está permitido porque los recursos del inventor no pueden ser independientes.

29. Solo deben utilizarse recursos de nivel superior, es decir, con un nivel como máximo, puesto que, de otro modo, la implementación de esas API sería muy compleja. Por ejemplo, debería utilizarse `https://wipo.int/api/v1/patents?inventorId=12345` en lugar de `https://wipo.int/api/v1/inventors/12345/patents`.

[RSG-13] Una API web DEBERÍA utilizar únicamente recursos de nivel superior. Si hay subrecursos, deben ser colecciones e implicar una asociación. Una entidad debe ser accesible como recurso de nivel superior o como subrecurso, pero no por ambas vías.

[RSG-14] Si un recurso puede ser independiente, DEBE ser un recurso de nivel superior; de lo contrario será un subrecurso.

[RSG-15] Para recuperar recursos anidados, DEBEN utilizarse parámetros de consulta en lugar de rutas URL.

30. Las API web pueden ser[2]: API web CRUD (Crear, Leer, Actualizar y Borrar) y API web Intent. Las API web CRUD modelan los cambios realizados en un recurso, es decir, las operaciones de crear, leer, actualizar y borrar. Las API web Intent, en cambio, modelan operaciones administrativas, como renovar, registrar o publicar. Para las operaciones CRUD deben utilizarse sustantivos y para los nombres de los recursos las API web Intent deben utilizar verbos. Aunque las API web CRUD son las más utilizadas, se pueden combinar ambos tipos de API. Por ejemplo, el consumidor de servicios podría utilizar una operación administrativa modelada con una API web Intent, que prepararía la ejecución de una o más operaciones de servicio de API web CRUD. Con las API web CRUD el solicitante de servicios tiene que preparar los algoritmos funcionales, mientras que con las API web Intent es el proveedor de servicios el que prepara esos algoritmos. Las API web CRUD, al contrario que las API web Intent, no son de diseño atómico[3].

– Por ejemplo, si el propietario de algunas marcas desea renovar los registros que caducarán pronto (el dd-mm-aaaa), será necesaria una combinación de las siguientes operaciones administrativas:

[2] Otra posibilidad es clasificar las API según su arquetipo. Véase, por ejemplo: *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*.

[3] Una API web Intent también permite la aplicación del patrón de Segregación de Responsabilidad de Consulta de Comando (CQRS). CQRS es un patrón que permite utilizar un modelo para actualizar la información y otro diferente para leerla. En muchos casos, sobre todo en dominios más complicados, el hecho de utilizar el mismo modelo conceptual para los comandos y las consultas deriva en un modelo más complejo que no resulta útil.

- recuperar las marcas cuyo registro expira el dd-mm-aaaa; y
- renovar el registro internacional de las marcas recuperadas.

En el caso de las API web CRUD, las operaciones administrativas anteriores se modelarían mediante una instrucción no atómica, lo que requeriría dos acciones como las siguientes:

Paso 1. Obtener en formato XML[4] todas las marcas pertenecientes a John Smith y cuyo registro caducará, por ejemplo, el 31-12-2018:

```
GET /api/v1/trademarks?holderFullName=John%20Smith&expiryDate=2018-12-31. HTTP/1.1
Host: wipo.int
Accept: application/xml
```

Se devuelve, por ejemplo, la siguiente respuesta HTTP:

```
HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<tmk:TrademarkBag xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:tmk="http://www.wipo.int/standards/XMLSchema/ST96/Trademark"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Trademark
TrademarkBag.xsd">
      <tmk:Trademark xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:tmk="http://www.wipo.int/standards/XMLSchema/ST96/Trademark"
com:operationCategory="Delete"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Trademark
Trademark.xsd">
            ...
            <com:RegistrationNumber>
                  <com:IPOfficeCode>IT</com:IPOfficeCode>

      <com:ST13ApplicationNumber>000000000000001</com:ST13ApplicationNumber>
            </com:RegistrationNumber>
            ...
            <com:ExpiryDate>2018-12-31</com:ExpiryDate>
            ...
      </tmk:Trademark>
      ...
</tmk:TrademarkBag>
```

Paso 2. Presentar una petición de renovación del registro de cada marca recuperada en el paso anterior (se muestra solo la primera petición de renovación):

```
POST /api/v1/trademarks/renewalRequests HTTP/1.1
Host: wipo.int
Accept: application/xml
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<tmk:MadridRenewal xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
```

[4] No se presenta un ejemplo correspondiente al formato JSON, dado que la instrucción no está codificada para devolver un formato específico.

```
xmlns:tmk="http://www.wipo.int/standards/XMLSchema/ST96/Trademark"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Trademark
MadridRenewal.xsd">
      ...
      <com:InternationalRegistrationNumber>000000000000001</com:InternationalRegist
rationNumber>
      ...
</tmk:MadridRenewal>
```

– El ejemplo anterior también podría modelarse mediante una API web Intent, con una instrucción atómica de petición de servicio como la siguiente[5]:

```
POST /api/v1/trademarks/findAndRenew?holderFullName=john%20smith&expiryDate=2018-
12-31
Host: wipo.int
```

31. El tipo de API web debería imponer restricciones sobre la forma en que se nombran los recursos para indicar cuál se está utilizando. Conviene tener en cuenta que los nombres de los recursos podrán estar en otros idiomas si necesitan ser traducidos por exigencias comerciales.

[RSG-16] Los recursos DEBERÍAN nombrarse con sustantivos para las API web CRUD y con verbos para las API web Intent.

[RSG-17] Si el nombre del recurso es un sustantivo, DEBERÍA usarse siempre la forma plural. NO DEBERÍAN utilizarse formas sustantivas irregulares. Por ejemplo, en lugar de /gente debería usarse /personas.

[RSG-18] Los nombres de los recursos, los segmentos y los parámetros de consulta DEBEN estar compuestos por palabras en inglés, conforme a la grafía del inglés básica del Diccionario de Oxford (Oxford English Dictionary). PUEDEN estar en otros idiomas los nombres de recursos que deben traducirse por razones comerciales.

Compatibilidad con múltiples formatos

32. Los distintos consumidores del servicio pueden tener requisitos diferentes en cuanto al formato de los datos de las respuestas del servicio. El tipo de medio de los datos no debería depender de los propios datos, a fin de que el servicio sea compatible con diversos tipos de medios. Además, una API web debe ser compatible con la negociación del tipo de contenido mediante el encabezado de petición HTTP `Accept` y el encabezado de respuesta HTTP `Content-Type`, tal y como exige la Norma RFC 7231 del IETF. Por ejemplo, para solicitar datos en formato JSON el encabezado `Accept` debe ser `Accept: application/json`, y para solicitar datos en formato XML `Accept:application/xml`. Lo mismo se aplica al encabezado `Content-Type`. Una API web puede ser compatible también con otras formas de negociación del tipo de contenido, como el parámetro de consulta (por ejemplo, `?format`) o el sufijo del URL (por ejemplo, `.json`).

[RSG-19] Una API web DEBERÍA utilizar para la negociación del tipo de contenido el encabezado de petición HTTP `Accept` y el encabezado de respuesta HTTP `Content-Type`.

33. Las API deben ser compatibles con los formatos XML y JSON. En el caso del formato XML, las respuestas deben ajustarse a las normas de la OMPI relativas al formato XML, como la Norma ST.96, y en el caso del formato JSON, las respuestas deben ajustarse a la Norma ST.97 de la OMPI. Debe haber una correspondencia coherente entre los dos tipos de formato.

[RSG-20] Una API web DEBE ser compatible con la negociación del tipo de contenido conforme a la Norma RFC 7231 del IETF.

[5] Se ha eliminado el elemento InternationalRegistrationNumber en la carga útil para referirse a todos los números de registro internacional. No debe utilizarse la ST.96 ni flexibilizarse, si bien el ejemplo presentado amplía los casos de uso permitidos por la ST.96.

[RSG-21] DEBE utilizarse el formato JSON cuando no se solicita un tipo de contenido específico.

[RSG-22] Una API web DEBERÍA devolver el código de estado `406 Not Acceptable` si no es compatible con el formato solicitado.

[RSG-23] Una API web DEBERÍA rechazar las peticiones que contengan encabezados de tipo de contenido inesperado o no aceptable y devolver el código de estado HTTP `406 Not Acceptable` o `415 Unsupported Media Type`.

[RSG-24] Las peticiones y respuestas (convención de nombres, formato del mensaje, estructura de datos y diccionario de datos) DEBERÍAN ajustarse a la Norma ST.96 para XML o la Norma ST.97 para JSON.

[RSJ-25] Los nombres de las propiedades de los objetos JSON DEBERÍAN proporcionarse en la convención *lower camel case* (ejemplo: applicantName).

[RSX-26] Los nombres de los componentes XML DEBERÍAN proporcionarse en la convención *upper camel case*.

[RSG-27] Una API web DEBE ser compatible como mínimo con el formato XML o JSON.

Métodos HTTP

34. Los métodos HTTP (o verbos HTTP) son un tipo de función proporcionada por un contrato uniforme para el procesamiento de los identificadores y datos de recursos. Los métodos HTTP deben utilizarse para lo que fueron diseñados, conforme a la semántica normalizada especificada en las normas RFC 7231 y RFC 5789 del IETF, a saber:

- `GET` – recuperar datos
- `HEAD` – similar a `GET` pero sin una carga útil de respuesta
- `POST` – enviar datos nuevos
- `PUT` – actualizar
- `PATCH` – actualizar parcialmente
- `DELETE` – borrar datos
- `TRACE` – realizar pruebas de eco
- `OPTIONS` – consultar los verbos compatibles con el servidor para un determinado URL

35. El contrato uniforme establece un conjunto de métodos para ser utilizados por los servicios dentro de una determinada colección o inventario. Los métodos HTTP de creación de túneles pueden ser útiles cuando los encabezados HTTP son rechazados por algunos cortafuegos.

36. Los métodos HTTP pueden seguir el principio de selección cuidadosa, que establece que solo se debe implementar la funcionalidad necesaria para el escenario de uso previsto. Algunos *proxies* solo son compatibles con los métodos `POST` y `GET`. Para sortear esa limitación, las API web pueden utilizar un método `POST` con un encabezado HTTP personalizado para crear un túnel sobre el método HTTP que se quiera usar.

[RSG-28] Los métodos HTTP DEBEN limitarse a los métodos estándares `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` y `HEAD`, conforme a las especificaciones de las normas RFC 7231 y RFC 5789 del IETF.

[RSG-29] Los métodos HTTP PUEDEN seguir el principio de selección cuidadosa, que establece que solo se debe implementar la funcionalidad necesaria para el escenario de uso previsto.

[RSG-30] Algunos *proxies* solo son compatibles con los métodos `POST` y `GET`. Para sortear esa limitación, las API web pueden utilizar un método `POST` con un encabezado HTTP personalizado para crear un túnel sobre el método HTTP que se quiera usar. DEBERÍA utilizarse el encabezado HTTP personalizado `X-HTTP-Method`.

[RSG-31] Si un método HTTP no es admitido, DEBERÍA devolverse el código de estado HTTP `405 Method Not Allowed`.

37. En algunos casos de uso, se debería poder admitir múltiples operaciones a la vez.

[RSG-32] Una API web DEBERÍA admitir operaciones por lotes (también conocidas como operaciones masivas) en lugar de múltiples peticiones individuales con el fin de reducir la latencia. Se debería utilizar la misma semántica para los métodos HTTP y los códigos de estado HTTP. La carga útil de respuesta DEBERÍA contener información sobre todas las operaciones por lotes. Si se producen múltiples errores, la carga útil de error DEBERÍA contener información sobre todas las incidencias (en el atributo `details`). Todas las operaciones masivas DEBERÍAN ser ejecutadas mediante una instrucción atómica.

GET

38. Según la Norma RFC 2616 del IETF, el protocolo HTTP no establece a priori ningún límite para la longitud de un URI. Por otra parte, los servidores deberían evitar depender de longitudes de URI que excedan los 255 bytes, ya que algunas implementaciones de clientes o *proxies* más antiguos quizá no admitan correctamente esas longitudes. En caso de que se supere dicho límite, se recomienda utilizar consultas con nombre. Otra posibilidad es especificar un conjunto de normas que establezcan la forma de cambiar de `GET` a `POST`. De conformidad con la Norma RFC 2616 del IETF, una petición `GET` debe ser idempotente, lo que significa que su respuesta será la misma con independencia del número de veces que se ejecute la petición.

[RSG-33] Para un punto final que extrae un solo recurso, si este no se encuentra, el método `GET` DEBE devolver el código de estado `404 Not Found`. En el caso de los puntos finales que devuelven listas de recursos, simplemente devolverán una lista vacía.

[RSG-34] Si un recurso es recuperado con éxito, el método `GET` DEBE devolver `200 OK`.

[RSG-35] Las peticiones `GET` DEBEN ser idempotentes.

[RSG-36] Cuando la longitud del URI excede los 255 bytes, DEBERÍA utilizarse el método `POST` en lugar del método `GET` debido a las limitaciones de este, o bien crear consultas con nombre si es posible.

HEAD

39. Los clientes pueden utilizar `HEAD` cuando necesiten información sobre una operación. Utilizando `HEAD` se obtiene el mismo encabezado HTTP que se obtendría si se hiciera una petición `GET`, pero sin el cuerpo. Esto permite al cliente conocer la información almacenada en caché, así como el tipo de contenido y el código de estado que se devolverá. Una petición `HEAD` DEBE ser idempotente según la Norma RFC 2616 del IETF.

[RSG-37] Las peticiones `HEAD` DEBEN ser idempotentes.

[RSG-38] Algunos *proxies* solo son compatibles con los métodos `POST` y `GET`. Una API web DEBERÍA admitir un encabezado de petición HTTP personalizado para sustituir el método HTTP con el fin de sortear esa limitación.

POST

40. Los clientes pueden utilizar `POST` para crear un recurso.

– Por ejemplo, la siguiente petición HTTP envía una solicitud de patente.

Ejemplo con cargas útiles en formato XML según la Norma ST.96

El cliente envía la petición de solicitud de patente en formato XML:

```
POST /v1/patents/applications HTTP/1.1
Host: wipo.int
Accept: application/xml
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:ApplicationBody xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:languageCode="pl" com:receivingOffice="ST" com:st96Version="V5_0"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
ApplicationBody_V5_0.xsd">
      ...
</pat:ApplicationBody>
```

Se devuelve la siguiente respuesta HTTP para indicar que la solicitud de patente se ha enviado correctamente:

```
HTTP/1.1 201 Created
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:ApplicationBody xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:languageCode="pl" com:receivingOffice="ST" com:st96Version="V5_0"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
ApplicationBody_V5_0.xsd" applicationBodyStatus=”pending”>
      ...
</pat:ApplicationBody>
```

Ejemplo con cargas útiles en formato JSON según la Norma ST.97

El cliente envía la petición de solicitud de patente en formato JSON:

```
POST /v1/patents/applications HTTP/1.1
Host: wipo.int
Accept: application/json
Content-Type: application/json
{
      " applicationBody ": {
            ...
      }
}
```

Se devuelve la siguiente respuesta HTTP para indicar que la solicitud de patente se ha enviado correctamente:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      " applicationBody ": {
            "applicationBodyStatus" : "pending",
            ...
      }
```

```
}
```

[RSG-39] Las peticiones `POST` NO DEBEN ser idempotentes de acuerdo con la Norma RFC 2616 del IETF.

[RSG-40] Si el recurso se creó correctamente, el encabezado HTTP `Location` DEBERÍA contener un URI (absoluto o relativo) que apunte al recurso creado.

[RSG-41] Si el recurso se creó correctamente, la respuesta DEBERÍA contener el código de estado `201 Created`.

[RSG-42] Si el recurso se creó correctamente, la carga útil de respuesta DEBERÍA contener por defecto el cuerpo del recurso creado, para que el cliente pueda utilizarlo sin hacer otra llamada HTTP.

PUT

41. Los clientes pueden utilizar `PUT` para reemplazar por completo un recurso existente. Se debe respetar la idempotencia del método `PUT`. Una petición `PUT` tiene una semántica de actualización (como se especifica en la Norma RFC 7231 del IETF), y una semántica de inserción.

[RSG-43] Las peticiones `PUT` DEBEN ser idempotentes.

[RSG-44] Si no se encuentra un recurso, `PUT` DEBE devolver el código de estado `404 Not Found`.

[RSG-45] Si un recurso se actualiza correctamente, `PUT` DEBE devolver el código de estado `200 OK` si se devuelve el recurso actualizado o `204 No Content` si no se devuelve.

PATCH

42. Los clientes pueden utilizar `PATCH` si requieren una actualización parcial. En ese caso `PATCH` tiene que ser idempotente.

– Por ejemplo, la siguiente petición actualiza solo un idioma de la patente dado el número de esta:

```
PATCH /api/v1/patents/publications/100000000000001 HTTP/1.1
Host: wipo.int
If-Match:456
Content-Type: application/merge-patch+json
{ "languageCode": "en" }
```

43. `PATCH` no deben ser idempotente según la Norma RFC 2616 del IETF. Para hacerlo idempotente, la API puede recurrir a la técnica del bloqueo optimista recomendada en la Norma RFC 5789 del IETF.

[RSG-46] Las peticiones `PATCH` NO DEBEN ser idempotentes.

[RSG-47] Si una API web implementa actualizaciones parciales, `PATCH` DEBERÍA ser idempotente. Para conseguirlo, la API PUEDE recurrir a la técnica del bloqueo optimista recomendada en la Norma RFC 5789 del IETF.

[RSG-48] Si un recurso no se encuentra, `PATCH` DEBE devolver el código de estado `404 Not Found`.

[RSJ-49] Si una API web implementa actualizaciones parciales mediante `PATCH`, DEBE utilizar el formato JSON Merge Patch para describir el conjunto de cambios parciales, como se especifica en la Norma RFC 7386 del IETF utilizando `application/merge-patch+json` como `Content-Type`.

DELETE

44. Los clientes pueden usar `DELETE` para eliminar un recurso. Una petición `DELETE` no debe ser idempotente según la Norma RFC 2616 del IETF.

[RSG-50] Las peticiones `DELETE` NO DEBEN ser idempotentes.

[RSG-51] Si no se encuentra un recurso, `DELETE` DEBE devolver el código de estado `404 Not Found`.

[RSG-52] Si un recurso se elimina correctamente, `DELETE` DEBE devolver el código de estado `200 OK` si se devuelve el recurso eliminado o `204 No Content` si no se devuelve.

TRACE

45. El método `TRACE` no requiere ninguna semántica de la API y se utiliza para probar y diagnosticar información conforme a la Norma RFC 2616 del IETF, por ejemplo, para probar una cadena de *proxies*. `TRACE` permite al cliente ver lo que se recibe en el otro extremo de la cadena de petición y utiliza esos datos. Una petición `TRACE` NO DEBE ser idempotente de acuerdo con la Norma RFC 2616 del IETF.

[RSG-53] El destinatario final es el servidor de origen o el primer *proxy* o puerta de enlace que recibe un valor del encabezado `Max-Forwards` de cero en la petición. Las peticiones `TRACE` NO DEBEN incluir un cuerpo.

[RSG-54] Las peticiones `TRACE` NO DEBEN ser idempotentes.

[RSG-55] El valor del encabezado HTTP `Via` DEBE servir para rastrear la cadena de petición.

[RSG-56] Para que el cliente pueda limitar la longitud de la cadena de petición DEBE utilizarse el encabezado HTTP `Max-Forwards`.

[RSG-57] Si la petición es válida, la respuesta DEBERÍA contener el mensaje completo de la petición en el cuerpo de la respuesta, con `message/http` como `Content-Type`.

[RSG-58] Las respuestas a `TRACE` NO DEBEN ser almacenadas en caché.

[RSG-59] DEBERÍA devolverse a `TRACE` el código de estado `200 OK`.

OPTIONS

46. Los clientes pueden usar `OPTIONS` cuando necesiten obtener información sobre una API web. El método `OPTIONS` no requiere ninguna semántica de la API. Una petición `OPTIONS` DEBE ser idempotente de acuerdo con la Norma RFC 2616 del IETF, relativa a los encabezados HTTP personalizados.

[RSG-60] Las peticiones `OPTIONS` DEBEN ser idempotentes.

47. Es una práctica común de las API web utilizar encabezados HTTP personalizados con "`X-`" como prefijo común, lo que no se aprueba ni se recomienda en el documento RFC 6648.

[RSG-61] NO DEBERÍAN utilizarse encabezados HTTP personalizados que empiecen con el prefijo "`X-`".

[RSG-62] NO DEBERÍAN utilizarse encabezados HTTP personalizados para modificar un método HTTP a menos que sea para resolver una limitación técnica (véase, por ejemplo, [RSG-39]).

[RSG-63] Los encabezados HTTP personalizados tendrán el formato `<organization>-<header name>`, y los nombres de `<organization>` y `<header>` DEBERÍAN seguir la convención *kebab case*.

48. Según los principios de diseño orientado a servicios, los clientes y los servicios deberían evolucionar de forma independiente. El versionado de servicios lo hace posible. Las implementaciones comunes del versionado de servicios son: el versionado por encabezado (usando un encabezado personalizado), el versionado por cadena de consulta (por ejemplo, `v=v1`), el versionado por tipo de medios (por ejemplo, `Accept: application/vnd.v1+json`) y el versionado por URI (por ejemplo, `/api/v1/inventors`).

[RSG-64] Una API web DEBERÍA admitir un único método de versionado de servicios, como el versionado por URI (por ejemplo, `/api/v1/inventors`), el versionado por encabezado (por ejemplo, `Accept-version: v1`), o el versionado por tipo de medios (por ejemplo, `Accept: application/vnd.v1+json`). NO DEBERÍA utilizarse el versionado por cadena de consulta.

49. De acuerdo con los principios de diseño orientado a servicios, los proveedores y consumidores de servicios también deben evolucionar de forma independiente. El consumidor de servicios no debería verse afectado por cambios menores (retrocompatibles) por parte del proveedor de servicios. Por consiguiente, el versionado de servicios debería utilizar solo versiones mayores. En el caso de las API internas no publicadas (por ejemplo, para el desarrollo y las pruebas) también pueden utilizarse versiones menores, como las versiones semánticas.

[RSG-65] DEBERÍA seguirse un esquema de numeración de versiones que considere solo el número de versión mayor (por ejemplo, `/v1`).

50. Los identificadores de puntos finales de servicio incluyen información que puede variar con el tiempo. Puede que no sea posible cambiar todas las referencias al punto final desactualizado, lo que puede dar lugar a que el consumidor de servicios no pueda seguir interactuando con el punto final del servicio. El proveedor de servicios puede devolver una respuesta de redireccionamiento. El redireccionamiento puede ser temporal o permanente. Los códigos de estado HTTP posibles son los siguientes:

	Permanente	**Temporal**
Permite la modificación del método de petición de `POST` a `GET`	301	302
No permite la modificación del método de petición de `POST` a `GET`	308	307

Los códigos de estado `301` y `302`, al ser más genéricos, son los más aconsejables para aumentar la flexibilidad y evitar dificultades innecesarias.

[RSG-66] Los contratos de servicio API PUEDEN incluir la función de redireccionamiento del punto final. Cuando un consumidor de servicios intenta invocar un servicio, es posible que se devuelva una respuesta de redireccionamiento para indicarle que vuelva a enviar la petición a un nuevo punto final. Los redireccionamientos PUEDEN ser temporales o permanentes:

- redireccionamiento temporal - mediante el encabezado de respuesta HTTP `Location` y el código de estado HTTP `302 Found`, conforme a la Norma RFC 7231 del IETF; o
- redireccionamiento permanente - mediante el encabezado de respuesta HTTP `Location` y el código de estado HTTP `301 Moved Permanently`, de conformidad con el documento RFC 7238 del IETF.

51. Una API, durante su ciclo de vida, pasará por una serie de fases principales: planificación y diseño, desarrollo, prueba, implementación y retirada. Es preferible que la organización o los proveedores de servicios publiquen su estrategia de ciclo de vida de la API en lugar de formular recomendaciones sobre el período de tiempo que la API debería permanecer preferentemente en una fase determinada. En el Anexo VII se presenta un diagrama con los componentes básicos que definen una estrategia de ciclo de vida.

[RSG-67] Los desarrolladores de API DEBERÍAN publicar las estrategias de ciclo de vida de las API con el fin de que los usuarios sepan cuánto tiempo se mantendrá una versión.

Patrones de consulta de datos

Opciones de paginación

52. La paginación es un mecanismo que permite a los clientes recuperar datos en páginas. Mediante la paginación se evita sobrecargar al proveedor de servicios con peticiones de recursos conforme a los principios de diseño. El servidor debería establecer un tamaño de página por defecto en caso de que el consumidor de servicios no especifique uno. Las peticiones paginadas pueden no ser idempotentes, dado que una petición paginada no crea una instantánea de los datos.

[RSG-68] Las API web DEBERÍAN ser compatibles con la paginación.

[RSG-69] Las peticiones paginadas PUEDEN NO ser idempotentes.

[RSG-70] Las API web DEBEN utilizar parámetros de consulta para implementar la paginación.

[RSG-71] Las API web NO DEBEN utilizar encabezados HTTP para implementar la paginación.

[RSG-72] DEBERÍAN utilizarse los parámetros de consulta `limit=<number of items to deliver>` y `offset=<number of items to skip>` (`limit` es el número de elementos que se devolverán (tamaño de página), y `offset` es el número de elementos que se saltarán (desplazamiento). Si no se especifica un límite de tamaño de página, DEBERÍA definirse un valor por defecto —general o por colección—; el `offset` por defecto DEBE ser cero “0”.

– A continuación se muestra un ejemplo de un URL válido:

```
https://wipo.int/api/v1/patents?limit=10&offset=20
```

[RSG-73] Los valores de los parámetros `limit` y `offset` DEBERÍAN incluirse en la respuesta.

Ordenación

53. La recuperación de datos puede requerir ordenar los datos en orden ascendente o descendente. También puede utilizarse un criterio de ordenación multiclave. El orden se establece mediante el uso del parámetro de cadena de consulta `sort`. El valor de ese parámetro es una lista de claves de ordenación separadas por comas a las que pueden añadirse direcciones de ordenación mediante dos puntos ‘:’. Las direcciones de ordenación admitidas son `asc` para el orden ascendente o `desc` para el orden descendente. El cliente puede especificar una dirección de ordenación distinta para cada clave. Si no se especifica una dirección de ordenación para una clave, el servidor establece una dirección por defecto.

Por ejemplo:

a) Se especifican solo las claves de ordenación:

```
sort=key1,key2
```

`key1` es la primera clave y `key2` es la segunda, y el servidor establece las direcciones de ordenación por defecto.

b) Se especifican algunas direcciones de ordenación:

```
sort=key1:asc,key2
```

`key1` es la primera clave (orden ascendente) y `key2` es la segunda clave (el servidor establece por defecto la dirección, como para cualquier clave de ordenación sin una dirección especificada).

c) Se especifica la dirección de ordenación para cada clave:

```
sort=key1:asc,key2:desc
```

`key1` es la primera clave (orden ascendente) y `key2` es la segunda clave (orden descendiente).

54. Para establecer la ordenación por criterios multiatributos, el valor del parámetro de consulta puede ser una lista de claves de ordenación separadas por comas a las que se puede añadir, mediante dos puntos ':', la dirección de ordenamiento, con `asc` para el orden ascendente o `desc` para el orden descendente.

[RSG-74] Una API web DEBERÍA admitir la ordenación.

[RSG-75] Para establecer criterios multiatributos de ordenación, DEBE utilizarse un parámetro de consulta. El valor de dicho parámetro será una lista de claves de ordenación separadas por comas a las que se puede añadir, mediante dos puntos ':', la dirección de ordenamiento, con `asc` para el orden ascendente o `desc` para el orden descendente. En caso de que no se especifique una dirección de ordenación para una clave, el servidor DEBE establecer una por defecto.

[RSG-76] Las API web DEBERÍAN devolver los criterios de ordenación en la respuesta.

Expansión

55. Un consumidor de servicios puede controlar la cantidad de datos que recibe expandiendo un solo campo a un mayor número de objetos. Esto suele combinarse con un sistema hipermedia. En lugar de pedir que se incluya el ID de una entidad vinculada, el solicitante de servicios puede pedir que se expanda la representación completa de la entidad dentro de los resultados. Las llamadas de servicio pueden utilizar expansiones para obtener todos los datos que necesitan en una única petición API:

– Por ejemplo, si se admite el sistema hipermedia, la siguiente petición HTTP recupera una patente y expande la variable solicitante de la patente.

Ejemplo con cargas útiles en formato JSON según la Norma ST.97

Recupera una patente a partir de su número[6]:

```
GET /api/v1/patents/publications/100000000000001 HTTP/1.1
Host: wipo.int
Accept: application/json
```

La respuesta HTTP es la siguiente:

```
HTTP/1.1 200 OK
Content-Type: application/json
200 OK
{

      "patentPublication":{
            "languageCode": "en",
            ...
```

[6] `Patente/PatentNumber.xsd`

```
            "bibliographicData": {
                  "st96Version": "V5_0",
                  "applicationIdentification": {
                        "ipOfficeCode": "XX",
                        "applicationNumber": {
                              "applicationNumberText": "13797521"
                        },
                        "inventionSubjectMatterCategory": "Utility",
                        "filingDate": "2013-03-12"
                  },
                  patentGrantIdentification": {
                        "ipOfficeCode": "XX",
                        "patentNumber": "100000000000001"
                  },
                  ...
                  "partyBag": {
                        "applicantBag": {
                              "applicant": {
                                    "href":
"https://wipo.int/api/v1/link/to/applicants"
                              },
                        ...
                        }
                  }
            },
            ...
      }
}
```

Si en lugar de la petición anterior se utiliza la petición HTTP que figura a continuación, se recupera la información completa del solicitante de la patente con el número 100000000000001:

```
GET /api/v1/patents/publications?id=100000000000001&expand=applicant HTTP/1.1
Host: wipo.int
Accept: application/json
```

La respuesta HTTP es la siguiente:

```
HTTP/1.1 200 OK
Content-Type: application/json
200 OK
{
      "patentPublication":{
      "languageCode": "en",
      ...
      "bibliographicData": {
            "st96Version": "V5_0",
            "applicationIdentification": {
                  "ipOfficeCode": "XX",
                  "applicationNumber": {
                        "applicationNumberText": "13797521"
                  },
                  "inventionSubjectMatterCategory": "Utility",
                  "filingDate": "2013-03-12"
            },
            patentGrantIdentification": {
                  "ipOfficeCode": "XX",
                  "patentNumber": "100000000000001"
            },
            ...
            "partyBag": {
                  "applicantBag": {
                        "applicant": [
                              {
                                    "sequenceNumber": "001",
                                    "publicationContact": [
                                          {
                                                "name": {
                                                      "personName": …,
                                    "applicantCategory": "Applicant",
                              },
                              {
                                    "sequenceNumber": "002",
                                    "publicationContact": [
                                          {
                                                "name": {
                                                      "personName": …
                                                }
                                          }
                                    ],
                                    "applicantCategory": "Applicant",
                              },
                              {
                                    "sequenceNumber": "003",
                                    "publicationContact": [
                                          {
                                                "name": {
                                                      "personName": …
                                                }
                                          }
                                    ],
                                    "applicantCategory": "Applicant",
                              },
                  ...
                  }
            }
      },
            ...   }}
```

56. Una API web puede ser compatible con la expansión del cuerpo del contenido que se devuelve.

[RSG-77] Una API web PUEDE ser compatible con la expansión del cuerpo del contenido que se devuelve. DEBERÍA utilizarse el parámetro de consulta `expand=<comma-separated list of attributes names>`.

Proyección

57. Una API web debería admitir la proyección de campos, para controlar la cantidad de datos de una entidad que se devuelven en respuesta a una petición API. La proyección de campo puede reducir el tiempo de respuesta y el tamaño de la carga útil. Si solo se requieren atributos específicos de los datos recuperados, debe utilizarse un parámetro de consulta de proyección en lugar de las rutas URL. El parámetro de consulta debería ser: "`fields="<comma-separated list of attribute names>`. Un parámetro de consulta de proyección es más fácil de implementar y puede recuperar múltiples atributos. Si se admite la proyección, no debería aplicarse el esquema XSD/JSON en la respuesta, ya que esta no será válida frente al esquema original XSD/JSON.

– Por ejemplo, el siguiente mensaje de petición devuelve solo el nombre completo del inventor de la patente solicitada:

Con cargas útiles en formato XML según la Norma ST.96

Obtiene el nombre completo del inventor de la patente con identificador id12345:

```
GET /api/v1/patents/inventors/id12345?fields=fullName
Host: wipo.int
Accept: application/xml
```

A continuación figura un ejemplo de mensaje de respuesta HTTP:

```
HTTP/1.1 200 OK
Content-Type: application/xml
<?xml version="1.0" encoding="UTF-8"?>
<pat:Inventor xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:sequenceNumber="String" com:id="ID1"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
PatentPublication_V5_0.xsd">
      <Contact>
            <Name>
                  <PersonName>
                        <PersonFullName>John Smith</PersonFullName>
                  </PersonName>
            </Name>
      </Contact>
</pat:Inventor>
```

Con cargas útiles en formato JSON según la Norma ST.97

Obtiene el nombre completo del inventor de la patente con identificador[7] id12345:

```
GET /api/v1/patents/inventors/id12345?fields=fullName
Host: wipo.int
Accept: application/json
```

A continuación figura un ejemplo de mensaje de respuesta HTTP:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      "inventor": {
            "sequenceNumber": "001",
            "Contact": [
                  {
                        "name": {
                              "personName": [
                                    {
                                          "personFullName": "John Smith"
                                    }
                              ]
                        }
                  }
            ]
}}
```

[RSG-78] En caso de que una API web admita la proyección, DEBERÍA utilizarse, en lugar de las rutas URL, un parámetro de consulta con el formato "`fields=`"`<comma-separated list of attribute names>`.

Número de elementos

58. En algunos casos de uso, el consumidor de la API puede estar interesado en el número de elementos de una colección. Es muy frecuente que esto se combine con la paginación para saber el número total de elementos de la colección.

– Por ejemplo, la petición HTTP que figura a continuación recupera un máximo de 3 publicaciones de patentes, salta los 4 primeros resultados y debería incluir también en la respuesta el número total de los resultados disponibles:

Ejemplo con cargas útiles en formato XML conforme a la Norma ST.96

```
GET /api/v1/patents/publications?count=true&limit=3&offset=4 HTTP/1.1
Host: wipo.int
Accept: application/xml
```

Ejemplo de respuesta HTTP devuelta:

```
HTTP/1.1 200 OK
Content-Type: application/xml
```

[7] `Common/id.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<pat:PatentPublication xmlns="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xmlns:pat="http://www.wipo.int/standards/XMLSchema/ST96/Patent"
com:languageCode="de" com:st96Version="V5_0"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Patent
PatentPublication_V5_0.xsd">
   ...
</pat:PatentPublication>
<pat:PatentPublication>
   ...
</pat:PatentPublication>
   ...
<pat:PatentPublication>
   ...
</pat:PatentPublication>
<count>100</count>
```

Ejemplo con cargas útiles en formato JSON según la Norma ST.97

```
GET /api/v1/patents/publications?count=true&limit=3&offset=4 HTTP/1.1
Host: wipo.int
Accept: application/json
```

Ejemplo de respuesta HTTP devuelta:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
      "patentPublication": [
            {
            ...
            },
            {
            ...
            },
            {
            ...
            }
      ],
      "count": 100
}
```

59. Una posibilidad es que la API web pueda admitir la devolución del número de elementos de una colección mediante una función *inline*, es decir, como parte de la respuesta que contiene la propia colección. Otra opción es que forme parte de una envoltura de metadatos, fuera del cuerpo principal de la respuesta.

[RSG-79] Las API web DEBEN admitir la devolución del número de elementos de una colección.

[RSG-80] Para permitir la devolución del número de elementos de una colección DEBE utilizarse un parámetro de consulta.

[RSG-81] DEBERÍA utilizarse el parámetro de consulta `count` para devolver el número de elementos de una colección.

[RSG-82] Las API web PUEDEN admitir la devolución del número de elementos de una colección mediante una función *inline*, es decir, como parte de la respuesta que contiene la propia colección. DEBE utilizarse un parámetro de consulta.

[RSG-83] DEBERÍA utilizarse el parámetro de consulta `count=true`. Si no se especifica el valor de `count`, debería fijarse por defecto en `false`.

[RSG-84] Si una API web admite la paginación, DEBERÍA permitir la devolución del número de la colección (es decir, el número total de elementos de la colección) en la respuesta mediante una función *inline*.

Expresiones de búsqueda complejas

60. Los parámetros de consulta permiten recuperar datos con pocos criterios de búsqueda. Si existe un caso de uso en el que se tengan que buscar datos mediante expresiones de búsqueda complejas (con múltiples criterios, expresiones booleanas y operadores de búsqueda), la API debe diseñarse utilizando un lenguaje de consulta más complejo. El lenguaje de consulta tiene que estar basado en una gramática de búsqueda.

61. El lenguaje de consulta contextual (CQL) es un lenguaje formal para representar las consultas a los sistemas de recuperación de información, como los motores de búsqueda, los catálogos bibliográficos y la información de las colecciones de los museos. Basado en la semántica de Z39.50[8], su objetivo de diseño es conseguir que las consultas puedan ser leídas y escritas y que el lenguaje sea intuitivo y mantenga la expresión de los lenguajes de consulta más complejos. Se trata solo de una de las opciones recomendadas para su uso, ampliamente utilizada en el mercado.

[RSG-85] Si una API web es compatible con expresiones de búsqueda complejas, DEBERÍA especificarse un lenguaje de consulta, como el CQL.

[RSG-86] Un contrato de servicio DEBE especificar la gramática compatible (como los campos, las funciones, las palabras clave y los operadores).

[RSG-87] DEBE utilizarse el parámetro de consulta '`q`'.

Control de errores

62. Las respuestas de error deberían utilizar siempre el código de estado HTTP apropiado seleccionado de la lista de códigos de estado HTTP estándares (Norma RFC 7807), que figura en el Anexo V. Cuando el solicitante espera la respuesta en formato JSON, se devuelven los detalles del error en una estructura de datos común. A menos que el proyecto lo requiera, no es necesario definir códigos de error específicos de cada aplicación. La información de seguimiento de la pila y otros datos relacionados con la depuración no deberían aparecer en el cuerpo de la respuesta de error en los entornos de producción.

Carga útil de error

63. El control de errores se realiza en dos niveles: en el nivel de protocolo (HTTP) y en el nivel de aplicación (carga útil devuelta). En el nivel de protocolo, la API web devuelve el código de estado HTTP que corresponda y, en el nivel de aplicación, la API web devuelve una carga útil que informa del error con suficiente granularidad (atributos obligatorios y opcionales).

64. En relación con los atributos obligatorios y opcionales para el control de errores en el nivel de aplicación:

a) Los atributos `code` y `message` que figuran a continuación son obligatorios y, si bien el atributo `message` puede cambiar con el tiempo, el atributo `code` no cambiará, sino que es fijo y siempre se referirá al problema en cuestión:

– `code` (entero) - código técnico de la situación de error que se utilizará con fines de asistencia; y
– `message` (cadena) - mensaje dirigido al usuario (localizable) que describe la petición de error como se solicita en el encabezado HTTP `Accept-Language` (véase RSG-114).

[8] Véase el capítulo *Material de referencia*.

b) Los siguientes atributos son condicionalmente obligatorios:

– `details` - si el procesamiento de errores requiere la anidación de las respuestas de error, debe utilizarse el campo `details` para ese fin. El campo `details` debe contener una matriz de objetos JSON que muestre las propiedades de `code` y `message` con la misma semántica antes descrita.

c) Los siguientes atributos son opcionales:

– `target` - la estructura de error puede contener el atributo `target` que describe un elemento de datos (por ejemplo, una ruta de recurso);
– `status` - duplicado del código de estado HTTP para propagarlo a lo largo de la cadena de llamada o para escribirlo en el registro de soporte sin necesidad de añadir explícitamente el código de estado HTTP una y otra vez;
– `moreInfo` - matriz de enlaces que contienen más información sobre la situación de error, como indicaciones para el usuario final; y
– `internalMessage` – un mensaje técnico, por ejemplo, para fines de registro.

65. El control de errores debería ajustarse a las normas HTTP (RFC 2616). Se recomienda una carga útil de error mínima:

– Por ejemplo, la respuesta HTTP que figura a continuación se devuelve cuando no se ha encontrado la marca para el número de registro internacional proporcionado:

Ejemplo con carga útil en formato XML conforme a la Norma ST.96

```
GET /api/v1/trademarks?irn=000000000000001John%20Smith&expiryDate=2018-12-31.
HTTP/1.1
Host: wipo.int
Accept: application/xml
```

A continuación se muestra un ejemplo de respuesta HTTP:

```
HTTP/1.1 404
Content-Type: application/xml
 <?xml version="1.0" encoding="UTF-8"?>
<com:TransactionError xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:com="http://www.wipo.int/standards/XMLSchema/ST96/Common"
xsi:schemaLocation="http://www.wipo.int/standards/XMLSchema/ST96/Common
TransactionError.xsd">
      <com:TransactionErrorCode>TRADEMARK_NOT_FOUND</com:TransactionErrorCode>
      <com:TransactionErrorText>The trademark with the provided International
Registration Number was not found</com:TransactionErrorCode>
</com:TransactionError>
```

Ejemplo con carga útil en formato JSON según la Norma ST.97

```
HTTP/1.1 404
Content-Type: application/json
{
      "transactionError": [
            {
                  "transactionErrorCode": "TRADEMARK_NOT_FOUND"
            },
            {
                  "transactionErrorText": "The trademark with the provided
International Registration Number was not found"
            },
      ]
}
```

[RSG-88] En el nivel de protocolo, una API web DEBE devolver un código de estado HTTP apropiado seleccionado de la lista de códigos de estado HTTP estándares.

[RSJ-89] En el nivel de aplicación, una API web DEBE devolver una carga útil que informe del error con suficiente granularidad. Los atributos `code` y `message` son obligatorios, el atributo `details` es condicionalmente obligatorio y los atributos `target`, `status`, `moreInfo`, e `internalMessage` son opcionales.

[RSG-90] Los errores NO DEBEN exponer datos esenciales para la seguridad o detalles técnicos internos, como las pilas de llamada en los mensajes de error.

[RSG-91] El encabezado HTTP `Reason-Phrase` (descrito en la Norma RFC 2616) NO DEBE utilizarse para transmitir mensajes de error.

ID de correlación

66. Normalmente, el consumo de un servicio provoca la activación de otros muchos servicios. Debería existir un mecanismo para correlacionar todas las activaciones de servicios en el mismo contexto de ejecución, por ejemplo, mediante la inclusión del ID de correlación en los mensajes de registro, lo que identifica unívocamente el error registrado. Conviene asignar con ese fin un nombre de encabezado —por ejemplo, normalmente se utilizan Request-ID o Correlation-ID—, en la fase de diseño de una API, lo que posteriormente favorecerá la compatibilidad entre diferentes API y las últimas implementaciones.

[RSG-92] Cada error registrado DEBERÍA tener un ID de correlación único. DEBERÍA utilizarse un encabezado HTTP personalizado con el nombre Correlation-ID.

Contrato de servicio

67. REST no es un protocolo o una arquitectura, sino un estilo de arquitectura con propiedades y restricciones de arquitectura. No hay normas oficiales para los contratos de API REST. En la presente norma la documentación de la API es equivalente al contrato de servicio REST. El contrato de servicio se basa en los tres elementos fundamentales que se indican a continuación:

a) Sintaxis del identificador de recursos: ¿cómo se puede expresar el lugar desde o hacia el que se transfieren los datos?
b) Métodos: ¿cuáles son los mecanismos de protocolo utilizados para transferir los datos?
c) Tipos de medios: ¿qué tipo de datos se están transfiriendo? Los servicios REST individuales utilizan estos elementos en diferentes combinaciones para exponer sus capacidades. La definición de un conjunto de

referencia de estos elementos para su uso en una colección (o inventario) de servicios hace que este tipo de contrato de servicio sea uniforme.

[RSG-93] El modelo de contrato de servicio DEBE especificar lo siguiente:

- la versión de la API;
- información sobre la semántica de los elementos de la API;
- los recursos;
- los atributos de los recursos;
- los parámetros de consulta;
- los métodos;
- los tipos de medios;
- la gramática de búsqueda (si se admite alguna);
- los códigos de estado HTTP;
- los métodos HTTP;
- las restricciones y las características distintivas; y
- la seguridad (por ejemplo, los esquemas privados).

[RSG-94] El modelo de contrato de servicio DEBERÍA incluir peticiones y respuestas en el esquema XML o en el esquema JSON y ejemplos de uso de la API en los formatos compatibles, a saber, XML o JSON.

[RSG-95] Una API REST DEBE proporcionar la documentación de la API como un contrato de servicio.

[RSG-96] Toda implementación de una API web que no cumpla con la presente norma DEBE ser documentada explícitamente en el contrato de servicio. Si no se especifica ninguna desviación de una de sus normas en el contrato de servicio, DEBE asumirse que se sigue la presente norma.

[RSG-97] Un contrato de servicio DEBE permitir la generación de código esquemático de cliente API.

[RSG-98] Un contrato de servicio DEBERÍA permitir la generación de código esquemático de servidor.

68. La documentación de la API web puede escribirse, por ejemplo, utilizando el lenguaje de modelado de API RESTful (RAML), la especificación OpenAPI (OAS) y el lenguaje de descripción de servicios web (WSDL). En la presente norma se recomienda utilizar RAML por ser el único lenguaje totalmente compatible con la validación de petición/respuesta que usa tanto esquema XSD como esquema JSON[9].

[RSG-99] La documentación de las API web DEBERÍA escribirse utilizando el RAML o la OAS. NO DEBERÍAN utilizarse formatos de documentación personalizados.

Tiempo de espera

69. De acuerdo con los principios de diseño orientado a servicios, debería limitarse el uso del servidor.

[RSG-100] Un consumidor de API web DEBERÍA poder especificar un tiempo de espera del servidor para cada petición; DEBERÍA utilizarse un encabezado HTTP personalizado. También DEBERÍA establecerse un tiempo máximo de espera del servidor para evitar un uso excesivo de los recursos del servidor.

[9] OAS es una especificación. También es compatible con Markdown, pero RAML no. Por otro lado, aunque tanto la OAS como el RAML son compatibles con el esquema JSON de validación de petición/respuesta, la OAS no es compatible con el esquema XSD. Por lo tanto, la OAS podrá ser recomendada cuando se completen sus funciones.

Control del estado

70. Si el proceso se desarrolla siguiendo los principios de REST, será el cliente, y no el servidor, el encargado de controlar el estado, ya que las API REST no tienen estado. Por ejemplo, si varios servidores implementan una sesión, debe desaconsejarse la duplicación.

Versionado por respuesta

71. La recuperación reiterada del mismo conjunto de datos puede suponer un consumo de ancho de banda sin que el conjunto de datos se haya modificado entre las peticiones. Los datos deberían ser recuperados solo si se han modificado. Para ello puede recurrirse a la validación de recursos basada en el contenido o a la validación de recursos basada en el tiempo. Si se utiliza el versionado por respuesta, el consumidor de servicios puede implementar el bloqueo optimista.

[RSG-101] Una API web DEBERÍA admitir la recuperación condicional de datos, para asegurar que solo se recuperen los datos que se hayan modificado. DEBERÍA utilizarse la validación de recursos basada en el contenido por ser la más precisa.

[RSG-102] Para implementar la validación de recursos basada en el contenido, DEBERÍA utilizarse el encabezado HTTP `ETag` en la respuesta para codificar el estado de los datos. El valor de `Etag` DEBERÍA utilizarse en los encabezados condicionales HTTP (como `If-Match` o `If-None-Match`) en las peticiones subsiguientes. Si no se modificaron los datos desde que la petición devolvió el `ETag`, el servidor DEBERÍA devolver el código de estado `304 Not Modified`. Este mecanismo se especifica en las normas RFC 7231 y RFC 7232 del IETF.

[RSG-103] Para implementar la validación de recursos basada en el tiempo DEBERÍA utilizarse el encabezado HTTP `Last-Modified`. Este mecanismo se especifica en las normas RFC 7231 y RFC 7232 del IETF.

[RSG-104] Utilizando el versionado por respuesta, un consumidor de servicios PUEDE implementar el bloqueo optimista.

Almacenamiento en caché

72. La implementación de una API web debería ser compatible con el control de caché para ahorrar ancho de banda, de acuerdo con la Norma RFC 7234 del IETF.

[RSG-105] Las API web DEBEN admitir el almacenamiento en caché de los resultados de `GET`; y PUEDEN admitir el almacenamiento en caché de los resultados de otros métodos HTTP.

[RSG-106] DEBERÍAN utilizarse los encabezados de respuesta HTTP `Cache-Control` y `Expires`. `Expires` PUEDE utilizarse para prestar asistencia a los clientes heredados.

Transferencia gestionada de archivos

73. Cuando se transfieren (es decir, se descargan o se suben) archivos grandes, es muy probable que se produzcan interrupciones en la red o algún otro tipo de fallo en la transmisión. Además, tanto el proveedor de servicios como el consumidor de servicios consumen una gran cantidad de memoria. Por consiguiente, se recomienda transferir los archivos grandes por partes mediante varias peticiones. Esta opción permite también conocer el progreso de la descarga o la subida total. La transferencia de archivos grandes por partes debe ser compatible con la reanudación y el proveedor de servicios debe indicar si permite la transferencia de archivos grandes por partes[10].

74. Hay dos enfoques para implementar este tipo de transferencia: uno consiste en usar el encabezado `Transfer-Encoding: chunked` y el otro en usar el encabezado `Content-Length`. Estos encabezados no deben usarse juntos. `Content-Length` indica el tamaño total del archivo transferido, por lo que el receptor conocerá la longitud del cuerpo y

[10] El proveedor de servicios puede devolver la ubicación del archivo y entonces el consumidor de servicios podrá llamar a un servicio de directorio para descargar el archivo. Finalmente, se requiere la descarga de una parte del archivo. Este párrafo no es aplicable a los protocolos que no son REST, como FTP o sFTP o rsync.

podrá estimar el tiempo necesario para completar la descarga. El encabezado `Transfer-Encoding: chunked` es útil para transmitir datos no acotados, como material sonoro o visual, pero no archivos. Se recomienda utilizar el encabezado `Content-Length` para la descarga ya que requiere un nivel bajo de utilización del servidor en comparación con `Transfer-Encoding: chunked, y` el encabezado `Transfer-Encoding: chunked` para la subida.

Una API web debe indicar si es compatible con descargas de archivos por partes, mediante respuestas a peticiones `HEAD` que incluyan los encabezados de respuesta HTTP `Accept-Ranges` y `Content-Length`. El primer encabezado debe indicar la unidad que puede ser usada para definir un rango, y nunca debe tener el valor 'ninguno'. El segundo indica el tamaño total del archivo que se descargará.

[RSG-107] Una API web DEBE indicar si permite descargas de archivos por partes, mediante respuestas a peticiones `HEAD` que incluyan los encabezados de respuesta HTTP `Accept-Ranges` y `Content-Length`.

75. Una API web que permita la descarga de archivos grandes debería admitir peticiones parciales de acuerdo con la Norma RFC 7232 del IETF, a saber:

- el consumidor de servicios debe utilizar el encabezado HTTP `Range` para realizar una petición de rango;
- La respuesta del proveedor de servicios debe contener los encabezados HTTP `Content-Range` y `Content-Length`; y
- La respuesta del proveedor de servicios debe incluir el código de estado HTTP `206 Partial Content` en caso de que la petición de rango sea correcta. Si el rango solicitado está fuera de los límites (los valores de rango no se superponen a la extensión del recurso), el servidor responde con el código de estado `416 Requested Range Not Satisfiable`. Si el rango solicitado es válido, se devuelve el código de estado `200 OK` desde un servidor.

[RSG-108] Una API web DEBERÍA permitir la descarga de archivos por partes. DEBERÍA admitirse la petición de múltiples rangos.

76. También pueden solicitarse múltiples rangos si se utiliza el encabezado HTTP `Content-Type: multipart/byteranges; boundary=XXXXX`. Una petición de rango puede ser condicional si se combina con los encabezados HTTP `ETag` o `If-Range`.

77. No existe ningún documento RFC del IETF relativo a la subida de archivos grandes. Por consiguiente, en la presente norma no se incluye ninguna recomendación al respecto.

[RSG-109] Una API web DEBERÍA especificar si admite la subida de archivos por partes.

[RSG-110] Una API web DEBERÍA ser compatible con la subida de archivos por partes. DEBERÍA admitirse la petición de múltiples rangos.

78. La Norma RFC 2616 del IETF no impone ningún límite de tamaño específico para las peticiones. El contrato de servicio de la API debe especificar el límite máximo para las peticiones. Además, durante la ejecución, el proveedor de servicios debe indicar al consumidor de servicios si se ha superado el límite máximo permitido.

[RSG-111] El proveedor de servicios DEBERÍA devolver con los encabezados de respuesta HTTP el código de estado HTTP `413 Request Entity Too Large` en caso de que la petición supere el límite máximo permitido. PUEDE utilizarse un encabezado HTTP personalizado para indicar el tamaño máximo de la petición.

Gestión de preferencias

79. Un proveedor de servicios puede permitir que un consumidor de servicios configure valores e influya en la forma en que el proveedor procesa sus peticiones. En la Norma RFC 7240 del IETF se describe un método estándar para llevar a cabo la gestión de preferencias.

[RSG-112] Si una API web admite la gestión de preferencias, esta DEBERÍA implementarse de acuerdo con la Norma RFC 7240 del IETF, es decir, DEBERÍA utilizarse el encabezado de petición HTTP `Prefer` y

devolverse el encabezado de respuesta HTTP `Preference-Applied` (que incluye la petición original).

[RSG-113] Si una API web es compatible con la gestión de preferencias, en el contrato de servicio DEBE indicarse la nomenclatura de las preferencias que PUEDEN establecerse mediante el encabezado `Prefer`.

Traducción

80. Un consumidor de servicios puede solicitar las respuestas en un idioma específico si el proveedor del servicio lo permite. En la Norma RFC 7231 del IETF se establecen especificaciones estándares para la gestión de una serie de lenguas naturales.

[RSG-114] Si una API web permite la traducción de datos, DEBE admitir el encabezado de petición HTTP `Accept-Language` con el que se indica el conjunto de lenguas naturales preferidas para la respuesta, conforme a lo especificado en la Norma RFC 7231 del IETF.

Operaciones de larga duración

81. Hay casos en los que las API web pueden conllevar operaciones de larga duración. Por ejemplo, la generación de un PDF por el proveedor de servicios puede llevar algunos minutos. En este párrafo se recomienda un patrón típico de intercambio de mensajes para implementar esos casos, como el que figura a continuación:

```
// (a)
GET https://wipo.int/api/v1/patents
Accept: application/pdf
…
// (b)
HTTP/1.1 202 Accepted
Location: https://wipo.int/api/v1/queues/12345
…
// (c1)
GET https://wipo.int/api/v1/queues/12345
…
HTTP/1.1 200 OK
…
// (c2)
GET https://wipo.int/api/v1/queues/12345
HTTP/1.1 303 See Other
Location: https://wipo.int/api/v1/path/to/pdf
…
// (c3)
GET https://wipo.int/api/v1/path/to/pdf
…
```

82. Si una API es compatible con operaciones de larga duración, estas deberían ejecutarse de forma asincrónica para garantizar que el usuario no tenga que esperar por la respuesta. La norma enunciada a continuación recomienda un enfoque para la implementación de dichas operaciones.

[RSG-115] Si la API permite operaciones de larga duración, estas DEBERÍAN ejecutarse de forma asíncrona. DEBERÍA seguirse el enfoque descrito a continuación:

a) el consumidor de servicios activa la operación de servicio;
b) la operación de servicio devuelve el código de estado `202 Accepted` conforme a la Norma RFC 7231 del IETF (sección 6.3.3), que indica que la petición fue aceptada para su procesamiento, pero este no se completó. La ubicación de la tarea pendiente que se creó también se devuelve con el encabezado HTTP `Location`; y

c) el consumidor de servicios llama a la ubicación devuelta para saber si el recurso está disponible. Si el recurso no está disponible, la respuesta DEBERÍA incluir el código de estado `200 OK` y el estado de la tarea (por ejemplo, pendiente) y PUEDE contener otra información (por ejemplo, un indicador de progreso y/o un enlace para cancelar o eliminar la tarea mediante el método HTTP `DELETE`). Si el recurso está disponible, la respuesta DEBERÍA incluir el código de estado `303 See Other`, y el encabezado HTTP `Location` DEBERÍA contener el URL para recuperar los resultados de la tarea.

Modelo de seguridad

Normas generales

83. En el marco de aplicación de la presente norma, se entiende por seguridad de la API los atributos de seguridad que son fundamentales para garantizar que la información accesible por la API y la propia API sean seguras durante todo su ciclo de vida. Dichos atributos son la confidencialidad, la integridad, la disponibilidad, la confianza, el no repudio, la compartimentación, la autenticación, la autorización y la auditoría.

[RSG-116] Confidencialidad: las API y su información DEBEN ser identificadas, clasificadas y protegidas en todo momento frente a acciones no autorizadas de acceso, difusión e interceptación. DEBEN seguirse los principios de mínimos privilegios, cero confianza, necesidad de conocer y necesidad de compartir[11].

[RSG-117] Garantía de integridad: las API y su información DEBEN estar protegidas contra acciones no autorizadas de modificación, duplicación, corrupción y destrucción. La modificación de la información DEBE realizarse mediante transacciones e interfaces aprobadas. La actualización de los sistemas DEBE llevarse a cabo mediante procesos aprobados de gestión de la configuración, gestión de cambios y gestión de parches.

[RSG-118] Disponibilidad: las API y su información DEBEN estar disponibles para los usuarios autorizados en el debido momento, conforme se especifica en los acuerdos de nivel de servicio, las políticas de control de acceso y los procesos operativos definidos.

[RSG-119] No repudio: toda transacción procesada o acción realizada por las API DEBE garantizar el no repudio mediante la auditoría, autorización y autenticación adecuadas, rutas seguras y servicios y mecanismos de no repudio.

[RSG-120] Autenticación, autorización, auditoría: los usuarios, los sistemas, las API o los dispositivos que participen en transacciones o acciones importantes DEBEN ser autenticados, autorizados mediante servicios de control de acceso basados en roles o atributos y mantener la separación de tareas. Además, todas las acciones DEBEN ser registradas y el nivel de autenticación debe aumentar en función del riesgo asociado a la información.

Directrices para una gestión de las API segura y resistente a las amenazas

84. Las API deberían diseñarse, construirse, probarse e implementarse teniendo en cuenta los requerimientos de seguridad y los riesgos. Las contramedidas y los controles necesarios deberían ser incorporados directamente en el diseño y no a posteriori. Se recomienda utilizar las mejores prácticas y normas, como el Proyecto Abierto de Seguridad de Aplicaciones Web (OWASP).

[RSG-121] En el desarrollo de las API DEBEN tenerse debidamente en cuenta las amenazas, los casos de uso malicioso, las técnicas de codificación segura, la seguridad de la capa de transporte (TLS) y las pruebas de seguridad, sobre todo:

- los `PUT` y `POST`: qué modificaciones de los datos internos podrían utilizarse para atacar o transmitir información errónea.
- los `DELETE`: podrían utilizarse para eliminar el contenido de un almacén de recursos internos;
- los métodos permitidos de la lista blanca: para asegurar que los métodos HTTP permitidos sean debidamente restringidos y que otros devuelvan un código de respuesta adecuado; y

[11] https://www.owasp.org/index.php/Security_by_Design_Principles

- los ataques más conocidos deberían considerarse durante la fase de modelado de amenazas del proceso de diseño para asegurar que el riesgo de amenaza no aumente. DEBEN tenerse en cuenta las amenazas y medidas de mitigación definidas en la lista de los diez riesgos más críticos de OWASP[12].

[RSG-122] En el desarrollo de las API DEBERÍAN seguirse las normas y las mejores prácticas que se enumeran a continuación:

- las mejores prácticas de codificación segura: *OWASP Secure Coding Principles* (Principios de Codificación Segura de OWASP);
- la seguridad de las API Rest: *REST Security Cheat Sheet* (Lista de Recomendaciones Prácticas sobre Seguridad REST);
- las entradas de escape y la protección de secuencias de comandos en sitios cruzados: *OWASP XSS Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Secuencias de Comandos en Sitios Cruzados);
- la prevención de ataques de inyección SQL: *OWASP SQL Injection Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Inyección SQL), *OWASP Parameterization Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Parametrización); y
- la seguridad de la capa de transporte: *OWASP Transport Layer Protection Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre la Seguridad de la Capa de Transporte).

[RSG-123] DEBEN realizarse pruebas de seguridad y evaluaciones de vulnerabilidad para garantizar que las API son seguras y resistentes a las amenazas. Para ello PUEDEN aprovecharse las pruebas de seguridad de aplicaciones estáticas y dinámicas (SAST/DAST), las herramientas de gestión automatizada de vulnerabilidades y las pruebas de penetración.

Cifrado, integridad y no repudio

85. Los servicios protegidos deben estar asegurados para proteger las credenciales de autenticación en tránsito: por ejemplo, las contraseñas, las claves de API o los *tokens* web JSON. También debería garantizarse la integridad de los datos transmitidos y el no repudio de las medidas adoptadas. Los mecanismos criptográficos seguros pueden garantizar la confidencialidad, el cifrado, la integridad y el no repudio. La confidencialidad directa perfecta es un medio para asegurar que las claves de sesión no se vean comprometidas.

[RSG-124] Los servicios protegidos DEBEN proporcionar puntos finales HTTP solo mediante TLS 1.2, o una versión superior, con un paquete de cifrado que incluya el protocolo Elliptic-curve Diffie–Hellman (ECDH) para el intercambio de claves.

[RSG-125] Cuando se consideran los protocolos de autenticación, DEBERÍA utilizarse una confidencialidad directa perfecta para garantizar la seguridad del transporte. NO DEBERÍAN permitirse los algoritmos criptográficos inseguros ni la compatibilidad hacia atrás con SSL 3 y TLS 1.0/1.1.

[RSG-126] Para una máxima seguridad y confianza, DEBERÍA establecerse una VPN Ipsec de sitio a sitio para proteger mejor la información transmitida a través de redes inseguras.

[RSG-127] La aplicación consumidora DEBERÍA validar la cadena de certificados TLS al realizar peticiones a recursos protegidos, incluida la comprobación de la lista de revocación de certificados.

[RSG-128] Los servicios protegidos DEBERÍAN utilizar solo certificados válidos emitidos por una autoridad certificadora de confianza.

[RSG-129] Los *tokens* DEBERÍAN ser firmados utilizando algoritmos de firma segura que cumplan con el estándar de firma digital (DSS) FIPS 186-4. DEBERÍAN considerarse los algoritmos de firma digital de Rivest, Shamir y Adleman (RSA) o de curva elíptica (ECDSA).

[12] https://owasp.org/www-project-top-ten/2017

Autenticación y autorización

86. La autorización permite controlar el acceso a un recurso. No solo abarca la implementación de controles de acceso, sino también la definición de esos controles, incluidas las normas y políticas de acceso, que deberían establecer el nivel de acceso solicitado aceptable para el proveedor y la aplicación consumidora. El fundamento del control de acceso consiste en que el proveedor concede o deniega a una aplicación consumidora y/o un consumidor el acceso a un recurso con un determinado nivel de granularidad. El acceso de grano grueso debería considerarse en el punto de petición de la API o de la puerta de enlace de la API, mientras que el control de acceso de grano fino debería considerarse en los servicios en segundo plano, si es posible. Puede utilizarse el modelo de control de acceso basado en roles (RBAC) o el modelo de control de acceso basado en atributos (ABAC).

87. Si un servicio está protegido, debería optarse por OpenID Connect en lugar de OAuth 2.0 ya que subsana muchas de las deficiencias de este último y proporciona una forma estandarizada de obtener los datos del perfil del propietario de un recurso, el formato estandarizado de JSON Web Token (JWT) y la criptografía. No deberían utilizarse otros esquemas de seguridad como la autorización básica de HTTP, que requiere que el cliente guarde una contraseña en texto plano para enviarla junto con cada petición. Además, la verificación de esa contraseña sería más lenta porque se tendría que acceder al almacén de credenciales. OAuth 2.0 no especifica el *token* de seguridad. Por consiguiente, debería utilizarse el JWT frente a, por ejemplo, el SAML 2.0, que requiere más información.

[RSG-130] La autenticación anónima DEBE utilizarse solo cuando los clientes y la aplicación que están utilizando acceden a información o funciones con un nivel de sensibilidad bajo que no deberían requerir autenticación, como la información pública.

[RSG-131] NO DEBE permitirse la autenticación con nombre de usuario y contraseña o con contraseña *hash* [generada mediante una función *hash* o función resumen].

[RSG-132] Si un servicio está protegido, DEBERÍA usarse el protocolo OpenID Connect.

[RSG-133] Cuando se utiliza JSON Web Token (JWT), el secreto JWT DEBERÍA tener una entropía alta para aumentar el factor de trabajo de los ataques de fuerza bruta; los *tokens* TTL y RTTL DEBERÍAN ser lo más cortos posibles; y la información sensible NO DEBERÍA almacenarse en la carga útil JWT.

88. Una opción común de diseño de seguridad es centralizar la autenticación de los usuarios, que debería ser almacenada en un proveedor de identidad (IdP) o localmente en los puntos finales REST.

89. Los servicios deberían procurar evitar la fuga de credenciales. Las contraseñas, los *tokens* de seguridad y las claves de API no deben aparecer en el URL, ya que podrían quedarse en los registros del servidor web que podrían ser utilizados. Por ejemplo, el siguiente URL no es apropiado por contener la clave de API: `https://wipo.int/api/patents?apiKey=a53f435643de32`.

[RSG-134] En las peticiones `POST y PUT`, los datos sensibles DEBERÍAN ser transferidos en el cuerpo de la petición o mediante los encabezados de la petición.

[RSG-135] En las peticiones `GET`, los datos sensibles DEBERÍAN ser transferidos en un encabezado HTTP.

[RSG-136] Para minimizar la latencia y reducir el acoplamiento entre los servicios protegidos, la decisión de control de acceso DEBERÍA ser tomada localmente por los puntos finales REST.

90. Autenticación basada en las claves de API. Siempre que se requiera una autenticación de sistema a sistema, deberían utilizarse claves de API generadas de forma automática y aleatoria. El riesgo inherente a este modo de autenticación es que quienes tengan una copia de la clave de API pueden utilizarla como si fueran la aplicación consumidora legítima. Por consiguiente, todas las comunicaciones deberían cumplir lo dispuesto en [RSG-124] para proteger la clave en tránsito. La responsabilidad de proteger adecuadamente la copia de la clave de API recae en el desarrollador de la aplicación. Si la clave de API está incluida en la aplicación consumidora, podría ser descifrada y extraída. Si se almacena en archivos de texto plano, podría ser robada y reutilizada con fines maliciosos. Por consiguiente, las claves de API deben estar protegidas por un almacén de credenciales o un mecanismo de gestión secreto. Las claves de API pueden utilizarse para controlar el uso de los servicios, incluidos los servicios públicos.

[RSG-137] DEBERÍAN utilizarse claves de API en los servicios protegidos y públicos para evitar que se sobrecargue a sus proveedores de servicios con múltiples peticiones (ataques de denegación de servicio). En el caso de los servicios protegidos, PUEDEN utilizarse claves de API para la monetización (planes adquiridos), la aplicación de políticas de calidad de servicio y la supervisión.

[RSG-138] Las claves de API PUEDEN combinarse con el encabezado de petición HTTP `user-agent` para determinar si el usuario es una persona o un agente de software, conforme a lo especificado en la Norma RFC 7231 del IETF.

[RSG-139] El proveedor de servicios DEBERÍA devolver junto con los encabezados de respuesta HTTP el estado de uso en el momento. Se PUEDEN devolver los siguientes datos de respuesta:

- límite de tasa (*rate limit*): máximo número de peticiones por minuto permitido en el sistema;
- límite de tasa restante (*rate limit remaining*): número de peticiones restantes (-1 indica que se ha superado el límite); y
- restablecimiento del límite de tasa (*rate limit reset*): tiempo (en segundos) necesario para que se restablezca el número máximo de peticiones permitidas.

[RSG-140] El proveedor de servicios DEBERÍA devolver el código de estado `429 Too Many Requests` si se reciben demasiadas peticiones en poco tiempo.

[RSG-141] Las claves de API DEBEN ser revocadas si el cliente incumple el contrato de uso, conforme a lo especificado por la oficina de propiedad intelectual.

[RSG-142] La transferencia de las claves de API DEBERÍA realizarse mediante encabezados HTTP personalizados. NO DEBERÍA realizarse mediante parámetros de consulta.

[RSG-143] Las claves de API DEBERÍAN generarse de forma aleatoria.

91. Si bien la criptografía y los certificados de clave pública se utilizan con bastante frecuencia, cuando una API web requiera proporcionar mayor seguridad con una autenticación más fuerte que la que ofrecen las claves de API debería utilizarse la autenticación mutua basada en certificados. Los certificados seguros y de confianza deben ser emitidos por una autoridad certificadora de confianza mutua mediante un proceso de establecimiento de confianza o una certificación cruzada. Se puede recurrir a una autenticación fuerte para mitigar los riesgos de seguridad de la identidad propios de los sistemas sensibles y las acciones privilegiadas. Deberían utilizarse certificados compartidos entre el cliente y el servidor, por ejemplo X.509.

[RSG-144] Los certificados seguros y de confianza DEBEN ser emitidos por una autoridad certificadora de confianza mutua mediante un proceso de establecimiento de confianza o una certificación cruzada.

[RSG-145] DEBERÍAN utilizarse certificados compartidos entre el cliente y el servidor, por ejemplo X.509, para mitigar los riesgos de seguridad de la identidad propios de los sistemas sensibles y las acciones privilegiadas.

[RSG-146] Para servicios con un alto nivel de privilegio, la autenticación mutua bidireccional entre el cliente y el servidor DEBERÍA utilizar certificados para ofrecer una mayor protección.

[RSG-147] DEBERÍA implementarse la autenticación multifactor para mitigar los riesgos de la identidad en aplicaciones con un perfil de alto riesgo, en sistemas que procesen información muy sensible o en acciones privilegiadas.

Disponibilidad y protección frente a amenazas

92. La disponibilidad en este contexto se refiere a la protección contra las amenazas para reducir al mínimo el tiempo de inactividad de la API, y al análisis de la forma en que pueden mitigarse las amenazas a las API mediante principios básicos de diseño. También abarca, entre otras acciones, el escalado para satisfacer la demanda y asegurar que los entornos de alojamiento sean estables. Todo ello se tiene en cuenta en todas las pilas de *hardware* y *software* que permiten el desarrollo de las API. Normalmente estas cuestiones se abordan en el marco de las normas de continuidad de las actividades y

recuperación en casos de desastre, que recomiendan un enfoque de evaluación de riesgos para definir los requisitos de disponibilidad.

Peticiones entre dominios

93. Algunas peticiones entre dominios, en particular las peticiones Ajax, están prohibidas por defecto en virtud de la política de seguridad de mismo origen. Con arreglo a dicha política, un navegador de Internet permite que los programas de instrucciones (*scripts*) contenidos en una primera página web accedan a los datos de una segunda página web, solo si ambas páginas web tienen el mismo origen (indicado mediante una combinación del esquema de URI, el nombre del anfitrión y el número de puerto).

94. El uso compartido de recursos de origen cruzado (CORS) es una norma del W3C que especifica de manera flexible las peticiones permitidas entre dominios. Al proporcionar los encabezados HTTP apropiados para CORS, la API REST indica al navegador los dominios u orígenes autorizados para realizar llamadas al servicio REST con JavaScript.

95. El JSON con relleno (JSONP) es un método para enviar datos JSON sin que influyan cuestiones de petición entre dominios. Introduce funciones de retrollamada para cargar datos JSON de diferentes dominios, lo cual es posible porque la etiqueta HTML `<script>` no está afectada por la política de mismo origen. Todo lo que se importa a través de esa etiqueta se ejecuta inmediatamente en el contexto global. En lugar de pasar un archivo JavaScript, se puede pasar un URL a un servicio que devuelva el código JavaScript.

96. Para sortear la restricción que impide realizar peticiones entre dominios se suelen seguir los siguientes enfoques:

- JSONP es una solución para permitir las peticiones entre dominios. No ofrece ningún mecanismo de detección de errores, es decir, si hay un problema y el servicio falla o responde con un error HTTP, no hay forma de que el cliente pueda saber qué ocurre. Simplemente, la aplicación Ajax se quedará colgada. Además, el sitio que usa JSONP confiará incondicionalmente en el JSON proporcionado desde un dominio diferente;
- Otra alternativa para permitir peticiones entre dominios es el elemento HTML `iframe`. Aplicando el método JavaScript `window.postMessage (message, targetOrigin)` al elemento `iframe` es posible pasar una petición a un sitio de un dominio diferente. El uso de `iframe` es compatible incluso con navegadores antiguos. Además, solo es compatible con `GET`. El origen de la página `iframe` siempre debe comprobarse por razones de seguridad; y
- CORS es un enfoque estandarizado que permite realizar una llamada a un dominio externo. Puede utilizar `XMLHttpRequest` para enviar y recibir datos y tiene un mecanismo de control de errores mejor que JSONP. Es compatible con muchos tipos de autorización en comparación con JSONP, que solo admite cookies. También es compatible con diversos métodos HTTP mientras que JSONP solo lo es con `GET`. Por otro lado, no siempre es posible implementar CORS, ya que los navegadores tienen que admitirlo y los consumidores de la API deben estar incluidos en la lista blanca de CORS.

[RSG-148] Si la API REST es pública, el valor del encabezado HTTP `Access-Control-Allow-Origin` DEBE ser '*'.

[RSG-149] Si la API REST está protegida, DEBERÍA utilizarse CORS, si es posible. De lo contrario, PUEDE utilizarse JSONP como alternativa, pero solo para peticiones `GET`, por ejemplo, cuando el usuario accede con un navegador antiguo. NO DEBERÍA utilizarse `iframe`.

Modelo de madurez de la API

97. Las API REST suelen clasificarse mediante un modelo de madurez. Aunque existen varios modelos, la presente norma hace referencia al modelo de madurez de Richardson, que establece cuatro niveles de madurez (de 0 a 3). Esta norma recomienda el nivel 2 para las API REST, ya que el nivel 3 resulta complejo de aplicar, requiere una importante inversión conceptual y de desarrollo por parte de los proveedores y consumidores de servicios, y no produce un beneficio inmediato para los consumidores de servicios.

98. Si una API web está en el nivel 3 del modelo de madurez de Richardson, debe utilizarse un formato de hipermedia. El lenguaje de aplicación de hipertexto (HAL)[13] es simple y es compatible con las respuestas JSON y XML. Sin embargo, se

[13] https://tools.ietf.org/html/draft-kelly-json-hal-08t

trata solo de una posible recomendación, junto con otros formatos de hipermedia, como el JSON-LD[14]. Debería utilizarse el esquema JSON[15] porque, aunque actualmente no existe una especificación para el nivel 3 del modelo de madurez de Richardson, es el que se considera más maduro. No deberían utilizarse los siguientes formatos de hipermedia: la Norma RFC 5988 del IETF y `Collection+JSON`.

99. Se recomienda que las instancias descritas por un esquema proporcionen un enlace a un esquema JSON descargable utilizando la relación de enlace `describedby`, tal como se define en la sección 8.1 del *Linked Data Protocol 1.0 [W3C.REC-ldp-20150226]*[16].

En HTTP, esas relaciones de enlace pueden añadirse a cualquier respuesta usando el encabezado `Link`, con arreglo a la Norma RFC 8288. A continuación figura un ejemplo de uso de dicho encabezado:

```
Link: <http://example.com/my-hyper-schema#>; rel="describedby"
```

[RSJ-150] Si se utilizan instancias descritas por un esquema, DEBERÍA utilizarse el encabezado `Link` para proporcionar un enlace a un esquema JSON descargable de acuerdo con la Norma RFC 8288.

[RSJ-151] Una API web DEBERÍA alcanzar al menos el nivel 2 (propiedades nativas de transporte) del modelo de madurez de Richardson. Se PUEDE implementar el nivel 3 (hipermedia) que permite descubrir la API en su totalidad.

100. Se puede diseñar un formato de hipermedia personalizado. En ese caso, se recomiendan una serie de atributos. Por ejemplo:

```
{
  "link": {
     "href": "/patents",
     "rel": "self"
  },
   ...
}
```

[RSJ-152] Para diseñar un formato de hipermedia personalizado DEBERÍAN utilizarse los siguientes atributos en el enlace de atributos:

- `href`: el URI de destino;
- `rel`: el significado del URI de destino;
- `self`: el URI hace referencia al propio recurso;
- `next`: el URI hace referencia a la página siguiente (si se utiliza durante la paginación);
- `previous`: el URI hace referencia a la página anterior (si se utiliza durante la paginación); y
- el nombre arbitrario `v` indica el significado personalizado de una relación.

API WEB SOAP

101. La presente norma recomienda el estilo de arquitectura REST como enfoque preferido para el diseño de las API. Las arquitecturas RESTful generalmente se diseñan, amplían e integran más fácilmente que las SOAP. Se incluye, no obstante, la arquitectura SOAP para que la norma sea más exhaustiva; no se proporcionan ejemplos ni casos de uso.

[14] https://www.w3.org/TR/json-ld/
[15] https://json-schema.org/specification.html#specification-documents
[16] http://json-schema.org/latest/json-schema-core.html#hypermedia

102. Una API web SOAP es una aplicación de *software* identificada por URI, cuyas interfaces y vinculaciones pueden ser definidas, descritas y descubiertas por objetos XML. También admite interacciones directas con otras aplicaciones de *software* que utilizan mensajes basados en XML a través de protocolos de Internet como SOAP y HTTP.

103. Un contrato basado en SOAP se describe en un lenguaje de descripción de servicios web (WSDL) establecido en un documento normativo del W3C. A lo largo del presente documento se utilizará WSDL para hacer referencia al contrato del servicio web definido en el documento WSDL.

104. Hay dos enfoques de desarrollo de los servicios web: *contract-last* (contrato al final, también conocido como *code-first*, código al principio) y *contract-first* (contrato al principio). Cuando se utiliza el enfoque de *contract-last*, se empieza con el código, y a partir de él se desarrolla el contrato de servicio web. Con el enfoque de *contract-first*, se empieza con el contrato WSDL, y se utiliza el código para implementar dicho contrato.

Normas generales

105. El perfil de interoperabilidad de servicios web o perfil WS-I es una de las normas más importantes en lo que respecta a las API SOAP, y proporciona una referencia para formular las especificaciones de servicios web que pueden funcionar conjuntamente. La WS-I proporciona directrices sobre cómo los servicios están expuestos entre sí y cómo transfieren la información (lo que se denomina mensajería). Es un perfil para implementar versiones específicas de algunos de los estándares de servicios web más importantes, como WSDL, SOAP o XML. La adhesión a determinados perfiles indica implícitamente la adhesión a versiones específicas de esos estándares de servicios web. *WS-I Basic Profile v1.1* proporciona una guía para el uso de XML 1.0, HTTP 1.1, UDDI, SOAP 1.1, WSDL 1.1 y UDDI 2.0. *WS-I Basic Profile 2.0* constituye una guía para utilizar SOAP 1.2, WSDL 1.1, UDDI 2.0, *WS-Addressing* y MTOM. SOAP 1.2 ofrece un modelo de procesamiento claro y contribuye a una mejor interoperabilidad. WSDL 2.0 se diseñó para resolver los problemas de interoperabilidad encontrados en WSDL 1.1 utilizando enlaces SOAP 1.2 mejorados.

[WS-01] Todos los WSDL DEBEN ajustarse al *WS-I Basic Profile 2.0*. PUEDE utilizarse WSDL 1.2.

106. Un enlace SOAP WSDL puede ser de estilo llamada a procedimiento remoto (RPC) o de estilo documento, y un enlace SOAP puede tener un uso codificado o literal, por lo que hay cinco modelos de estilo/uso posibles: RPC/codificado *(RPC/encoded*), RPC/literal *(RPC/literal*), documento/codificado (*document/encoded*), documento/literal (*document/literal*), y documento/literal envuelto (*document/literal wrapped*).

[WS-02] Los servicios DEBEN seguir los modelos de estilo documento y de uso literal (ya sea documento/literal o documento/literal envuelto). Cuando haya gráficos, DEBE utilizarse el modelo RPC/codificado.

[WS-03] Cuando hay casos de uso excepcionales, como las operaciones WDSL sobrecargadas, DEBERÍAN utilizarse todos los demás estilos.

107. La parte WSDL concreta debería separarse de la parte WSDL abstracta para proporcionar una interfaz más modular y flexible. La parte WSDL abstracta define los tipos de datos, los mensajes, las operaciones y el tipo de puerto. La parte WSDL concreta define el enlace, el puerto y el servicio.

[WS-04] El WSDL DEBERÍA separarse en una parte abstracta y otra concreta.

[WS-05] Todos los tipos de datos DEBERÍAN definirse en un archivo XSD e importarse en la parte WSDL abstracta.

[WS-06] La parte WSDL concreta DEBE definir un solo servicio con un puerto.

Esquemas

108. Los esquemas utilizados en el WSDL deben ajustarse a la Norma ST.96 de la OMPI. A efectos de reutilización y modularidad, los esquemas deben ser documentos separados que se incluyan o importen en el WSDL, en lugar de ser definidos directamente en el WSDL. Esto permitirá realizar cambios en la estructura XML sin modificar el WSDL.

[WS-07] El esquema definido en el elemento `wsdl:types` DEBE ser importado de un archivo de esquema independiente, para permitir la modularidad y la reutilización.

[WS-08] La importación de un esquema externo DEBE implementarse usando el elemento `xsd:import`, y no el elemento `xsd:include`.

[WS-09] El elemento `xsd:any` NO DEBE utilizarse para especificar un elemento raíz en el cuerpo del mensaje.

[WS-10] El espacio de nombres de destino para el WSDL (atributo `targetNamespace` en `wsdl:definitions`) DEBE ser diferente del espacio de nombres de destino del esquema (atributo `targetNamespace` en `xsd:schema`).

[WS-11] Las peticiones y respuestas (convención de nombres, formato de mensajes, estructura de datos y diccionario de datos) DEBERÍAN ajustarse a la Norma ST.96 de la OMPI.

Nombres y versiones

109. En la asignación de nombres a los servicios y a los elementos WSDL deberían aplicarse convenciones de nombres adecuadas, conformes con las aplicadas en la Norma ST.96 de la OMPI.

[WS-12] Los nombres de servicios DEBEN seguir la convención *upper camel case* y tener el sufijo 'Service'. Por ejemplo, `https://wipo.int/PatentsService`.

[WS-13] Los nombres de elementos WSDL —`message` (mensaje), `part` (parte), `porType` (tipo de puerto), `operation` (operación), `input` (entrada), `output` (salida) y `binding` (enlace)— DEBERÍAN seguir la convención *upper camel case.*

[WS-14] Los nombres de mensajes de petición DEBERÍAN tener el sufijo 'Request'.

[WS-15] Los nombres de mensajes de respuesta DEBERÍAN tener el sufijo 'Response'.

[WS-16] Los nombres de operaciones DEBERÍAN seguir el formato `<Verb><Object>{<Qualifier>}`, donde `<Verb>` indica la operación (preferiblemente `Get`, `Create`, `Update`, o `Delete`, según corresponda) sobre el `<Object>` de la operación, seguido opcionalmente por un `<Qualifier>` del `<Object>`.

110. Todos los nombres de operaciones tendrán al menos dos partes. Se podrá incluir una tercera parte opcional para aclarar y/o especificar el propósito de la operación. Las tres partes son: `<Verb>`, `<Object>` y `<Qualifier>` (opcional). Cada parte se describe en detalle a continuación.

Verbo. El nombre de cada operación comenzará con un verbo. A continuación figuran ejemplos de verbos de uso común:

Verbo	Descripción	Ejemplo
`Get`	Obtener un único objeto	GetBibData
`Create`	Obtener un objeto nuevo	CreateBibData
`Update`	Actualizar un objeto	UpdateBibData
`Delete`	Eliminar un objeto	DeleteCustomer

Objeto. El sustantivo que sigue al verbo será una descripción sucinta e inequívoca de la función de la operación. El objetivo es que los consumidores entiendan sin ambigüedades lo que produce la operación. Dado que la definición de algunas entidades no es común en los distintos centros de costes, el objeto puede ser un campo compuesto en el que el primer nodo es el centro de costes y el segundo nodo la entidad. Por ejemplo, `PatentCustomer`.

Calificador. El propósito del atributo opcional `qualifier` del objeto es aclarar el dominio operacional o temático (por ejemplo, `GetCustomerList`). `Get` indica la operación que se realizará para el cliente y `List` especifica que el propósito es obtener una lista de clientes y no solo un cliente como en `GetCustomer`.

111. De acuerdo con los principios de diseño orientado a servicios, los proveedores y consumidores de servicios deberían evolucionar de forma independiente. El consumidor de servicios no debería verse afectado por cambios menores (retrocompatibles) por parte del proveedor de servicios. Por consiguiente, el versionado de servicios debería utilizar solo números de versión mayores. Para las API internas (por ejemplo, para el desarrollo y las pruebas) también pueden utilizarse versiones menores, como las versiones semánticas.

[WS-17] El nombre del archivo WSDL DEBERÍA ajustarse al siguiente patrón: <nombre del servicio>_V<número de versión mayor>.

[WS-18] El espacio de nombres del archivo WSDL DEBERÍA contener la versión del servicio. Por ejemplo, `https://wipo.int/PatentsService/V1”.`

112. La descripción del servicio y sus operaciones se proporciona como documentación WSDL.

[WS-19] El elemento `wsdl:documentation` DEBERÍA utilizarse en el WSDL con la descripción del servicio (como primer elemento hijo de `wsdl:definitions` en el WSDL) y sus operaciones.

Diseño del contrato de servicio web

113. Un contrato de servicio web debería incluir una interfaz técnica compuesta por un lenguaje de descripción de servicios web (WSDL), definiciones de esquemas XML y una política de servicios web (*WS-Policy)*, así como una interfaz no técnica integrada por uno o más documentos de descripción de servicios.

114. El WSDL, parte del contrato de servicio, debe ser diseñado antes de cualquier desarrollo de código. En ningún caso deber ser generado automáticamente a partir del código. La máxima debe ser *contract-first* y no *code-first*. Todos los contratos de servicio web deben ajustarse al *WS-I Basic Profile*. Cualquier proyecto que se genere automáticamente a partir del código podrá ser objeto de modificaciones para asegurar su conformidad con las normas.

Adhesión de políticas a las definiciones WSDL

115. Los contratos de servicios web pueden ampliarse con políticas de seguridad que incluyen limitaciones, condiciones y cualidades complementarias normalmente relacionadas con el funcionamiento de los servicios. Las políticas de seguridad pueden formularse en formato legible por personas y formar parte de un acuerdo de nivel de servicio complementario, o en formato legible por máquinas en el momento de ejecución. Las políticas legibles por máquinas se definen usando el lenguaje *WS-Policy* y las especificaciones conexas.

[WS-20] Las definiciones de políticas DEBEN figurar por separado en un documento de definición de *WS-Policy*, al que se hace referencia en el documento WSDL mediante el elemento `wsp:PolicyReference.`

[WS-21] Las políticas globales o específicas de un dominio DEBERÍAN ser independientes y aplicables a múltiples servicios.

[WS-22] Los puntos de adhesión de políticas DEBERÍAN ser compatibles con WSDL 1.1 o una versión posterior, preferiblemente la versión 2.0, los elementos de los puntos de adhesión y los temas de políticas conexas (servicio, punto final, operación y mensaje).

SOAP - Seguridad de servicios web

116. La seguridad de servicios web (WSS): seguridad de mensajes SOAP (*WSS: SOAP Message Security*) consiste en un conjunto de mejoras en la mensajería SOAP que proporciona integridad y confidencialidad a los mensajes. *WSS: SOAP Message Security* es ampliable y puede incluir una serie de modelos de seguridad y tecnologías de cifrado. *WSS: SOAP Message Security* proporciona tres mecanismos principales que pueden utilizarse de forma independiente o conjuntamente:

- La posibilidad de enviar *tokens* de seguridad como parte de un mensaje, y de asociarlos con el contenido del mensaje;
- La posibilidad de proteger el contenido de un mensaje de modificaciones no autorizadas y no detectadas (integridad del mensaje); y

– La posibilidad de proteger el contenido de un mensaje de una divulgación no autorizada (confidencialidad del mensaje).

WSS: SOAP Message Security puede utilizarse conjuntamente con otras extensiones de servicios web y protocolos específicos de cada aplicación con el fin de satisfacer una serie de requisitos de seguridad.

[WS-23] Los servicios web que utilizan mensajes SOAP DEBERÍAN estar protegidos de acuerdo con las recomendaciones de *WSS: SOAP Message Security*.

FORMATOS DE TIPOS DE DATOS

117. En la presente norma se recomiendan formatos de tipos de datos primarios, como la hora, la fecha y el idioma, conforme a las recomendaciones de las Normas ST.96 y ST.97 de la OMPI, que se utilizan para las peticiones y respuestas en XML y JSON, respectivamente, y para los parámetros de consulta.

[CS-01] Los objetos de hora DEBEN tener el formato especificado en la Norma RFC 3339 del IETF (es un perfil de la ISO 8601).

[CS-02] La información de la zona horaria DEBERÍA especificarse conforme a la Norma RFC 3339 del IETF. Por ejemplo: `20:54:21+00:00`.

[CS-03] Los objetos de fecha DEBEN tener el formato especificado en la Norma RFC 3339 del IETF (es un perfil de la ISO 8601). Por ejemplo: `2018-10-19`.

[CS-04] Los objetos de fecha y hora (es decir, de marca de tiempo) DEBEN tener el formato especificado en la Norma RFC 3339 del IETF (es un perfil de la ISO 8601).

[CS-05] La zona horaria aplicable DEBERÍA especificarse conforme a la Norma RFC 3339 del IETF. Por ejemplo: `2017-02-14T20:54:21+00:00`.

[CS-06] DEBE usarse la Norma ISO 4217 alfa-3: *Códigos de dividas* para representar las divisas. La precisión del valor (es decir, el número de dígitos después del punto decimal) PUEDE variar dependiendo de los requisitos operacionales.

[CS-07] DEBEN utilizarse los códigos de dos letras de la Norma ST.3 de la OMPI para referirse a las oficinas de propiedad intelectual, los Estados y otras entidades y organizaciones, así como a los países o las organizaciones designados y prioritarios.

[CS-08] DEBE utilizarse la Norma ISO 3166-1 alfa-2: *Códigos de representación de nombres de países y sus subdivisiones* para representar los nombres de países, dependencias y demás zonas de especial interés geopolítico, a partir de las listas de nombres de países de las Naciones Unidas.

[CS-09] DEBE usarse la Norma ISO 639-1 alfa-2: *Códigos de representación de nombres de idiomas* para los códigos de idiomas.

[CS-10] DEBERÍAN utilizarse las unidades de medida conforme a lo descrito en el *Unified Code for Units of Measure* (basado en las definiciones de la Norma ISO 80000). Por ejemplo, la unidad de medida de la masa es el kilogramo (kg).

[CSJ-11] Los caracteres utilizados en las enumeraciones DEBEN limitarse al conjunto siguiente: {a-z, A-Z, 0-9, punto ".", coma ",", espacio " ", guion "-" y guion bajo "_"}.

[CSJ-12] DEBEN utilizarse los términos de representación del Anexo VI para los nombres de propiedades atómicas.

[CSJ-13] Las siglas y las abreviaturas que aparecen al principio del nombre de una propiedad DEBEN estar en minúsculas. El resto de siglas y abreviaturas y valores de enumeraciones DEBEN aparecer en mayúsculas.

CONFORMIDAD

118. La presente norma está diseñada como un conjunto de normas de diseño y convenciones que se pueden incorporar a las API para servicios web existentes o nuevas con objeto de proporcionar una funcionalidad común. No todos los servicios serán compatibles con todas las convenciones definidas en la norma debido a cuestiones comerciales (por ejemplo, es posible que no se requiera la función calidad de servicio) o a limitaciones técnicas (por ejemplo, es posible que ya se utilice OAuth 2.0).

119. La presente norma define dos niveles de conformidad: A y AA. Conviene tener en cuenta que las normas de diseño que incluyen la palabra PUEDE no se tienen en cuenta a la hora de determinar la conformidad.

120. Se recomienda que las API para servicios web admitan tantas funcionalidades adicionales, más allá de su nivel de conformidad, como sean apropiadas para su escenario de uso previsto.

121. Se definen dos niveles de conformidad:

- **Nivel A**: el nivel de conformidad A indica que la API sigue las normas generales de diseño (RSG) obligatorias, que en la presente norma son las que incluyen la palabra DEBE. Además, también se deben cumplir las normas específicas del tipo de respuesta devuelta. A continuación se indican los subniveles de conformidad:
    - Nivel AJ: si la respuesta devuelta está en el formato JSON de la Norma ST.97, deben cumplirse todas las normas generales de diseño (RSG) que incluyan la palabra DEBE, así como todas las normas específicas del formato de respuesta JSON (RSJ) que incluyan la palabra DEBE;
    - Nivel AX: si la respuesta devuelta es una instancia XML de la Norma ST.96, deben cumplirse todas las normas generales de diseño (RSG) que incluyan la palabra DEBE, así como todas las normas específicas del formato de respuesta XML (RSX) que incluyan la palabra DEBE; y
    - Nivel A: si la respuesta devuelta está en formato JSON o XML, deben cumplirse todas las normas generales de diseño (RSG) que incluyan la palabra DEBE, así como todas las normas de diseño específicas del formato de respuesta JSON (RSJ) que incluyan la palabra DEBE y todas las normas de diseño específicas del formato de respuesta XML (RSX) que incluyan la palabra DEBE.
- **Nivel AA:** el nivel de conformidad AA indica que la API satisface el nivel A y se siguen todas las normas de diseño recomendadas, que en la presente norma son las que incluyen la palabra DEBERÍA. Al igual que en el nivel A, hay subniveles de conformidad en función del tipo de respuesta, a saber:
    - Nivel AAJ: se cumple el nivel AJ, así como las normas de diseño recomendadas (que incluyen la palabra DEBERÍA) específicas del formato de respuesta JSON; y
    - Nivel AAX: se cumple el nivel AX, así como las normas de diseño recomendadas (que incluyen la palabra DEBERÍA) específicas del formato de respuesta XML.

122. En el Anexo I figura una matriz de trazabilidad que relaciona las normas de diseño con los niveles de conformidad.

MATERIAL DE REFERENCIA

Normas de la OMPI

Norma ST.3 de la OMPI	Códigos de dos letras recomendados para la representación de Estados, otras entidades y organizaciones intergubernamentales
Norma ST.96 de la OMPI	Tratamiento en XML de la información relativa a la propiedad intelectual
Norma ST.97 de la OMPI	Tratamiento en JSON de la información relativa a la propiedad intelectual

Normas y convenciones

Documento RFC 2119 del IETF	*Palabras* clave *para indicar distintos niveles de exigencia en documentos RFC* – www.ietf.org/rfc/rfc2119.txt
Norma RFC 3339 del IETF	*Fecha y hora en Internet: marcas de tiempo* –www.ietf.org/rfc/rfc3339.txt
Norma RFC 3986 del IETF	*Identificador* uniforme *de recursos (URI): sintaxis genérica* - www.ietf.org/rfc/rfc3986.txt
Norma RFC 5789 del IETF	*Método HTTP* PATCH – https://tools.ietf.org/rfc/rfc5789.txt
Norma RFC 5988 del IETF	*Vinculación web* – https://tools.ietf.org/rfc/rfc5988.txt
Documento RFC 6648 del IEFT	Desaconsejando *el prefijo "X-" y construcciones similares en los protocolos de aplicación* – https://tools.ietf.org/rfc/rfc6648.txt
Norma RFC 6750 del IETF	*El marco de* autorización *de OAuth 2.0: uso del token portador* – https://tools.ietf.org/rfc/rfc6750.txt
Norma RFC 7231 del IETF	*Protocolo de* transferencia *de hipertexto* (HTTP/1.1): semántica y contenido – www.ietf.org/rfc/rfc7231.txt
Norma RFC 7232 del IETF	*Protocolo de transferencia de hipertexto* (HTTP/1.1): peticiones condicionales – www.ietf.org/rfc/rfc7232.txt
Norma RFC 7234 del IETF	*Protocolo de transferencia de hipertexto (HTTP/1.1):* almacenamiento *en caché* – www.ietf.org/rfc/rfc7234.txt
Norma RFC 7386 del IETF	*JSON Merge Patch* – www.ietf.org/rfc/rfc7386.txt
Norma RFC 7240 del IETF	*Encabezado HTTP Prefer* – https://tools.ietf.org/rfc/rfc7240.txt
Norma RFC 7519 del IETF	JSON Web Token – www.ietf.org/rfc/rfc7519.txt
Norma RFC 7540 del IETF	*Protocolo de transferencia de hipertexto versión 2 (HTTP/2)* – https://tools.ietf.org/html/rfc7540
Documento BCP-47 del IETF	*Etiquetas para identificar idiomas* – https://tools.ietf.org/rfc/bcp/bcp47.txt
Norma ISO 639-1 alfa-2	*Códigos de representación de nombres de idiomas* – https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
Norma ISO 3166-1 alfa-2	*Códigos de representación de nombres de países y sus subdivisiones* – https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2
Norma ISO 3166-1 alfa-3	*Códigos de representación de nombres de países y sus subdivisiones* – https://en.wikipedia.org/wiki/ISO_3166-1_alpha-3
Norma ISO 4217	*Códigos de divisas* – www.iso.org/iso/home/standards/currency_codes.htm

Norma ISO 8601 *Formatos de fecha y hora* – https://en.wikipedia.org/wiki/ISO_8601

OData https://www.odata.org/

OData Metadata Service Entity Model de OASIS – http://docs.oasis-open.org/odata/odata/v4.0/os/models/MetadataService.edmx

OData JSON Format Version 4.0 de OASIS. Editado por Ralf Handl, Michael Pizzo y Mark Biamonte. Última versión – http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html

OData Atom Format Version 4.0 de OASIS. Editado por Martin Zurmuehl, Michael Pizzo y Ralf Handl. Última versión – http://docs.oasis-open.org/odata/odata-atom-format/v4.0/odata-atom-format-v4.0.html

OData Version 4.0 de OASIS

- *Part 1: Protocol*– http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html
- *Part 2: URL Conventions* – http://docs.oasis-open.org/odata/odata/v4.0/os/part2-url-conventions/odata-v4.0-os-part2-url-conventions.html
- *Part 3: Common Schema Definition Language (CSDL)* - http://docs.oasis-open.org/odata/odata/v4.0/os/part3-csdl/odata-v4.0-os-part3-csdl.html

ABNF components de OASIS *OData ABNF Construction Rules Version 4.0 and OData ABNF Test Cases* – http://docs.oasis-open.org/odata/odata/v4.0/os/abnf/

Vocabulary components de OASIS *OData Core Vocabulary, OData Measures Vocabulary and OData Capabilities Vocabulary* – http://docs.oasis-open.org/odata/odata/v4.0/os/vocabularies/

Esquemas XML de OASIS:

OData EDMX XML Schema and OData EDM XML Schema – http://docs.oasis-open.org/odata/odata/v4.0/os/schemas/

Security Assertion Markup Language (SAML) V2.0 Technical Overview de OASIS – http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html

Lenguaje de modelado de API RESTful (RAML) http://raml.org

OpenAPI Initiative www.openapis.org

Modelo de madurez de Richardson para las API REST – https://martinfowler.com/articles/richardsonMaturityModel.html

HAL – http://stateless.co/hal_specification.html

JSON-LD https://json-ld.org

Collection+JSON - Document Format http://amundsen.com/media-types/collection/format/

BadgerFish – http://badgerfish.ning.com/

Versionado semántico – https://semver.org/

REST – https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

CQL – https://en.wikipedia.org/wiki/Contextual_Query_Language

Z39.50 – https://www.loc.gov/z3950/agency/Z39-50-2003.pdf

WS-I Basic Profile 2.0 – http://ws-i.org/profiles/basicprofile-2.0-2010-11-09.html

W3C SOAP 1.2 Part 1 Messaging Framework – https://www.w3.org/TR/soap12-part1/

W3C SOAP 1.2 Part 2 Adjuncts – https://www.w3.org/TR/soap12-part2/

W3C WSDL Version 2.0 Part 1 Core Language – https://www.w3.org/TR/wsdl20/

W3C CORS - https://www.w3.org/TR/cors/

W3C Matrix Parameters – https://www.w3.org/DesignIssues/MatrixURIs.html

API REST de las oficinas de PI

EPO, *Open Patent Services*	*OPS v 3.2* – https://developers.epo.org
USPTO, PatentsView	http://www.patentsview.org/api/doc.html
WIPO, ePCT v1.1	https://pct.wipo.int/
EUIPO, TMview, Designview, TMclass	http://www.tm-xml.org/TM-XML/TM-XML_xml/TM-XML_TM-Search.xml

API REST de empresas y directrices de diseño

Facebook	https://developers.facebook.com/docs/graph-api/reference
GitHub	https://developer.github.com/v3
Google: Guía de diseño de API	https://cloud.google.com/apis/design/
Azure	https://docs.microsoft.com/en-us/rest/api/
OpenAPI	https://swagger.io/docs/specification/about/
OData	http://www.odata.org/documentation/
JSON API	http://jsonapi.org/format/
Microsoft *API Design*	https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
API REST de JIRA	https://developer.atlassian.com/server/jira/platform/jira-rest-api-examples
API REST de Confluence	https://developer.atlassian.com/server/confluence/
API de Ebay	https://developer.ebay.com/api-docs/static/ebay-rest-landing.html
Oracle REST *Data Services*	http://www.oracle.com/technetwork/developer-tools/rest-data-services/overview/index.html
API REST de PayPal	https://developer.paypal.com/docs/api/overview/
Data on the Web Best Practices	https://www.w3.org/TR/dwbp/#intro
SAP Guidelines *for Future REST API Harmonization*	https://d.dam.sap.com/m/xAUymP/54014_GB_54014_enUS.pdf
API GitHub	https://developer.github.com/v3/

Zalando	https://github.com/zalando/ReSTful-api-guidelines
Dropbox	https://www.dropbox.com/developers
Twitter	https://developer.twitter.com/en/docs

Otros

CQRS	https://martinfowler.com/bliki/CQRS.html
ITU	https://www.itu.int/en/ITU-T/ipr/Pages/open.aspx
OWASP Rest Security Cheat Sheet	https://www.owasp.org/index.php/REST_Security_Cheat_Sheet
DDD	https://martinfowler.com/bliki/BoundedContext.html
Principios de la arquitectura REST	https://en.wikipedia.org/wiki/Representational_state_transfer
Principio de abierto/cerrado	https://en.wikipedia.org/wiki/Open/closed_principle
Which style of WSDL should I use? (¿Qué estilo de WSDL debería usar?)	https://www.ibm.com/developerworks/library/ws-whichwsdl/

Gobierno de Nueva Zelandia

API Standard and Guidelines	https://www.ict.govt.nz/guidance-and-resources/standards-compliance/api-standard-and-guidelines/
Cross site scripting prevention cheat sheet	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
OWASP Cheat Sheet Series	https://cheatsheetseries.owasp.org/
Digital Signature Standard (DSS)	https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.186-4.pdf

SOAP Message Security 1.0, OASIS Standard 200401 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

SOA Principles of Service Design, Thomas Erl (2008)

[Sigue el Anexo I]

ANEXO I

LISTA DE NORMAS DE DISEÑO Y CONVENCIONES DE SERVICIOS WEB RESTFUL

Versión 1.1

Revisión aprobada por el Comité de Normas Técnicas de la OMPI (CWS)
en su décima sesión, celebrada el 25 de noviembre de 2022

En los cuadros del presente Anexo se resumen las normas de diseño y las convenciones de los servicios web RESTful, y se especifican los requisitos básicos exigidos para alcanzar un determinado nivel de conformidad, con el fin de facilitar la implementación de las API para servicios web. A continuación se proporciona información sobre cada cuadro:

- En el Cuadro 1 se resumen las normas de diseño que deben cumplirse para alcanzar el nivel AJ de conformidad (respuesta devuelta en formato JSON);
- En el Cuadro 2 se resumen las normas de diseño que deben cumplirse para alcanzar el nivel AX de conformidad (respuesta devuelta en formato XML);
- En el Cuadro 3 se resumen las normas de diseño que deben cumplirse para alcanzar el nivel AAJ de conformidad (respuesta devuelta en formato JSON); y
- En el Cuadro 4 se resumen las normas de diseño que deben cumplirse para alcanzar el nivel AAX de conformidad (respuesta devuelta en formato XML).

[Nota editorial: para lograr el nivel A de conformidad solo hay que seguir las normas de los cuadros 1 y 2. Para lograr el nivel AA de conformidad es necesario seguir las normas de los cuadros 3 y 4. La tercera letra indica el tipo de respuesta devuelta.]

Cuadro 1. Nivel de conformidad para respuesta en formato JSON

ID de la norma	Descripción de la norma	Referencias cruzadas y observaciones
[RSG-01]	DEBE usarse la barra, "/", en la ruta del URI para indicar la relación jerárquica entre los recursos, pero la ruta NO DEBE terminar con una barra, ya que esta no proporciona ningún valor semántico y puede llevar a confusión.	AJ, AX, AAJ, AAX
[RSG-02]	Los nombres de los recursos DEBEN seguir un patrón de nomenclatura coherente.	AJ, AX, AAJ, AAX
[RSG-04]	Los parámetros de consulta DEBEN seguir un patrón de nomenclatura coherente.	AJ, AX
[RSG-06]	El patrón del URL para una API web DEBE contener la palabra 'api' en el URI.	AJ, AX, AAJ, AAX
[RSG-07]	NO DEBEN utilizarse parámetros matriciales.	AJ, AX, AAJ, AAX
[RSG-08]	Una API web DEBE aplicar sistemáticamente códigos de estado HTTP según se describe en los documentos RFC del IETF.	AJ, AX, AAJ, AAX
[RSG-10]	Si la API detecta valores de entrada incorrectos, DEBE devolver el código de estado HTTP `400 Bad Request`. La carga útil de error DEBE indicar el valor incorrecto.	AJ, AX, AAJ, AAX
[RSG-12]	Si la API detecta valores válidos que requieren funcionalidades no implementadas, DEBE devolver el código de estado HTTP `501 Not Implemented`. La carga útil de error DEBE indicar el valor sin implementar.	AJ, AX, AAJ, AAX
[RSG-14]	Si un recurso puede ser independiente, DEBE ser un recurso de nivel superior; de lo contrario será un subrecurso.	AJ, AX, AAJ, AAX
[RSG-15]	Para obtener recursos anidados, DEBEN utilizarse parámetros de consulta en lugar de rutas URL.	AJ, AX, AAJ, AAX
[RSG-18]	Los nombres de los recursos, los segmentos y los parámetros de consulta DEBEN estar compuestos por palabras en inglés, conforme a la grafía del inglés básica del Diccionario de Oxford (Oxford English	AJ, AX, AAJ, AAX

	Dictionary). PUEDEN estar en otros idiomas los nombres de recursos que deben traducirse por razones comerciales.	
[RSG-20]	Una API web DEBE ser compatible con la negociación del tipo de contenido conforme a la Norma RFC 7231 del IETF.	AJ, AX, AAJ, AAX
[RSG-21]	DEBE utilizarse el formato JSON cuando no se solicita un tipo de contenido específico.	AJ, AX, AAJ, AAX
[RSG-27]	Una API web DEBE ser compatible como mínimo con el formato XML o JSON.	AJ, AX, AAJ, AAX
[RSG-28]	Los métodos HTTP DEBEN limitarse a los métodos estándares `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` y `HEAD`, conforme a las especificaciones de las normas RFC 7231 y RFC 5789 del IETF.	AJ, AX, AAJ, AAX
[RSG-33]	Para un punto final que extrae un solo recurso, si este no se encuentra, el método GET DEBE devolver el código de estado `404 Not Found`. En el caso de los puntos finales que devuelven listas de recursos, simplemente devolverán una lista vacía.	AJ, AX, AAJ, AAX
[RSG-34]	Si un recurso es recuperado con éxito, el método `GET` DEBE devolver `200 OK`.	AJ, AX, AAJ, AAX
[RSG-35]	Las peticiones `GET` DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-37]	Las peticiones `HEAD` DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-39]	Las peticiones `POST` NO DEBEN ser idempotentes de acuerdo con la Norma RFC 2616 del IETF.	AJ, AX, AAJ, AAX
[RSG-43]	Las peticiones `PUT` DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-44]	Si no se encuentra un recurso, `PUT` DEBE devolver el código de estado `404 Not Found`.	AJ, AX, AAJ, AAX
[RSG-45]	Si un recurso se actualiza correctamente, `PUT` DEBE devolver el código de estado `200 OK` si se devuelve el recurso actualizado o `204 No Content` si no se devuelve.	AJ, AX, AAJ, AAX
[RSG-46]	Las peticiones `PATCH` NO DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-48]	Si un recurso no se encuentra, `PATCH` DEBE devolver el código de estado `404 Not Found`.	AJ, AX, AAJ, AAX
[RSJ-49]	Si una API web implementa actualizaciones parciales mediante `PATCH`, DEBE utilizar el formato `JSON Merge Patch` para describir el conjunto de cambios parciales, como se especifica en la Norma RFC 7386 del IETF (utilizando `application/merge-patch+json` como `Content-Type`).	AJ, AAJ
[RSG-50]	Las peticiones `DELETE` NO DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-51]	Si no se encuentra un recurso, `DELETE` DEBE devolver el código de estado `404 Not Found`.	AJ, AX, AAJ, AAX
[RSG-52]	Si un recurso se elimina correctamente, `DELETE` DEBE devolver el código de estado `200 OK` si se devuelve el recurso eliminado o `204 No Content` si no se devuelve.	AJ, AX, AAJ, AAX
[RSG-53]	El destinatario final es el servidor de origen o el primer *proxy* o puerta de enlace que recibe un valor del encabezado `Max-Forwards` de cero en la petición. Las peticiones `TRACE` NO DEBEN incluir un cuerpo.	AJ, AX, AAJ, AAX
[RSG-54]	Las peticiones `TRACE` NO DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-55]	El valor del encabezado HTTP `Via` DEBE servir para rastrear la cadena de petición.	AJ, AX, AAJ, AAX
[RSG-56]	Para que el cliente pueda limitar la longitud de la cadena de petición DEBE utilizarse el encabezado HTTP `Max-Forwards`.	AJ, AX, AAJ, AAX
[RSG-58]	Las respuestas a `TRACE` NO DEBEN ser almacenadas en caché.	AJ, AX, AAJ, AAX
[RSG-60]	Las peticiones `OPTIONS` DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-70]	Las API web DEBEN utilizar parámetros de consulta para implementar la paginación.	AJ, AX, AAJ, AAX
[RSG-71]	Las API web NO DEBEN utilizar encabezados HTTP para implementar la paginación.	AJ, AX, AAJ, AAX

[RSG-75]	Para establecer criterios multiatributos de ordenación, DEBE utilizarse un parámetro de consulta. El valor de dicho parámetro será una lista de claves de ordenación separadas por comas a las que se puede añadir, mediante dos puntos ':', la dirección de ordenamiento, con `asc` para el orden ascendente o `desc` para el orden descendente. En caso de que no se especifique una dirección de ordenación para una clave, el servidor DEBE establecer una por defecto.	AJ, AX, AAJ, AAX
[RSG-76]	Las API web DEBERÍAN devolver los criterios de ordenación en la respuesta.	AJ, AX, AAJ, AAX
[RSG-79]	Las API web DEBEN admitir la devolución del número de elementos de una colección.	AJ, AX, AAJ, AAX
[RSG-80]	Para permitir la devolución del número de elementos de una colección DEBE utilizarse un parámetro de consulta.	AJ, AX, AAJ, AAX
[RSG-82]	Las API web PUEDEN admitir la devolución del número de elementos de una colección mediante una función *inline*, es decir, como parte de la respuesta que contiene la propia colección. DEBE utilizarse un parámetro de consulta.	AJ, AX, AAJ, AAX
[RSG-86]	Un contrato de servicio DEBE especificar la gramática compatible (como los campos, las funciones, las palabras clave y los operadores).	AJ, AX, AAJ, AAX
[RSG-87]	DEBE utilizarse el parámetro de consulta '`q`'.	AJ, AX, AAJ, AAX
[RSG-88]	En el nivel de protocolo, una API web DEBE devolver un código de estado HTTP apropiado seleccionado de la lista de códigos de estado HTTP estándares.	AJ, AX, AAJ, AAX
[RSJ-89]	En el nivel de aplicación, una API web DEBE devolver una carga útil que informe del error con suficiente granularidad. Los atributos `code` y `message` son obligatorios, el atributo `details` es condicionalmente obligatorio y los atributos `target`, `status`, `moreInfo`, e `internalMessage` son opcionales.	AJ, AX, AAJ, AAX
[RSG-90]	Los errores NO DEBEN exponer datos esenciales para la seguridad o detalles técnicos internos, como las pilas de llamada en los mensajes de error.	AJ, AX, AAJ, AAX
[RSG-91]	El encabezado HTTP `Reason-Phrase` (descrito en la Norma RFC 2616) NO DEBE utilizarse para transmitir mensajes de error.	AJ, AX, AAJ, AAX
[RSG-93]	El modelo de contrato de servicio DEBE especificar lo siguiente: – la versión de la API; – información sobre la semántica de los elementos de la API; – los recursos; – los atributos de los recursos; – los parámetros de consulta; – los métodos; – los tipos de medios; – la gramática de búsqueda (si se admite alguna); – los códigos de estado HTTP; – los métodos HTTP; – las restricciones y las características distintivas; y – la seguridad (por ejemplo, los esquemas privados).	AJ, AX, AAJ, AAX
[RSG-95]	Una API REST DEBE proporcionar la documentación de la API como un contrato de servicio.	AJ, AX, AAJ, AAX
[RSG-96]	Toda implementación de una API web que no cumpla con la presente norma DEBE ser documentada explícitamente en el contrato de servicio. Si no se especifica ninguna desviación de una de sus normas en el contrato de servicio, DEBE asumirse que se sigue la presente norma.	AJ, AX, AAJ, AAX
[RSG-97]	Un contrato de servicio DEBE permitir la generación de código esquemático de cliente API.	AJ, AX, AAJ, AAX

[RSG-105]	Las API web DEBEN admitir el almacenamiento en caché de los resultados de `GET`; y PUEDEN admitir el almacenamiento en caché de los resultados de otros métodos HTTP.	AJ, AX, AAJ
[RSG-113]	Si una API web es compatible con la gestión de preferencias, en el contrato de servicio DEBE indicarse la nomenclatura de las preferencias que PUEDEN establecerse mediante el encabezado `Prefer`.	AAJ, AAX, AJ, AX
[RSG-114]	Si una API web permite la traducción de datos, DEBE admitir el encabezado de petición HTTP `Accept-Language` con el que se indica el conjunto de lenguas naturales preferidas para la respuesta, conforme a lo especificado en la Norma RFC 7231 del IETF.	AJ, AX, AAJ, AAX
[RSG-116]	Confidencialidad: las API y su información DEBEN ser identificadas, clasificadas y protegidas en todo momento frente a acciones no autorizadas de acceso, difusión e interceptación. DEBEN seguirse los principios de mínimos privilegios, cero confianza, necesidad de conocer y necesidad de compartir.	AJ, AX, AAJ, AAX
[RSG-117]	Garantía de integridad: las API y su información DEBEN estar protegidas contra acciones no autorizadas de modificación, duplicación, corrupción y destrucción. La modificación de la información DEBE realizarse mediante transacciones e interfaces aprobadas. La actualización de los sistemas DEBE llevarse a cabo mediante procesos aprobados de gestión de la configuración, gestión de cambios y gestión de parches.	AJ, AX, AAJ, AAX
[RSG-118]	Disponibilidad: las API y su información DEBEN estar disponibles para los usuarios autorizados en el debido momento, conforme se especifica en los acuerdos de nivel de servicio, las políticas de control de acceso y los procesos operativos definidos.	AJ, AX, AAJ, AAX
[RSG-119]	No repudio: toda transacción procesada o acción realizada por las API DEBE garantizar el no repudio mediante la auditoría, autorización y autenticación adecuadas, rutas seguras y servicios y mecanismos de no repudio.	AJ, AX, AAJ, AAX
[RSG-120]	Autenticación, autorización, auditoría: los usuarios, los sistemas, las API o los dispositivos que participen en transacciones o acciones importantes DEBEN ser autenticados, autorizados mediante servicios de control de acceso basados en roles o atributos y mantener la separación de tareas. Además, todas las acciones DEBEN ser registradas y el nivel de autenticación debe aumentar en función del riesgo asociado a la información.	AJ, AX, AAJ, AAX
[RSG-121]	En el desarrollo de las API DEBEN tenerse debidamente en cuenta las amenazas, los casos de uso malicioso, las técnicas de codificación segura, la seguridad de la capa de transporte (TLS) y las pruebas de seguridad, sobre todo: – los PUT y POST: qué modificaciones de los datos internos podrían utilizarse para atacar o transmitir información errónea; – los DELETE: podrían utilizarse para eliminar el contenido de un almacén de recursos internos; – los métodos permitidos de la lista blanca: para asegurar que los métodos HTTP permitidos sean debidamente restringidos y que otros devuelvan un código de respuesta adecuado; y – los ataques más conocidos deberían considerarse durante la fase de modelado de amenazas del proceso de diseño para asegurar que el riesgo de amenaza no aumente. DEBEN tenerse en cuenta las amenazas y medidas de mitigación definidas en la lista de los diez riesgos más críticos de OWASP.	AJ, AX, AAJ, AAX
[RSG-122]	En el desarrollo de las API DEBERÍAN seguirse las normas y las mejores prácticas que se enumeran a continuación: – las mejores prácticas de codificación segura: *OWASP Secure Coding Principles* (Principios de Codificación Segura de OWASP);	AJ, AX, AAX, AAJ

	– la seguridad de las API Rest: *REST Security Cheat Sheet* (Lista de Recomendaciones Prácticas sobre Seguridad REST); – las entradas de escape y la protección de secuencias de comandos en sitios cruzados: *OWASP XSS Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Secuencias de Comandos en Sitios Cruzados); – la prevención de ataques de inyección SQL: *OWASP SQL Injection Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Inyección SQL), *OWASP Parameterization Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Parametrización); y – la seguridad de la capa de transporte: *OWASP Transport Layer Protection Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Seguridad de la Capa de Transporte).	
[RSG-123]	DEBEN realizarse pruebas de seguridad y evaluaciones de vulnerabilidad para garantizar que las API son seguras y resistentes a las amenazas. Para ello PUEDEN aprovecharse las pruebas de seguridad de aplicaciones estáticas y dinámicas (SAST/DAST), las herramientas de gestión automatizada de vulnerabilidades y las pruebas de penetración.	AJ, AX, AAX, AAJ
[RSG-124]	Los servicios protegidos DEBEN proporcionar puntos finales HTTP solo mediante TLS 1.2, o una versión superior, con un paquete de cifrado que incluya el protocolo Elliptic-curve Diffie–Hellman (ECDH) para el intercambio de claves.	AJ, AX, AAJ, AAX
[RSG-130]	La autenticación anónima DEBE utilizarse solo cuando los clientes y la aplicación que están utilizando acceden a información o funciones con un nivel de sensibilidad bajo que no deberían requerir autenticación, como la información pública.	AJ, AX, AAJ, AAX
[RSG-131]	NO DEBE permitirse la autenticación con nombre de usuario y contraseña o con contraseña *hash* [generada mediante una función *hash* o función resumen].	AJ, AX, AAJ, AAX
[RSG-141]	Las claves de API DEBEN ser revocadas si el cliente incumple el contrato de uso, conforme a lo especificado por la oficina de propiedad intelectual.	AJ, AX, AAJ, AAX
[RSG-144]	Los certificados seguros y de confianza DEBEN ser emitidos por una autoridad certificadora de confianza mutua mediante un proceso de establecimiento de confianza o una certificación cruzada.	AJ, AX, AAJ, AAX
[RSG-145]	DEBERÍAN utilizarse certificados compartidos entre el cliente y el servidor, por ejemplo X.509, para mitigar los riesgos de seguridad de la identidad propios de los sistemas sensibles y las acciones privilegiadas.	AJ, AX, AAJ, AAX
[RSG-148]	Si la API REST es pública, el valor del encabezado HTTP `Access-Control-Allow-Origin` DEBE ser '*'.	AJ, AX, AAJ, AAX

Cuadro 2. Nivel de conformidad para respuesta en formato XML

ID de la norma	Descripción de la norma	Referencias cruzadas y observaciones
[RSG-01]	DEBE usarse la barra, "/", en la ruta del URI para indicar la relación jerárquica entre los recursos, pero la ruta NO DEBE terminar con una barra, ya que esta no proporciona ningún valor semántico y puede llevar a confusión.	AJ, AX, AAJ, AAX
[RSG-02]	Los nombres de los recursos DEBEN seguir un patrón de nomenclatura coherente.	AJ, AX, AAJ, AAX
[RSG-04]	Los parámetros de consulta DEBEN seguir un patrón de nomenclatura coherente.	AJ, AX
[RSG-06]	El patrón del URL para una API web DEBE contener la palabra 'api' en el URI.	AJ, AX, AAJ, AAX
[RSG-07]	NO DEBEN utilizarse parámetros matriciales.	AJ, AX, AAJ, AAX
[RSG-08]	Una API web DEBE aplicar sistemáticamente códigos de estado HTTP según se describe en los documentos RFC del IETF.	AJ, AX, AAJ, AAX
[RSG-10]	Si la API detecta valores de entrada incorrectos, DEBE devolver el código de estado HTTP `400 Bad Request`. La carga útil de error DEBE indicar el valor incorrecto.	AJ, AX, AAJ, AAX

[RSG-12]	Si la API detecta valores válidos que requieren funcionalidades no implementadas, DEBE devolver el código de estado HTTP `501 Not Implemented`. La carga útil de error DEBE indicar el valor sin implementar.	AJ, AX, AAJ, AAX
[RSG-14]	Si un recurso puede ser independiente, DEBE ser un recurso de nivel superior; de lo contrario será un subrecurso.	AJ, AX, AAJ, AAX
[RSG-15]	Para obtener recursos anidados, DEBEN utilizarse parámetros de consulta en lugar de rutas URL.	AJ, AX, AAJ, AAX
[RSG-18]	Los nombres de los recursos, los segmentos y los parámetros de consulta DEBEN estar compuestos por palabras en inglés, conforme a la grafía del inglés básica del Diccionario de Oxford (Oxford English Dictionary). PUEDEN estar en otros idiomas los nombres de recursos que deben traducirse por razones comerciales.	AJ, AX, AAJ, AAX
[RSG-20]	Una API web DEBE ser compatible con la negociación del tipo de contenido conforme a la Norma RFC 7231 del IETF.	AJ, AX, AAJ, AAX
[RSG-21]	DEBE utilizarse el formato JSON cuando no se solicita un tipo de contenido específico.	AJ, AX, AAJ, AAX
[RSG-27]	Una API web DEBE ser compatible como mínimo con el formato XML o JSON.	AJ, AX, AAJ, AAX
[RSG-28]	Los métodos HTTP DEBEN limitarse a los métodos estándares `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` y `HEAD`, conforme a las especificaciones de las normas RFC 7231 y RFC 5789 del IETF.	AJ, AX, AAJ, AAX
[RSG-33]	Para un punto final que extrae un solo recurso, si este no se encuentra, el método `GET` DEBE devolver el código de estado `404 Not Found`. En el caso de los puntos finales que devuelven listas de recursos, simplemente devolverán una lista vacía.	AJ, AX, AAJ, AAX
[RSG-34]	Si un recurso es recuperado con éxito, el método `GET` DEBE devolver `200 OK`.	AJ, AX, AAJ, AAX
[RSG-35]	Las peticiones `GET` DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-37]	Las peticiones `HEAD` DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-39]	Las peticiones `POST` NO DEBEN ser idempotentes de acuerdo con la Norma RFC 2616 del IETF.	AJ, AX, AAJ, AAX
[RSG-43]	Las peticiones `PUT` DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-44]	Si no se encuentra un recurso, `PUT` DEBE devolver el código de estado `404 Not Found`.	AJ, AX, AAJ, AAX
[RSG-45]	Si un recurso se actualiza correctamente, `PUT` DEBE devolver el código de estado `200 OK` si se devuelve el recurso actualizado o `204 No Content` si no se devuelve.	AJ, AX, AAJ, AAX
[RSG-46]	Las peticiones `PATCH` NO DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-48]	Si un recurso no se encuentra, `PATCH` DEBE devolver el código de estado `404 Not Found`.	AJ, AX, AAJ, AAX
[RSG-50]	Las peticiones `DELETE` NO DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-51]	Si no se encuentra un recurso, `DELETE` DEBE devolver el código de estado `404 Not Found`.	AJ, AX, AAJ, AAX
[RSG-52]	Si un recurso se elimina correctamente, `DELETE` DEBE devolver el código de estado `200 OK` si se devuelve el recurso eliminado o `204 No Content` si no se devuelve.	AJ, AX, AAJ, AAX
[RSG-53]	El destinatario final es el servidor de origen o el primer *proxy* o puerta de enlace que recibe un valor del encabezado `Max-Forwards` de cero en la petición. Las peticiones `TRACE` NO DEBEN incluir un cuerpo.	AJ, AX, AAJ, AAX
[RSG-54]	Las peticiones `TRACE` NO DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-55]	El valor del encabezado HTTP `Via` DEBE servir para rastrear la cadena de petición.	AJ, AX, AAJ, AAX
[RSG-56]	Para que el cliente pueda limitar la longitud de la cadena de petición DEBE utilizarse el encabezado HTTP `Max-Forwards`.	AJ, AX, AAJ, AAX

[RSG-58]	Las respuestas a `TRACE` NO DEBEN ser almacenadas en caché.	AJ, AX, AAJ, AAX
[RSG-60]	Las peticiones `OPTIONS` DEBEN ser idempotentes.	AJ, AX, AAJ, AAX
[RSG-70]	Las API web DEBEN utilizar parámetros de consulta para implementar la paginación.	AJ, AX, AAJ, AAX
[RSG-71]	Las API web NO DEBEN utilizar encabezados HTTP para implementar la paginación.	AJ, AX, AAJ, AAX
[RSG-75]	Para establecer criterios multiatributos de ordenación, DEBE utilizarse un parámetro de consulta. El valor de dicho parámetro será una lista de claves de ordenación separadas por comas a las que se puede añadir, mediante dos puntos ':', la dirección de ordenamiento, con `asc` para el orden ascendente o `desc` para el orden descendente. En caso de que no se especifique una dirección de ordenación para una clave, el servidor DEBE establecer una por defecto.	AJ, AX, AAJ, AAX
[RSG-76]	Las API web DEBERÍAN devolver los criterios de ordenación en la respuesta.	AJ, AX, AAJ, AAX
[RSG-79]	Las API web DEBEN admitir la devolución del número de elementos de una colección.	AJ, AX, AAJ, AAX
[RSG-80]	Para permitir la devolución del número de elementos de una colección DEBE utilizarse un parámetro de consulta.	AJ, AX, AAJ, AAX
[RSG-82]	Las API web PUEDEN admitir la devolución del número de elementos de una colección mediante una función *inline*, es decir, como parte de la respuesta que contiene la propia colección. DEBE utilizarse un parámetro de consulta.	AJ, AX, AAJ, AAX
[RSG-86]	Un contrato de servicio DEBE especificar la gramática compatible (como los campos, las funciones, las palabras clave y los operadores).	AJ, AX, AAJ, AAX
[RSG-87]	DEBE utilizarse el parámetro de consulta '`q`'.	AJ, AX, AAJ, AAX
[RSG-88]	En el nivel de protocolo, una API web DEBE devolver un código de estado HTTP apropiado seleccionado de la lista de códigos de estado HTTP estándares.	AJ, AX, AAJ, AAX
[RSJ-89]	En el nivel de aplicación, una API web DEBE devolver una carga útil que informe del error con suficiente granularidad. Los atributos `code` y `message` son obligatorios, el atributo `details` es condicionalmente obligatorio y los atributos `target`, `status`, `moreInfo`, e `internalMessage` son opcionales.	AJ, AX, AAJ, AAX
[RSG-90]	Los errores NO DEBEN exponer datos esenciales para la seguridad o detalles técnicos internos, como las pilas de llamada en los mensajes de error.	AJ, AX, AAJ, AAX
[RSG-91]	El encabezado HTTP `Reason-Phrase` (descrito en la Norma RFC 2616) NO DEBE utilizarse para transmitir mensajes de error.	AJ, AX, AAJ, AAX
[RSG-93]	El modelo de contrato de servicio DEBE especificar lo siguiente: – la versión de la API; – información sobre la semántica de los elementos de la API; – los recursos; – los atributos de los recursos; – los parámetros de consulta; – los métodos; – los tipos de medios; – la gramática de búsqueda (si se admite alguna); – los códigos de estado HTTP; – los métodos HTTP; – las restricciones y las características distintivas; y – la seguridad (por ejemplo, los esquemas privados).	AJ, AX, AAJ, AAX
[RSG-95]	Una API REST DEBE proporcionar la documentación de la API como un contrato de servicio.	AJ, AX, AAJ, AAX
[RSG-96]	Toda implementación de una API web que no cumpla con la presente norma DEBE ser documentada explícitamente en el contrato de servicio.	AJ, AX, AAJ, AAX

	Si no se especifica ninguna desviación de una de sus normas en el contrato de servicio, DEBE asumirse que se sigue la presente norma.	
[RSG-97]	Un contrato de servicio DEBE permitir la generación de código esquemático de cliente API.	AJ, AX, AAJ, AAX
[RSG-105]	Las API web DEBEN admitir el almacenamiento en caché de los resultados de `GET`; y PUEDEN admitir el almacenamiento en caché de los resultados de otros métodos HTTP.	AJ, AX, AAJ
[RSG-113]	Si una API web es compatible con la gestión de preferencias, en el contrato de servicio DEBE indicarse la nomenclatura de las preferencias que PUEDEN establecerse mediante el encabezado `Prefer`.	AAJ, AAX, AJ, AX
[RSG-114]	Si una API web permite la traducción de datos, DEBE admitir el encabezado de petición HTTP `Accept-Language` con el que se indica el conjunto de lenguas naturales preferidas para la respuesta, conforme a lo especificado en la Norma RFC 7231 del IETF.	AAJ, AAX, AJ, AX
[RSG-116]	Confidencialidad: las API y su información DEBEN ser identificadas, clasificadas y protegidas en todo momento frente a acciones no autorizadas de acceso, difusión e interceptación. DEBEN seguirse los principios de mínimos privilegios, cero confianza, necesidad de conocer y necesidad de compartir.	AAJ, AAX, AJ, AX
[RSG-117]	Garantía de integridad: las API y su información DEBEN estar protegidas contra acciones no autorizadas de modificación, duplicación, corrupción y destrucción. La modificación de la información DEBE realizarse mediante transacciones e interfaces aprobadas. La actualización de los sistemas DEBE llevarse a cabo mediante procesos aprobados de gestión de la configuración, gestión de cambios y gestión de parches.	AAJ, AAX, AJ, AX
[RSG-118]	Disponibilidad: las API y su información DEBEN estar disponibles para los usuarios autorizados en el debido momento, conforme se especifica en los acuerdos de nivel de servicio, las políticas de control de acceso y los procesos operativos definidos.	AAJ, AAX, AJ, AX
[RSG-119]	No repudio: toda transacción procesada o acción realizada por las API DEBE garantizar el no repudio mediante la auditoría, autorización y autenticación adecuadas, rutas seguras y servicios y mecanismos de no repudio.	AAJ, AAX, AJ, AX
[RSG-120]	Autenticación, autorización, auditoría: los usuarios, los sistemas, las API o los dispositivos que participen en transacciones o acciones importantes DEBEN ser autenticados, autorizados mediante servicios de control de acceso basados en roles o atributos y mantener la separación de tareas. Además, todas las acciones DEBEN ser registradas y el nivel de autenticación debe aumentar en función del riesgo asociado a la información.	AAJ, AAX, AJ, AX
[RSG-121]	En el desarrollo de las API DEBEN tenerse debidamente en cuenta las amenazas, los casos de uso malicioso, las técnicas de codificación segura, la seguridad de la capa de transporte (TLS) y las pruebas de seguridad, sobre todo: – los PUT y POST: qué modificaciones de los datos internos podrían utilizarse para atacar o transmitir información errónea; – los DELETE: podrían utilizarse para eliminar el contenido de un almacén de recursos internos; – los métodos permitidos de la lista blanca: para asegurar que los métodos HTTP permitidos sean debidamente restringidos y que otros devuelvan un código de respuesta adecuado; y – los ataques más conocidos deberían considerarse durante la fase de modelado de amenazas del proceso de diseño para asegurar que no aumente el riesgo de amenaza. DEBEN tenerse en cuenta las amenazas y medidas de mitigación definidas en la lista de los diez riesgos más críticos de OWASP.	AAJ, AAX, AJ, AX
[RSG-122]	En el desarrollo de las API DEBERÍAN seguirse las normas y las mejores prácticas que se enumeran a continuación:	AJ, AX, AAX, AAJ

	– las mejores prácticas de codificación segura: *OWASP Secure Coding Principles* (Principios de Codificación Segura de OWASP); – la seguridad de las API Rest: *REST Security Cheat Sheet* (Lista de Recomendaciones Prácticas sobre Seguridad REST); – las entradas de escape y la protección de secuencias de comandos en sitios cruzados: *OWASP XSS Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Secuencias de Comandos en Sitios Cruzados); – la prevención de ataques de inyección SQL: *OWASP SQL Injection Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Inyección SQL), *OWASP Parameterization Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Parametrización); y – la seguridad de la capa de transporte: *OWASP Transport Layer Protection Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Seguridad de la Capa de Transporte).	
[RSG-123]	DEBEN realizarse pruebas de seguridad y evaluaciones de vulnerabilidad para garantizar que las API son seguras y resistentes a las amenazas. Para ello PUEDEN aprovecharse las pruebas de seguridad de aplicaciones estáticas y dinámicas (SAST/DAST), las herramientas de gestión automatizada de vulnerabilidades y las pruebas de penetración.	AJ, AX, AAJ, AAX
[RSG-124]	Los servicios protegidos DEBEN proporcionar puntos finales HTTP solo mediante TLS 1.2, o una versión superior, con un paquete de cifrado que incluya el protocolo Elliptic-curve Diffie–Hellman (ECDH) para el intercambio de claves.	AJ, AX, AAJ, AAX
[RSG-130]	La autenticación anónima DEBE utilizarse solo cuando los clientes y la aplicación que están utilizando acceden a información o funciones con un nivel de sensibilidad bajo que no deberían requerir autenticación, como la información pública.	AJ, AX, AAJ, AAX
[RSG-131]	NO DEBE permitirse la autenticación con nombre de usuario y contraseña o con contraseña *hash* [generada mediante una función *hash* o función resumen].	AJ, AX, AAJ, AAX
[RSG-141]	Las claves de API DEBEN ser revocadas si el cliente incumple el contrato de uso, conforme a lo especificado por la oficina de propiedad intelectual.	AJ, AX, AAJ, AAX
[RSG-144]	Los certificados seguros y de confianza DEBEN ser emitidos por una autoridad certificadora de confianza mutua mediante un proceso de establecimiento de confianza o una certificación cruzada.	AJ, AX, AAJ, AAX
[RSG-145]	DEBERÍAN utilizarse certificados compartidos entre el cliente y el servidor, por ejemplo X.509, para mitigar los riesgos de seguridad de la identidad propios de los sistemas sensibles y las acciones privilegiadas.	AJ, AX, AAJ, AAX
[RSG-148]	Si la API REST es pública, el valor del encabezado HTTP `Access-Control-Allow-Origin` DEBE ser '*'.	AJ, AX, AAJ, AAX

Cuadro 3. Nivel de conformidad AAJ

ID de la norma	Norma	Referencias cruzadas y observaciones
[RSG-01]	DEBE usarse la barra, "/", en la ruta del URI para indicar la relación jerárquica entre los recursos, pero la ruta NO DEBE terminar con una barra, ya que esta no proporciona ningún valor semántico y puede llevar a confusión.	AAJ, AAX, AX, AJ
[RSG-02]	Los nombres de los recursos DEBEN seguir un patrón de nomenclatura coherente.	AAJ, AAX, AX, AJ
[RSG-03]	DEBERÍAN utilizarse las convenciones *lower case* o *kebab case* para los nombres de los recursos, que PUEDEN ser abreviados.	AAJ, AAX
[RSG-05]	DEBERÍA utilizarse la convención *lower camel case* para los parámetros de consulta, que PUEDEN ser abreviados.	AAJ, AAX
[RSG-06]	El patrón del URL para una API web DEBE contener la palabra 'api' en el URI.	AAJ, AAX, AX, AJ
[RSG-07]	NO DEBEN utilizarse parámetros matriciales.	AAJ, AAX, AX, AJ
[RSG-08]	Una API web DEBE aplicar sistemáticamente códigos de estado HTTP según se describe en los documentos RFC del IETF	AAJ, AAX, AX, AJ

[RSG-09]	Las API web DEBERÍAN utilizar los códigos recomendados en el Anexo V para clasificar los errores.	AAX, AAJ
[RSG-10]	Si la API detecta valores de entrada incorrectos, DEBE devolver el código de estado HTTP `400 Bad Request`. La carga útil de error DEBE indicar el valor incorrecto.	AAJ, AAX, AX, AJ
[RSG-11]	Si la API detecta nombres sintácticamente correctos de argumentos no esperados (en la petición o en los parámetros de consulta), DEBERÍA ignorarlos.	AAJ, AAX
[RSG-12]	Si la API detecta valores válidos que requieren funcionalidades no implementadas, DEBE devolver el código de estado HTTP `501 Not Implemented`. La carga útil de error DEBE indicar el valor sin implementar.	AAJ, AAX, AX, AJ
[RSG-13]	Una API web DEBERÍA utilizar únicamente recursos de nivel superior. Si hay subrecursos, deben ser colecciones e implicar una asociación. Una entidad debería ser accesible como recurso de nivel superior o como subrecurso, pero no por ambas vías.	AAJ, AAX
[RSG-14]	Si un recurso puede ser independiente, DEBE ser un recurso de nivel superior; de lo contrario será un subrecurso.	AAJ, AAX, AX, AJ
[RSG-15]	Para obtener recursos anidados, DEBEN utilizarse parámetros de consulta en lugar de rutas URL.	AAJ, AAX, AX, AJ
[RSG-16]	Los recursos DEBERÍAN nombrarse con sustantivos para las API web CRUD y con verbos para las API web Intent.	AAJ, AAX
[RSG-17]	Si el nombre del recurso es un sustantivo, DEBERÍA usarse siempre la forma plural. NO DEBERÍAN utilizarse formas sustantivas irregulares. Por ejemplo, en lugar de `/gente` debería usarse `/personas`.	AAJ, AAX
[RSG-18]	Los nombres de los recursos, los segmentos y los parámetros de consulta DEBEN estar compuestos por palabras en inglés, conforme a la grafía del inglés básica del Diccionario de Oxford (Oxford English Dictionary). PUEDEN estar en otros idiomas los nombres de recursos que deben traducirse por razones comerciales.	AAJ, AAX, AX, AJ
[RSG-19]	Una API web DEBERÍA utilizar para la negociación del tipo de contenido el encabezado de petición HTTP `Accept` y el encabezado de respuesta HTTP `Content-Type`.	AAJ, AAX
[RSG-20]	Una API web DEBE ser compatible con la negociación del tipo de contenido conforme a la Norma RFC 7231 del IETF.	AAJ, AAX, AX, AJ
[RSG-21]	DEBE utilizarse el formato JSON cuando no se solicita un tipo de contenido específico.	AAJ, AAX, AX, AJ
[RSG-22]	Una API web DEBERÍA devolver el código de estado `406 Not Acceptable` si no es compatible con el formato solicitado.	AAJ, AAX
[RSG-23]	Una API web DEBERÍA rechazar las peticiones que contengan encabezados de tipo de contenido inesperado o no aceptable y devolver el código de estado `HTTP 406 Not Acceptable` o `415 Unsupported Media Type`.	AAJ, AAX
[RSG-24]	Las peticiones y respuestas (convención de nombres, formato del mensaje, estructura de datos y diccionario de datos) DEBERÍAN ajustarse a la Norma ST.96 para XML o la Norma ST.97 para JSON.	AAX, AAJ
[RSJ-25]	Los nombres de las propiedades de los objetos JSON DEBERÍAN proporcionarse en la convención lower camel case (ejemplo: `applicantName`).	AAJ
[RSG-27]	Una API web DEBE ser compatible como mínimo con el formato XML o JSON.	AAJ, AAX, AX, AJ
[RSG-28]	Los métodos HTTP DEBEN limitarse a los métodos estándares `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` y `HEAD`, conforme a las especificaciones de las normas RFC 7231 y RFC 5789 del IETF.	AAJ, AAX, AX, AJ
[RSG-29]	Los métodos HTTP PUEDEN seguir el principio de selección cuidadosa, que establece que solo se debe implementar la funcionalidad necesaria para el escenario de uso previsto.	AAJ, AAX
[RSG-30]	Algunos *proxies* solo son compatibles con los métodos `POST` y `GET`. Para sortear esa limitación, las API web pueden utilizar un método `POST` con un encabezado HTTP personalizado para crear un túnel sobre el método HTTP que se quiera usar. DEBERÍA utilizarse el encabezado HTTP personalizado `X-HTTP-Method`.	AAJ, AAX

[RSG-31]	Si un método HTTP no es admitido, DEBERÍA devolverse el código de estado HTTP `405 Method Not Allowed`.	AAJ, AAX
[RSG-32]	Una API web DEBERÍA admitir operaciones por lotes (también conocidas como operaciones masivas) en lugar de múltiples peticiones individuales con el fin de reducir la latencia. Se debería utilizar la misma semántica para los métodos HTTP y los códigos de estado HTTP. La carga útil de respuesta DEBERÍA contener información sobre todas las operaciones por lotes. Si se producen múltiples errores, la carga útil de error DEBERÍA contener información sobre todas las incidencias (en el atributo `details`). Todas las operaciones masivas DEBERÍAN ser ejecutadas mediante una instrucción atómica.	AAJ, AAX
[RSG-33]	Para un punto final que extrae un solo recurso, si este no se encuentra, el método GET DEBE devolver el código de estado `404 Not Found`. En el caso de los puntos finales que devuelven listas de recursos, simplemente devolverán una lista vacía.	AAJ, AAX, AX, AJ
[RSG-34]	Si un recurso es recuperado con éxito, el método `GET` DEBE devolver `200 OK`.	AAJ, AAX, AX, AJ
[RSG-35]	Las peticiones `GET` DEBEN ser idempotentes.	AAJ, AAX, AX, AJ
[RSG-36]	Cuando la longitud del URI excede los 255 bytes, DEBERÍA utilizarse el método `POST` en lugar del método `GET` debido a las limitaciones de este, o bien crear consultas con nombre si es posible.	AAJ, AAX
[RSG-37]	Las peticiones `HEAD` DEBEN ser idempotentes.	AAJ, AAX, AX, AJ
[RSG-38]	Algunos *proxies* solo son compatibles con los métodos `POST` y `GET`. Una API web DEBERÍA admitir un encabezado de petición HTTP personalizado para sustituir el método HTTP con el fin de sortear esa limitación.	AAJ, AAX
[RSG-39]	Las peticiones `POST` NO DEBEN ser idempotentes de acuerdo con la Norma RFC 2616 del IETF.	AAJ, AAX, AX, AJ
[RSG-40]	Si el recurso se creó correctamente, el encabezado HTTP `Location` DEBERÍA contener un URI (absoluto o relativo) que apunte al recurso creado.	AAJ, AAX
[RSG-41]	Si el recurso se creó correctamente, la respuesta DEBERÍA contener el código de estado `201 Created`.	AAJ, AAX
[RSG-42]	Si el recurso se creó correctamente, la carga útil de respuesta DEBERÍA contener por defecto el cuerpo del recurso creado, para que el cliente pueda utilizarlo sin hacer otra llamada HTTP.	AAJ, AAX
[RSG-43]	Las peticiones `PUT` DEBEN ser idempotentes.	AAJ, AAX, AX, AJ
[RSG-44]	Si no se encuentra un recurso, `PUT` DEBE devolver el código de estado `404 Not Found`.	AAJ, AAX, AX, AJ
[RSG-45]	Si un recurso se actualiza correctamente, `PUT` DEBE devolver el código de estado `200 OK` si se devuelve el recurso actualizado o `204 No Content` si no se devuelve.	AAJ, AAX, AX, AJ
[RSG-46]	Las peticiones `PATCH` NO DEBEN ser idempotentes.	AAJ, AAX, AX, AJ
[RSG-47]	Si una API web implementa actualizaciones parciales, `PATCH` DEBERÍA ser idempotente. Para conseguirlo, la API PUEDE recurrir a la técnica del bloqueo optimista recomendada en la Norma RFC 5789 del IETF.	AAJ, AAX
[RSG-48]	Si un recurso no se encuentra, `PATCH` DEBE devolver el código de estado `404 Not Found`.	AAJ, AAX, AX, AJ
[RSJ-49]	Si una API web implementa actualizaciones parciales mediante `PATCH`, DEBE utilizar el formato `JSON Merge Patch` para describir el conjunto de cambios parciales, como se especifica en la Norma RFC 7386 del IETF (utilizando `application/merge-patch+json` como `Content-Type`).	AAJ, AJ
[RSG-50]	Las peticiones `DELETE` NO DEBEN ser idempotentes.	AAJ, AAX, AX, AJ
[RSG-51]	Si no se encuentra un recurso, `DELETE` DEBE devolver el código de estado `404 Not Found`.	AAJ, AAX, AX, AJ
[RSG-52]	Si un recurso se elimina correctamente, `DELETE` DEBE devolver el código de estado `200 OK` si se devuelve el recurso eliminado o `204 No Content` si no se devuelve.	AAJ, AAX, AX, AJ
[RSG-53]	El destinatario final es el servidor de origen o el primer *proxy* o puerta de enlace que recibe un valor del encabezado `Max-Forwards` de cero en la petición. Las peticiones `TRACE` NO DEBEN incluir un cuerpo.	AAJ, AAX, AX, AJ
[RSG-54]	Las peticiones `TRACE` NO DEBEN ser idempotentes.	AAJ, AAX, AX, AJ

[RSG-55]	El valor del encabezado HTTP `Via` DEBE servir para rastrear la cadena de petición.	AAJ, AAX, AX, AJ
[RSG-56]	Para que el cliente pueda limitar la longitud de la cadena de petición DEBE utilizarse el encabezado HTTP `Max-Forwards`.	AAJ, AAX, AX, AJ
[RSG-57]	Si la petición es válida, la respuesta DEBERÍA contener el mensaje completo de la petición en el cuerpo de la respuesta, con `message/http` como `Content-Type`.	AAJ, AAX
[RSG-58]	Las respuestas a `TRACE` NO DEBEN ser almacenadas en caché.	AAJ, AAX, AX, AJ
[RSG-59]	DEBERÍA devolverse a `TRACE` el código de estado `200 OK`.	AAJ, AAX
[RSG-60]	Las peticiones `OPTIONS` DEBEN ser idempotentes.	AAJ, AAX, AX, AJ
[RSG-61]	NO DEBERÍAN utilizarse encabezados HTTP personalizados que empiecen con el prefijo "`X-`".	AAJ, AAX
[RSG-62]	NO DEBERÍAN utilizarse encabezados HTTP personalizados para modificar un método HTTP a menos que sea para resolver una limitación técnica (véase, por ejemplo, [RSG-39]).	AAJ, AAX
[RSG-63]	Los encabezados HTTP personalizados tendrán el formato `<organization>-<header name>`, y los nombres de `<organization>` y `<header>` DEBERÍAN seguir la convención *kebab case*.	AAJ, AAX
[RSG-64]	Una API web DEBERÍA admitir un único método de versionado de servicios, como el versionado por URI (por ejemplo, `/api/v1/inventors`), el versionado por encabezado (por ejemplo, `Accept-version: v1`), o el versionado por tipo de medios (por ejemplo, `Accept: application/vnd.v1+json`). NO DEBERÍA utilizarse el versionado por cadena de consulta.	AAJ, AAX
[RSG-65]	DEBERÍA seguirse un esquema de numeración de versiones que considere solo el número de versión mayor (por ejemplo, `/v1`).	AAJ, AAX
[RSG-66]	Los contratos de servicio API PUEDEN incluir la función de redireccionamiento del punto final. Cuando un consumidor de servicios intenta invocar un servicio, es posible que se devuelva una respuesta de redireccionamiento para indicarle que vuelva a enviar la solicitud a un nuevo punto final. Los redireccionamientos PUEDEN ser temporales o permanentes: – redireccionamiento temporal - mediante el encabezado de respuesta HTTP `Location` y el código de estado HTTP `302 Found`, conforme a la Norma RFC 7231 del IETF; o – redireccionamiento permanente - mediante el encabezado de respuesta HTTP `Location` y el código de estado HTTP `301 Moved Permanently`, de conformidad con el documento RFC 7238 del IETF.	AAJ, AAX
[RSG-67]	Los desarrolladores de API DEBERÍAN publicar las estrategias de ciclo de vida de las API con el fin de que los usuarios sepan cuánto tiempo se mantendrá una versión.	AAJ, AAX
[RSG-68]	Las API web DEBERÍAN ser compatibles con la paginación.	AAJ, AAX
[RSG-69]	Las peticiones paginadas PUEDEN NO ser idempotentes.	AAJ, AAX
[RSG-70]	Las API web DEBEN utilizar parámetros de consulta para implementar la paginación.	AAJ, AAX, AX, AJ
[RSG-71]	Las API web NO DEBEN utilizar encabezados HTTP para implementar la paginación.	AAJ, AAX, AX, AJ
[RSG-72]	DEBERÍAN utilizarse los parámetros de consulta `limit=<number of items to deliver>` y `offset=<number of items to skip>`; `limit` es el número de elementos que se devolverán (tamaño de la página), y `offset` es el número de elementos que se saltarán (desplazamiento). Si no se especifica un límite de tamaño de página, DEBERÍA definirse un valor por defecto, general o por colección; el offset por defecto DEBE ser cero “0”. A continuación se muestra un ejemplo de URL válido: https://wipo.int/api/v1/patents?limit=10&offset=20	AAJ, AAX
[RSG-73]	Los valores de los parámetros `limit` y `offset` DEBERÍAN incluirse en la respuesta.	AAJ, AAX
[RSG-74]	Una API web DEBERÍA admitir la ordenación.	AAJ, AAX

[RSG-75]	Para establecer criterios multiatributos de ordenación, DEBE utilizarse un parámetro de consulta. El valor de dicho parámetro será una lista de claves de ordenación separadas por comas a las que se puede añadir, mediante dos puntos ':', la dirección de ordenamiento, con `asc` para el orden ascendente o `desc` para el orden descendente. En caso de que no se especifique una dirección de ordenación para una clave, el servidor DEBE establecer una por defecto.	AAJ, AAX, AX, AJ
[RSG-76]	Las API web DEBERÍAN devolver los criterios de ordenación en la respuesta.	AAJ, AAX, AX, AJ
[RSG-77]	Una API web PUEDE ser compatible con la expansión del cuerpo del contenido que se devuelve. DEBERÍA utilizarse el parámetro de consulta `expand=<comma-separated list of attributes names>`.	AAJ, AAX
[RSG-78]	En caso de que una API web admita la proyección, DEBERÍA utilizarse, en lugar de las rutas URL, un parámetro de consulta con el formato `"fields="<comma-separated list of attribute names>`.	AAJ, AAX
[RSG-79]	Las API web DEBEN admitir la devolución del número de elementos de una colección.	AAJ, AAX, AX, AJ
[RSG-80]	Para permitir la devolución del número de elementos de una colección DEBE utilizarse un parámetro de consulta.	AAJ, AAX, AX, AJ
[RSG-81]	DEBERÍA utilizarse el parámetro de consulta `count` para devolver el número de elementos de una colección.	AAJ, AAX
[RSG-82]	Las API web PUEDEN admitir la devolución del número de elementos de una colección mediante una función *inline*, es decir, como parte de la respuesta que contiene la propia colección. DEBE utilizarse un parámetro de consulta.	AAJ, AAX, AX, AJ
[RSG-83]	DEBERÍA utilizarse el parámetro de consulta `count=true`. Si no se especifica el valor de `count`, debería fijarse por defecto en `false`.	AAJ, AAX
[RSG-84]	Si una API web admite la paginación, DEBERÍA permitir la devolución del número de la colección (es decir, el número total de elementos de la colección) en la respuesta mediante una función *inline*.	AAJ, AAX
[RSG-85]	Si una API web es compatible con expresiones de búsqueda complejas, DEBERÍA especificarse un lenguaje de consulta, como el CQL.	AAJ, AAX
[RSG-86]	Un contrato de servicio DEBE especificar la gramática compatible (como los campos, las funciones, las palabras clave y los operadores).	AAJ, AAX, AX, AJ
[RSG-87]	DEBE utilizarse el parámetro de consulta '`q`'.	AAJ, AAX, AX, AJ
[RSG-88]	En el nivel de protocolo, una API web DEBE devolver un código de estado HTTP apropiado seleccionado de la lista de códigos de estado HTTP estándares.	AAJ, AAX, AX, AJ
[RSJ-89]	En el nivel de aplicación, una API web DEBE devolver una carga útil que informe del error con suficiente granularidad. Los atributos `code` y `message` son obligatorios, el atributo `details` es condicionalmente obligatorio y los atributos `target`, `status`, `moreInfo`, e `internalMessage` son opcionales.	AAJ, AAX, AX, AJ
[RSG-90]	Los errores NO DEBEN exponer datos esenciales para la seguridad o detalles técnicos internos, como las pilas de llamada en los mensajes de error.	AAJ, AAX, AX, AJ
[RSG-91]	El encabezado HTTP `Reason-Phrase` (descrito en la Norma RFC 2616) NO DEBE utilizarse para transmitir mensajes de error.	AAJ, AAX, AX, AJ
[RSG-92]	Cada error registrado DEBERÍA tener un ID de correlación único. DEBERÍA utilizarse un encabezado HTTP personalizado con el nombre Correlation-ID.	AAJ, AAX
[RSG-93]	El modelo de contrato de servicio DEBE especificar lo siguiente: – la versión de la API; – información sobre la semántica de los elementos de la API; – los recursos; – los atributos de los recursos; – los parámetros de consulta; – los métodos; – los tipos de medios; – la gramática de búsqueda (si se admite alguna); – los códigos de estado HTTP; – los métodos HTTP;	AAJ, AAX, AX, AJ

	– las restricciones y las características distintivas; y – la seguridad (por ejemplo, los esquemas privados).	
[RSG-94]	El modelo de contrato de servicio DEBERÍA incluir peticiones y respuestas en el esquema XML o en el esquema JSON y ejemplos de uso de la API en los formatos compatibles, a saber, XML o JSON.	AAJ, AAX
[RSG-95]	Una API REST DEBE proporcionar la documentación de la API como un contrato de servicio.	AAJ, AAX, AX, AJ
[RSG-96]	Toda implementación de una API web que no cumpla con la presente norma DEBE ser documentada explícitamente en el contrato de servicio. Si no se especifica ninguna desviación de una de sus normas en el contrato de servicio, DEBE asumirse que se sigue la presente norma.	AAJ, AAX, AX, AJ
[RSG-97]	Un contrato de servicio DEBE permitir la generación de código esquemático de cliente API.	AAJ, AAX, AX, AJ
[RSG-98]	Un contrato de servicio DEBERÍA permitir la generación de código esquemático de servidor.	AAJ, AAX
[RSG-99]	La documentación de las API web DEBERÍA escribirse utilizando el RAML o la OAS. NO DEBERÍAN utilizarse formatos de documentación personalizados.	AAJ, AAX
[RSG-100]	Un consumidor de API web DEBERÍA poder especificar un tiempo de espera del servidor para cada petición; DEBERÍA utilizarse un encabezado HTTP personalizado. También DEBERÍA establecerse un tiempo máximo de espera del servidor para evitar un uso excesivo de los recursos del servidor.	AAJ, AAX
[RSG-101]	Una API web DEBERÍA admitir la recuperación condicional de datos, para asegurar que solo se recuperen los datos que se hayan modificado. DEBERÍA utilizarse la validación de recursos basada en el contenido por ser la más precisa.	AAJ, AAX
[RSG-102]	Para implementar la validación de recursos basada en el contenido, DEBERÍA utilizarse el encabezado HTTP `ETag` en la respuesta para codificar el estado de los datos. El valor de `Etag` DEBERÍA utilizarse en los encabezados condicionales HTTP (como `If-Match` o `If-None-Match`) de las peticiones subsiguientes. Si no se modificaron los datos desde que la petición devolvió el `ETag`, el servidor DEBERÍA devolver el código de estado `304 Not Modified`. Este mecanismo se especifica en las normas RFC 7231 y RFC 7232 del IETF.	AAJ, AAX
[RSG-103]	Para implementar la validación de recursos basada en el tiempo DEBERÍA utilizarse el encabezado HTTP `Last-Modified`. Este mecanismo se especifica en las normas RFC 7231 y RFC 7232 del IETF.	AAJ, AAX
[RSG-104]	Utilizando el versionado por respuesta, un consumidor de servicios PUEDE implementar el bloqueo optimista.	AAJ, AAX
[RSG-105]	Las API web DEBEN admitir el almacenamiento en caché de los resultados de `GET`; y PUEDEN admitir el almacenamiento en caché de los resultados de otros métodos HTTP.	AAJ, AJ, AX
[RSG-106]	DEBERÍAN utilizarse los encabezados de respuesta HTTP `Cache-Control` y `Expires`. `Expires` PUEDE utilizarse para prestar asistencia a los clientes heredados.	AAJ, AAX
[RSG-107]	Una API web DEBE indicar si permite descargas de archivos por partes, mediante respuestas a peticiones `HEAD` que incluyan los encabezados de respuesta HTTP `Accept-Ranges` y `Content-Length`.	AAJ, AAX
[RSG-108]	Una API web DEBERÍA permitir la descarga de archivos por partes. DEBERÍA admitirse la petición de múltiples rangos.	AAJ, AAX
[RSG-109]	Una API web DEBERÍA especificar si admite la subida de archivos por partes.	AAJ, AAX
[RSG-110]	Una API web DEBERÍA ser compatible con la subida de archivos por partes. DEBERÍA admitirse la petición de múltiples rangos.	AAJ, AAX
[RSG-111]	El proveedor de servicios DEBERÍA devolver con los encabezados de respuesta HTTP el código de estado HTTP `413 Request Entity Too Large` en caso de que la petición supere el límite máximo permitido. PUEDE utilizarse un encabezado HTTP personalizado para indicar el tamaño máximo de la petición.	AAJ, AAX
[RSG-112]	Si una API web admite la gestión de preferencias, esta DEBERÍA implementarse de acuerdo con la Norma RFC 7240 del IETF, es decir, DEBERÍA utilizarse el encabezado de petición HTTP `Prefer` y	AAJ, AAX

	devolverse el encabezado de respuesta HTTP `Preference-Applied` (que incluye la petición original).	
[RSG-113]	Si una API web es compatible con la gestión de preferencias, en el contrato de servicio DEBE indicarse la nomenclatura de las preferencias que PUEDEN establecerse mediante el encabezado `Prefer`.	AAJ, AAX, AJ, AX
[RSG-114]	Si una API web permite la traducción de datos, DEBE admitir el encabezado de petición HTTP `Accept-Language` con el que se indica el conjunto de lenguas naturales preferidas para la respuesta, conforme a lo especificado en la Norma RFC 7231 del IETF.	AAJ, AAX, AJ, AX
[RSG-115]	Si la API permite operaciones de larga duración, estas DEBERÍAN ejecutarse de forma asíncrona. DEBERÍA seguirse el enfoque descrito a continuación: a) el consumidor de servicios activa la operación de servicio; b) la operación de servicio devuelve el código de estado 202 Accepted conforme a la Norma RFC 7231 del IETF (sección 6.3.3), que indica que la petición fue aceptada para su procesamiento, pero este no se completó. La ubicación de la tarea pendiente que se creó también se devuelve con el encabezado HTTP Location; y c) el consumidor de servicios llama a la ubicación devuelta para saber si el recurso está disponible. Si el recurso no está disponible, la respuesta DEBERÍA incluir el código de estado 200 OK y el estado de la tarea (por ejemplo, pendiente) y PUEDE contener otra información (por ejemplo, un indicador de progreso y/o un enlace para cancelar o eliminar la tarea mediante el método HTTP DELETE). Si el recurso está disponible, la respuesta DEBERÍA incluir el código de estado 303 See Other, y el encabezado HTTP Location DEBERÍA contener el URL para recuperar los resultados de la tarea.	AAJ, AAX
[RSG-116]	Confidencialidad: las API y su información DEBEN ser identificadas, clasificadas y protegidas en todo momento frente a acciones no autorizadas de acceso, difusión e interceptación. DEBEN seguirse los principios de mínimos privilegios, cero confianza, necesidad de conocer y necesidad de compartir.	AAJ, AAX, AJ, AX
[RSG-117]	Garantía de integridad: las API y su información DEBEN estar protegidas contra acciones no autorizadas de modificación, duplicación, corrupción y destrucción. La modificación de la información DEBE realizarse mediante transacciones e interfaces aprobadas. La actualización de los sistemas DEBE llevarse a cabo mediante procesos aprobados de gestión de la configuración, gestión de cambios y gestión de parches.	AAJ, AAX, AJ, AX
[RSG-118]	Disponibilidad: las API y su información DEBEN estar disponibles para los usuarios autorizados en el debido momento, conforme se especifica en los acuerdos de nivel de servicio, las políticas de control de acceso y los procesos operativos definidos.	AAJ, AAX, AJ, AX
[RSG-119]	No repudio: toda transacción procesada o acción realizada por las API DEBE garantizar el no repudio mediante la auditoría, autorización y autenticación adecuadas, rutas seguras y servicios y mecanismos de no repudio.	AAJ, AAX, AJ, AX
[RSG-120]	Autenticación, autorización, auditoría: los usuarios, los sistemas, las API o los dispositivos que participen en transacciones o acciones importantes DEBEN ser autenticados, autorizados mediante servicios de control de acceso basados en roles o atributos y mantener la separación de tareas. Además, todas las acciones DEBEN ser registradas y el nivel de autenticación debe aumentar en función del riesgo asociado a la información.	AAJ, AAX, AJ, AX
[RSG-121]	En el desarrollo de las API DEBEN tenerse debidamente en cuenta las amenazas, los casos de uso malicioso, las técnicas de codificación segura, la seguridad de la capa de transporte (TLS) y las pruebas de seguridad, sobre todo:	AAJ, AAX, AJ, AX

	– los PUT y POST: qué modificaciones de los datos internos podrían utilizarse para atacar o transmitir información errónea; – los DELETE: podrían utilizarse para eliminar el contenido de un almacén de recursos internos; – los métodos permitidos de la lista blanca: para asegurar que los métodos HTTP permitidos sean debidamente restringidos y que otros devuelvan un código de respuesta adecuado; y – los ataques más conocidos deberían considerarse durante la fase de modelado de amenazas del proceso de diseño para asegurar que el riesgo de amenaza no aumente. DEBEN tenerse en cuenta las amenazas y medidas de mitigación definidas en la lista de los diez riesgos más críticos de OWASP.	
[RSG-122]	En el desarrollo de las API DEBERÍAN seguirse las normas y las mejores prácticas que se enumeran a continuación: – las mejores prácticas de codificación segura: *OWASP Secure Coding Principles* (Principios de Codificación Segura de OWASP); – la seguridad de las API Rest: *REST Security Cheat Sheet* (Lista de Recomendaciones Prácticas sobre Seguridad REST); – las entradas de escape y la protección de secuencias de comandos en sitios cruzados: *OWASP XSS Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Secuencias de Comandos en Sitios Cruzados); – la prevención de ataques de inyección SQL: *OWASP SQL Injection Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Inyección SQL), *OWASP Parameterization Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Parametrización); y – la seguridad de la capa de transporte: *OWASP Transport Layer Protection Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Seguridad de la Capa de Transporte).	AAJ, AAX, AJ, AX
[RSG-123]	DEBEN realizarse pruebas de seguridad y evaluaciones de vulnerabilidad para garantizar que las API son seguras y resistentes a las amenazas. Para ello PUEDEN aprovecharse las pruebas de seguridad de aplicaciones estáticas y dinámicas (SAST/DAST), las herramientas de gestión automatizada de vulnerabilidades y las pruebas de penetración.	AAJ, AAX, AJ, AX
[RSG-124]	Los servicios protegidos DEBEN proporcionar puntos finales HTTP solo mediante TLS 1.2, o una versión superior, con un paquete de cifrado que incluya el protocolo Elliptic-curve Diffie–Hellman (ECDH) para el intercambio de claves.	AAJ, AAX, AJ, AX
[RSG-125]	Cuando se consideran los protocolos de autenticación, DEBERÍA utilizarse una confidencialidad directa perfecta para garantizar la seguridad del transporte. NO DEBERÍAN permitirse los algoritmos criptográficos inseguros ni la compatibilidad hacia atrás con SSL 3 y TLS 1.0/1.1.	AAX, AAJ
[RSG-126]	Para una máxima seguridad y confianza, DEBERÍA establecerse una VPN Ipsec de sitio a sitio para proteger mejor la información transmitida a través de redes inseguras.	AAX, AAJ
[RSG-127]	La aplicación consumidora DEBERÍA validar la cadena de certificados TLS al realizar peticiones a recursos protegidos, incluida la comprobación de la lista de revocación de certificados.	AAX, AAJ
[RSG-128]	Los servicios protegidos DEBERÍAN utilizar solo certificados válidos emitidos por una autoridad certificadora de confianza.	AAX, AAJ
[RSG-129]	Los *tokens* DEBERÍAN ser firmados utilizando algoritmos de firma segura que cumplan con el estándar de firma digital (DSS) FIPS 186-4. DEBERÍAN considerarse los algoritmos de firma digital de Rivest, Shamir y Adleman (RSA) o de curva elíptica (ECDSA).	AAX, AAJ
[RSG-130]	La autenticación anónima DEBE utilizarse solo cuando los clientes y la aplicación que están utilizando acceden a información o funciones con	AAJ, AAX, AJ, AX

	un nivel de sensibilidad bajo que no deberían requerir autenticación, como la información pública.	
[RSG-131]	NO DEBE permitirse la autenticación con nombre de usuario y contraseña o con contraseña *hash* [generada mediante una función *hash* o función resumen].	AAJ, AAX, AJ, AX
[RSG-132]	Si un servicio está protegido, DEBERÍA usarse el protocolo OpenID Connect.	AAX, AAJ
[RSG-133]	Cuando se utiliza un JSON Web Token (JWT), el secreto JWT DEBERÍA tener una entropía alta para aumentar el factor de trabajo de los ataques de fuerza bruta; los *tokens* TTL y RTTL DEBERÍAN ser lo más cortos posibles; y la información sensible NO DEBERÍA almacenarse en la carga útil JWT.	AAX, AAJ
[RSG-134]	En las peticiones `POST y PUT`, los datos sensibles DEBERÍAN ser transferidos en el cuerpo de la petición o mediante los encabezados de la petición.	AAX, AAJ
[RSG-135]	En las peticiones `GET`, los datos sensibles DEBERÍAN ser transferidos en un encabezado HTTP.	AAX, AAJ
[RSG-136]	Para minimizar la latencia y reducir el acoplamiento entre los servicios protegidos, la decisión de control de acceso DEBERÍA ser tomada localmente por los puntos finales REST.	AAX, AAJ
[RSG-137]	DEBERÍAN utilizarse claves de API en los servicios protegidos y públicos para evitar que se sobrecargue a sus proveedores de servicios con múltiples peticiones (ataques de denegación de servicio). En el caso de los servicios protegidos, PUEDEN utilizarse claves de API para la monetización (planes adquiridos), la aplicación de políticas de calidad de servicio y la supervisión.	AAX, AAJ
[RSG-138]	Las claves de API PUEDEN combinarse con el encabezado de petición HTTP `user-agent` para determinar si el usuario es una persona o un agente de software, conforme a lo especificado en la Norma RFC 7231 del IETF.	AAX, AAJ
[RSG-139]	El proveedor de servicios DEBERÍA devolver junto con los encabezados de respuesta HTTP el estado de uso en el momento. Se PUEDEN devolver los siguientes datos de respuesta: – límite de tasa (*rate limit*): máximo número de peticiones por minuto permitido en el sistema; – límite de tasa restante (*rate limit remaining*): número de peticiones restantes (-1 indica que se ha superado el límite); y – restablecimiento del límite de tasa (*rate limit reset*): tiempo (en segundos) necesario para que se restablezca el número máximo de peticiones permitidas.	AAX, AAJ
[RSG-140]	El proveedor de servicios DEBERÍA devolver el código de estado `429 Too Many Requests` si se reciben demasiadas peticiones en poco tiempo.	AAX, AAJ
[RSG-141]	Las claves de API DEBEN ser revocadas si el cliente incumple el contrato de uso, conforme a lo especificado por la oficina de propiedad intelectual.	AAJ, AAX, AJ, AX
[RSG-142]	La transferencia de las claves de API DEBERÍA realizarse mediante encabezados HTTP personalizados. NO DEBERÍA realizarse mediante parámetros de consulta.	AAX, AAJ
[RSG-143]	Las claves de API DEBERÍAN generarse de forma aleatoria.	AAX, AAJ
[RSG-144]	Los certificados seguros y de confianza DEBEN ser emitidos por una autoridad certificadora de confianza mutua mediante un proceso de establecimiento de confianza o una certificación cruzada.	AAJ, AAX, AJ, AX
[RSG-145]	DEBERÍAN utilizarse certificados compartidos entre el cliente y el servidor, por ejemplo X.509, para mitigar los riesgos de seguridad de la identidad propios de los sistemas sensibles y las acciones privilegiadas.	AAJ, AAX, AJ, AX
[RSG-146]	Para servicios con un alto nivel de privilegio, la autenticación mutua bidireccional entre el cliente y el servidor DEBERÍA utilizar certificados para ofrecer una mayor protección.	AAX, AAJ
[RSG-147]	DEBERÍA implementarse la autenticación multifactor para mitigar los riesgos de la identidad en aplicaciones con un perfil de alto riesgo, en sistemas que procesen información muy sensible o en acciones privilegiadas.	AAX, AAJ

[RSG-148]	Si la API REST es pública, el valor del encabezado HTTP `Access-Control-Allow-Origin` DEBE ser '*'.	AAJ, AAX, AJ, AX
[RSG-149]	Si la API REST está protegida, DEBERÍA usarse el intercambio de recursos de origen cruzado (CORS), si es posible. De lo contrario, PUEDE utilizarse JSONP como alternativa, pero solo para peticiones `GET`, por ejemplo, cuando el usuario accede con un navegador antiguo. NO DEBERÍA utilizarse `iframe`.	AAX, AAJ
[RSJ-150]	Si se utilizan instancias descritas por un esquema, DEBERÍA utilizarse el encabezado `Link` para proporcionar un enlace a un esquema JSON descargable de acuerdo con la Norma RFC 8288.	AAJ
[RSJ-151]	Una API web DEBERÍA alcanzar al menos el nivel 2 (propiedades nativas de transporte) del modelo de madurez de Richardson. Se PUEDE implementar el nivel 3 (hipermedia) que permite descubrir la API en su totalidad.	AAJ
[RSJ-152]	Para diseñar un formato de hipermedia personalizado DEBERÍAN utilizarse los siguientes atributos en el enlace de atributos: – `href`: el URI de destino; – `rel`: el significado del URI de destino; – `self`: el URI hace referencia al propio recurso; – `next`: el URI hace referencia a la página siguiente (si se utiliza durante la paginación); – `previous`: el URI hace referencia a la página anterior (si se utiliza durante la paginación); y – el nombre arbitrario v indica el significado personalizado de una relación.	AAJ

Cuadro 4. Nivel de conformidad AAX

ID de la norma	Norma	Referencias cruzadas y observaciones
[RSG-01]	DEBE usarse la barra, "/", en la ruta del URI para indicar la relación jerárquica entre los recursos, pero la ruta NO DEBE terminar con una barra, ya que esta no proporciona ningún valor semántico y puede llevar a confusión.	AAJ, AAX
[RSG-02]	Los nombres de los recursos DEBEN seguir un patrón de nomenclatura coherente.	AAJ, AAX, AJ, AX
[RSG-03]	DEBERÍAN utilizarse las convenciones *lower case* o *kebab case* para los nombres de los recursos, que PUEDEN ser abreviados.	AAJ, AAX
[RSG-05]	DEBERÍA utilizarse la convención *lower camel case* para los parámetros de consulta, que PUEDEN ser abreviados.	AAJ, AAX
[RSG-06]	El patrón del URL para una API web DEBE contener la palabra 'api' en el URI.	AAJ, AAX, AX, AJ
[RSG-07]	NO DEBEN utilizarse parámetros matriciales.	AAJ, AAX, AX, AJ
[RSG-08]	Una API web DEBE aplicar sistemáticamente códigos de estado HTTP según se describe en los documentos RFC del IETF	AAJ, AAX, AX, AJ
[RSG-09]	Las API web DEBERÍAN utilizar los códigos recomendados en el Anexo V para clasificar los errores.	AAX, AAJ
[RSG-10]	Si la API detecta valores de entrada incorrectos, DEBE devolver el código de estado HTTP `400 Bad Request`. La carga útil de error DEBE indicar el valor incorrecto.	AAJ, AAX, AX, AJ
[RSG-11]	Si la API detecta nombres sintácticamente correctos de argumentos no esperados (en la petición o en los parámetros de consulta), DEBERÍA ignorarlos.	AAJ, AAX
[RSG-12]	Si la API detecta valores válidos que requieren funcionalidades no implementadas, DEBE devolver el código de estado HTTP `501 Not Implemented`. La carga útil de error DEBE indicar el valor sin implementar.	AAJ, AAX, AX, AJ
[RSG-13]	Una API web DEBERÍA utilizar únicamente recursos de nivel superior. Si hay subrecursos, deben ser colecciones e implicar una asociación. Una entidad debería ser accesible como recurso de nivel superior o como subrecurso, pero no por ambas vías.	AAJ, AAX
[RSG-14]	Si un recurso puede ser independiente, DEBE ser un recurso de nivel superior; de lo contrario será un subrecurso.	AAJ, AAX, AX, AJ
[RSG-15]	Para obtener recursos anidados, DEBEN utilizarse parámetros de consulta en lugar de rutas URL.	AAJ, AAX, AX, AJ
[RSG-16]	Los recursos DEBERÍAN nombrarse con sustantivos para las API web CRUD y con verbos para las API web Intent.	AAJ, AAX
[RSG-17]	Si el nombre del recurso es un sustantivo, DEBERÍA usarse siempre la forma plural. NO DEBERÍAN utilizarse formas sustantivas irregulares. Por ejemplo, en lugar de `/gente` debería usarse `/personas`.	AAJ, AAX
[RSG-18]	Los nombres de los recursos, los segmentos y los parámetros de consulta DEBEN estar compuestos por palabras en inglés, conforme a la grafía del inglés básica del Diccionario de Oxford (Oxford English Dictionary). PUEDEN estar en otros idiomas los nombres de recursos que deben traducirse por razones comerciales.	AAJ, AAX, AX, AJ
[RSG-19]	Una API web DEBERÍA utilizar para la negociación del tipo de contenido el encabezado de petición HTTP `Accept` y el encabezado de respuesta HTTP `Content-Type`.	AAJ, AAX
[RSG-20]	Una API web DEBE ser compatible con la negociación del tipo de contenido conforme a la Norma RFC 7231 del IETF.	AAJ, AAX, AX, AJ
[RSG-21]	DEBE utilizarse el formato JSON cuando no se solicita un tipo de contenido específico.	AAJ, AAX, AX, AJ
[RSG-22]	Una API web DEBERÍA devolver el código de estado `406 Not Acceptable` si no es compatible con el formato solicitado.	AAJ, AAX
[RSG-23]	Una API web DEBERÍA rechazar las peticiones que contengan encabezados de tipo de contenido inesperado o no aceptable y devolver el código de estado `HTTP 406 Not Acceptable` o `415 Unsupported Media Type`.	AAJ, AAX
[RSG-24]	Las peticiones y respuestas (convención de nombres, formato del mensaje, estructura de datos y diccionario de datos) DEBERÍAN ajustarse a la Norma ST.96 para XML o la Norma ST.97 para JSON.	AAX
[RSX-26]	Los nombres de los componentes XML DEBERÍAN proporcionarse en la convención *upper camel case*.	AAX
[RSG-27]	Una API web DEBE ser compatible como mínimo con el formato XML o JSON.	AAJ, AAX, AX, AJ

[RSG-28]	Los métodos HTTP DEBEN limitarse a los métodos estándares `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` y `HEAD`, conforme a las especificaciones de las normas RFC 7231 y RFC 5789 del IETF.	AAJ, AAX, AX, AJ
[RSG-29]	Los métodos HTTP PUEDEN seguir el principio de selección cuidadosa, que establece que solo se debe implementar la funcionalidad necesaria para el escenario de uso previsto.	AAJ, AAX
[RSG-30]	Algunos *proxies* solo son compatibles con los métodos `POST` y `GET`. Para sortear esa limitación, las API web pueden utilizar un método `POST` con un encabezado HTTP personalizado para crear un túnel sobre el método HTTP que se quiera usar. DEBERÍA utilizarse el encabezado HTTP personalizado `X-HTTP-Method`.	AAJ, AAX
[RSG-31]	Si un método HTTP no es admitido, DEBERÍA devolverse el código de estado HTTP `405 Method Not Allowed`.	AAJ, AAX
[RSG-32]	Una API web DEBERÍA admitir operaciones por lotes (también conocidas como operaciones masivas) en lugar de múltiples peticiones individuales con el fin de reducir la latencia. Se debería utilizar la misma semántica para los métodos HTTP y los códigos de estado HTTP. La carga útil de respuesta DEBERÍA contener información sobre todas las operaciones por lotes. Si se producen múltiples errores, la carga útil de error DEBERÍA contener información sobre todas las incidencias (en el atributo `details`). Todas las operaciones masivas DEBERÍAN ser ejecutadas mediante una instrucción atómica.	AAJ, AAX
[RSG-33]	Para un punto final que extrae un solo recurso, si este no se encuentra, el método GET DEBE devolver el código de estado `404 Not Found`. En el caso de los puntos finales que devuelven listas de recursos, simplemente devolverán una lista vacía.	AAJ, AAX, AX, AJ
[RSG-34]	Si un recurso es recuperado con éxito, el método `GET` DEBE devolver `200 OK`.	AAJ, AAX, AX, AJ
[RSG-35]	Las peticiones `GET` DEBEN ser idempotentes.	AAJ, AAX, AX, AJ
[RSG-36]	Cuando la longitud del URI excede los 255 bytes, DEBERÍA utilizarse el método POST en lugar del método `GET` debido a las limitaciones de este, o bien crear consultas con nombre si es posible.	AAJ, AAX
[RSG-37]	Las peticiones `HEAD` DEBEN ser idempotentes.	AAJ, AAX, AX, AJ
[RSG-38]	Algunos *proxies* solo son compatibles con los métodos `POST` y `GET`. Una API web DEBERÍA admitir un encabezado de petición HTTP personalizado para sustituir el método HTTP con el fin de sortear esa limitación.	AAJ, AAX
[RSG-39]	Las peticiones `POST` NO DEBEN ser idempotentes de acuerdo con la Norma RFC 2616 del IETF.	AAJ, AAX, AX, AJ
[RSG-40]	Si el recurso se creó correctamente, el encabezado HTTP `Location` DEBERÍA contener un URI (absoluto o relativo) que apunte al recurso creado.	AAJ, AAX
[RSG-41]	Si el recurso se creó correctamente, la respuesta DEBERÍA contener el código de estado `201 Created`.	AAJ, AAX
[RSG-42]	Si el recurso se creó correctamente, la carga útil de respuesta DEBERÍA contener por defecto el cuerpo del recurso creado, para que el cliente pueda utilizarlo sin hacer otra llamada HTTP.	AAJ, AAX
[RSG-43]	Las peticiones `PUT` DEBEN ser idempotentes.	AAJ, AAX, AX, AJ
[RSG-44]	Si no se encuentra un recurso, `PUT` DEBE devolver el código de estado `404 Not Found`.	AAJ, AAX, AX, AJ
[RSG-45]	Si un recurso se actualiza correctamente, `PUT` DEBE devolver el código de estado `200 OK` si se devuelve el recurso actualizado o `204 No Content` si no se devuelve.	AAJ, AAX, AX, AJ
[RSG-46]	Las peticiones `PATCH` NO DEBEN ser idempotentes.	AAJ, AAX, AX, AJ
[RSG-47]	Si una API web implementa actualizaciones parciales, `PATCH` DEBERÍA ser idempotente. Para conseguirlo, la API PUEDE recurrir a la técnica del bloqueo optimista recomendada en la Norma RFC 5789 del IETF.	AAJ, AAX
[RSG-48]	Si un recurso no se encuentra, `PATCH` DEBE devolver el código de estado `404 Not Found`.	AAJ, AAX, AX, AJ
[RSG-50]	Las peticiones `DELETE` NO DEBEN ser idempotentes.	AAJ, AAX, AX, AJ
[RSG-51]	Si no se encuentra un recurso, `DELETE` DEBE devolver el código de estado `404 Not Found`.	AAJ, AAX, AX, AJ
[RSG-52]	Si un recurso se elimina correctamente, `DELETE` DEBE devolver el código de estado `200 OK` si se devuelve el recurso eliminado o `204 No Content` si no se devuelve.	AAJ, AAX, AX, AJ
[RSG-53]	El destinatario final es el servidor de origen o el primer *proxy* o puerta de enlace que recibe un valor del encabezado `Max-Forwards` de cero en la petición. Las peticiones `TRACE` NO DEBEN incluir un cuerpo.	AAJ, AAX, AX, AJ
[RSG-54]	Las peticiones `TRACE` NO DEBEN ser idempotentes.	AAJ, AAX, AX, AJ

[RSG-55]	El valor del encabezado HTTP `Via` DEBE servir para rastrear la cadena de petición.	AAJ, AAX, AX, AJ
[RSG-56]	Para que el cliente pueda limitar la longitud de la cadena de petición DEBE utilizarse el encabezado HTTP `Max-Forwards`.	AAJ, AAX, AX, AJ
[RSG-57]	Si la petición es válida, la respuesta DEBERÍA contener el mensaje completo de la petición en el cuerpo de la respuesta, con `message/http` como `Content-Type`.	AAJ, AAX
[RSG-58]	Las respuestas a `TRACE` NO DEBEN ser almacenadas en caché.	AAJ, AAX, AX, AJ
[RSG-59]	DEBERÍA devolverse a `TRACE` el código de estado `200 OK`.	AAJ, AAX
[RSG-60]	Las peticiones `OPTIONS` DEBEN ser idempotentes.	AAJ, AAX, AX, AJ
[RSG-61]	NO DEBERÍAN utilizarse encabezados HTTP personalizados que empiecen con el prefijo "`X-`".	AAJ, AAX
[RSG-62]	NO DEBERÍAN utilizarse encabezados HTTP personalizados para modificar un método HTTP a menos que sea para resolver una limitación técnica (véase, por ejemplo, [RSG-39]).	AAJ, AAX
[RSG-63]	Los encabezados HTTP personalizados tendrán el formato `<organization>-<header name>`, y los nombres de `<organization>` y `<header>` DEBERÍAN seguir la convención *kebab case*.	AAJ, AAX
[RSG-64]	Una API web DEBERÍA admitir un único método de versionado de servicios, como el versionado por URI (por ejemplo, `/api/v1/inventors`), el versionado por encabezado (por ejemplo, `Accept-version: v1`), o el versionado por tipo de medios (por ejemplo, `Accept: application/vnd.v1+json`). NO DEBERÍA utilizarse el versionado por cadena de consulta.	AAJ, AAX
[RSG-65]	DEBERÍA seguirse un esquema de numeración de versiones que considere solo el número de versión mayor (por ejemplo, `/v1`).	AAJ, AAX
[RSG-66]	Los contratos de servicio API PUEDEN incluir la función de redireccionamiento del punto final. Cuando un consumidor de servicios intenta invocar un servicio, es posible que se devuelva una respuesta de redireccionamiento para indicarle que vuelva a enviar la solicitud a un nuevo punto final. Los redireccionamientos PUEDEN ser temporales o permanentes: – redireccionamiento temporal: mediante el encabezado de respuesta HTTP `Location` y el código de estado HTTP `302 Found`, conforme a la Norma RFC 7231 del IETF; o – redireccionamiento permanente: mediante el encabezado de respuesta HTTP `Location` y el código de estado HTTP `301 Moved Permanently`, de conformidad con el documento RFC 7238 del IETF.	AAJ, AAX
[RSG-67]	Los desarrolladores de API DEBERÍAN publicar las estrategias de ciclo de vida de las API con el fin de que los usuarios sepan cuánto tiempo se mantendrá una versión.	AAJ, AAX
[RSG-68]	Las API web DEBERÍAN ser compatibles con la paginación.	AAJ, AAX
[RSG-69]	Las peticiones paginadas PUEDEN NO ser idempotentes.	AAJ, AAX
[RSG-70]	Las API web DEBEN utilizar parámetros de consulta para implementar la paginación.	AAJ, AAX, AX, AJ
[RSG-71]	Las API web NO DEBEN utilizar encabezados HTTP para implementar la paginación.	AAJ, AAX, AX, AJ
[RSG-72]	DEBERÍAN utilizarse los parámetros de consulta `limit=<number of items to deliver>` y `offset=<number of items to skip>; limit` es el número de elementos que se devolverán (tamaño de la página), y `offset` es el número de elementos que se saltarán (desplazamiento). Si no se especifica un límite de tamaño de página, DEBERÍA definirse un valor por defecto, general o por colección; el offset por defecto DEBE ser cero “0”. A continuación se muestra un ejemplo de un URL válido: `https://wipo.int/api/v1/patents?limit=10&offset=20`	AAJ, AAX
[RSG-73]	Los valores de los parámetros `limit` y `offset` DEBERÍAN incluirse en la respuesta.	AAJ, AAX
[RSG-74]	Una API web DEBERÍA admitir la ordenación.	AAJ, AAX
[RSG-75]	Para establecer criterios multiatributos de ordenación, DEBE utilizarse un parámetro de consulta. El valor de dicho parámetro será una lista de claves de ordenación separadas por comas a las que se puede añadir, mediante dos puntos ':', la dirección de ordenamiento, con `asc` para el orden ascendente o `desc` para el	AAJ, AAX, AX, AJ

	orden descendente. En caso de que no se especifique una dirección de ordenación para una clave, el servidor DEBE establecer una por defecto.	
[RSG-76]	Las API web DEBERÍAN devolver los criterios de ordenación en la respuesta.	AAJ, AAX, AX, AJ
[RSG-77]	Una API web PUEDE ser compatible con la expansión del cuerpo del contenido que se devuelve. DEBERÍA utilizarse el parámetro de consulta `expand=<comma-separated list of attributes names>`.	AAJ, AAX
[RSG-78]	En caso de que una API web admita la proyección, DEBERÍA utilizarse, en lugar de las rutas URL, un parámetro de consulta con el formato `"fields="<comma-separated list of attribute names>`.	AAJ, AAX
[RSG-79]	Las API web DEBEN admitir la devolución del número de elementos de una colección.	AAJ, AAX, AX, AJ
[RSG-80]	Para permitir la devolución del número de elementos de una colección DEBE utilizarse un parámetro de consulta.	AAJ, AAX, AX, AJ
[RSG-81]	DEBERÍA utilizarse el parámetro de consulta `count` para devolver el número de elementos de una colección.	AAJ, AAX
[RSG-82]	Las API web PUEDEN admitir la devolución del número de elementos de una colección mediante una función *inline*, es decir, como parte de la respuesta que contiene la propia colección. DEBE utilizarse un parámetro de consulta.	AAJ, AAX, AX, AJ
[RSG-83]	DEBERÍA utilizarse el parámetro de consulta `count=true`. Si no se especifica el valor de `count`, debería fijarse por defecto en `false`.	AAJ, AAX
[RSG-84]	Si una API web admite la paginación, DEBERÍA permitir la devolución del número de la colección (es decir, el número total de elementos de la colección) en la respuesta mediante una función *inline*.	AAJ, AAX
[RSG-85]	Si una API web es compatible con expresiones de búsqueda complejas, DEBERÍA especificarse un lenguaje de consulta, como el CQL.	AAJ, AAX
[RSG-86]	Un contrato de servicio DEBE especificar la gramática compatible (como los campos, las funciones, las palabras clave y los operadores).	AAJ, AAX, AX, AJ
[RSG-87]	DEBE utilizarse el parámetro de consulta '`q`'.	AAJ, AAX, AX, AJ
[RSG-88]	En el nivel de protocolo, una API web DEBE devolver un código de estado HTTP apropiado seleccionado de la lista de códigos de estado HTTP estándares.	AAJ, AAX, AX, AJ
[RSJ-89]	En el nivel de aplicación, una API web DEBE devolver una carga útil que informe del error con suficiente granularidad. Los atributos `code` y `message` son obligatorios, el atributo `details` es condicionalmente obligatorio y los atributos `target`, `status`, `moreInfo`, e `internalMessage` son opcionales.	AAJ, AAX, AX, AJ
[RSG-90]	Los errores NO DEBEN exponer datos esenciales para la seguridad o detalles técnicos internos, como las pilas de llamada en los mensajes de error.	AAJ, AAX, AX, AJ
[RSG-91]	El encabezado HTTP `Reason-Phrase` (descrito en la Norma RFC 2616) NO DEBE utilizarse para transmitir mensajes de error.	AAJ, AAX, AX, AJ
[RSG-92]	Cada error registrado DEBERÍA tener un ID de correlación único. DEBERÍA utilizarse un encabezado HTTP personalizado con el nombre Correlation-ID.	AAJ, AAX
[RSG-93]	El modelo de contrato de servicio DEBE especificar lo siguiente: – la versión de la API; – información sobre la semántica de los elementos de la API; – los recursos; – los atributos de los recursos; – los parámetros de consulta; – los métodos; – los tipos de medios; – la gramática de búsqueda (si se admite alguna); – los códigos de estado HTTP; – los métodos HTTP; – las restricciones y las características distintivas; y – la seguridad (por ejemplo, los esquemas privados).	AAJ, AAX, AX, AJ
[RSG-94]	El modelo de contrato de servicio DEBERÍA incluir peticiones y respuestas en el esquema XML o en el esquema JSON y ejemplos de uso de la API en los formatos compatibles, a saber, XML o JSON.	AAJ, AAX
[RSG-95]	Una API REST DEBE proporcionar la documentación de la API como un contrato de servicio.	AAJ, AAX, AX, AJ

[RSG-96]	Toda implementación de una API web que no cumpla con la presente norma DEBE ser documentada explícitamente en el contrato de servicio. Si no se especifica ninguna desviación de una de sus normas en el contrato de servicio, DEBE asumirse que se sigue la presente norma.	AAJ, AAX, AX, AJ
[RSG-97]	Un contrato de servicio DEBE permitir la generación de código esquemático de cliente API.	AAJ, AAX, AX, AJ
[RSG-98]	Un contrato de servicio DEBERÍA permitir la generación de código esquemático de servidor.	AAJ, AAX
[RSG-99]	La documentación de las API web DEBERÍA escribirse utilizando el RAML o la OAS. NO DEBERÍAN utilizarse formatos de documentación personalizados.	AAJ, AAX
[RSG-100]	Un consumidor de API web DEBERÍA poder especificar un tiempo de espera del servidor para cada petición; DEBERÍA utilizarse un encabezado HTTP personalizado. También DEBERÍA establecerse un tiempo máximo de espera del servidor para evitar un uso excesivo de los recursos del servidor.	AAJ, AAX
[RSG-101]	Una API web DEBERÍA admitir la recuperación condicional de datos, para asegurar que solo se recuperen los datos que se hayan modificado. DEBERÍA utilizarse la validación de recursos basada en el contenido por ser la más precisa.	AAJ, AAX
[RSG-102]	Para implementar la validación de recursos basada en el contenido, DEBERÍA utilizarse el encabezado HTTP `ETag` en la respuesta para codificar el estado de los datos. El valor de `Etag` DEBERÍA utilizarse en los encabezados condicionales HTTP (como `If-Match` o `If-None-Match`) de las peticiones subsiguientes. Si no se modificaron los datos desde que la petición devolvió el `ETag`, el servidor DEBERÍA devolver el código de estado `304 Not Modified`. Este mecanismo se especifica en las normas RFC 7231 y RFC 7232 del IETF.	AAJ, AAX
[RSG-103]	Para implementar la validación de recursos basada en el tiempo DEBERÍA utilizarse el encabezado HTTP `Last-Modified`. Este mecanismo se especifica en las normas RFC 7231 y RFC 7232 del IETF.	AAJ, AAX
[RSG-104]	Utilizando el versionado por respuesta, un consumidor de servicios PUEDE implementar el bloqueo optimista.	AAJ, AAX
[RSG-106]	DEBERÍAN utilizarse los encabezados de respuesta HTTP `Cache-Control` y `Expires`. `Expires` PUEDE utilizarse para prestar asistencia a los clientes heredados.	AAJ, AAX
[RSG-107]	Una API web DEBE indicar si permite descargas de archivos por partes, mediante respuestas a peticiones `HEAD` que incluyan los encabezados de respuesta HTTP `Accept-Ranges` y `Content-Length`.	AAJ, AAX
[RSG-108]	Una API web DEBERÍA permitir la descarga de archivos por partes. DEBERÍA admitirse la petición de múltiples rangos.	AAJ, AAX
[RSG-109]	Una API web DEBERÍA especificar si admite la subida de archivos por partes.	AAJ, AAX
[RSG-110]	Una API web DEBERÍA ser compatible con la subida de archivos por partes. DEBERÍA admitirse la petición de múltiples rangos.	AAJ, AAX
[RSG-111]	El proveedor de servicios DEBERÍA devolver con los encabezados de respuesta HTTP el código de estado HTTP `413 Request Entity Too Large` en caso de que la petición supere el límite máximo permitido. PUEDE utilizarse un encabezado HTTP personalizado para indicar el tamaño máximo de la petición.	AAJ, AAX
[RSG-112]	Si una API web admite la gestión de preferencias, esta DEBERÍA implementarse de acuerdo con la Norma RFC 7240 del IETF, es decir, DEBERÍA utilizarse el encabezado de petición HTTP `Prefer` y devolverse el encabezado de respuesta HTTP `Preference-Applied` (que incluye la petición original).	AAJ, AAX
[RSG-113]	Si una API web es compatible con la gestión de preferencias, en el contrato de servicio DEBE indicarse la nomenclatura de las preferencias que PUEDEN establecerse mediante el encabezado `Prefer`.	AAJ, AAX, AJ, AX
[RSG-114]	Si una API web permite la traducción de datos, DEBE admitir el encabezado de petición HTTP `Accept-Language` con el que se indica el conjunto de lenguas naturales preferidas para la respuesta, conforme a lo especificado en la Norma RFC 7231 del IETF.	AAJ, AAX, AJ, AX
[RSG-115]	Si la API permite operaciones de larga duración, estas DEBERÍAN ejecutarse de forma asíncrona. DEBERÍA seguirse el enfoque descrito a continuación: a) el consumidor de servicios activa la operación de servicio; b) la operación de servicio devuelve el código de estado 202 Accepted conforme a la Norma RFC 7231 del IETF (sección 6.3.3), que indica que la petición fue aceptada para su procesamiento, pero este no se completó.	AAJ, AAX

	La ubicación de la tarea pendiente que se creó también se devuelve con el encabezado HTTP Location; y c) el consumidor de servicios llama a la ubicación devuelta para saber si el recurso está disponible. Si el recurso no está disponible, la respuesta DEBERÍA incluir el código de estado 200 OK y el estado de la tarea (por ejemplo, pendiente) y PUEDE contener otra información (por ejemplo, un indicador de progreso y/o un enlace para cancelar o eliminar la tarea mediante el método HTTP DELETE). Si el recurso está disponible, la respuesta DEBERÍA incluir el código de estado 303 See Other, y el encabezado HTTP Location DEBERÍA contener el URL para recuperar los resultados de la tarea.	
[RSG-116]	Confidencialidad: las API y su información DEBEN ser identificadas, clasificadas y protegidas en todo momento frente a acciones no autorizadas de acceso, difusión e interceptación. DEBEN seguirse los principios de mínimos privilegios, cero confianza, necesidad de conocer y necesidad de compartir.	AAJ, AAX, AJ, AX
[RSG-117]	Garantía de integridad: las API y su información DEBEN estar protegidas contra acciones no autorizadas de modificación, duplicación, corrupción y destrucción. La modificación de la información DEBE realizarse mediante transacciones e interfaces aprobadas. La actualización de los sistemas DEBE llevarse a cabo mediante procesos aprobados de gestión de la configuración, gestión de cambios y gestión de parches.	AAJ, AAX, AJ, AX
[RSG-118]	Disponibilidad: las API y su información DEBEN estar disponibles para los usuarios autorizados en el debido momento, conforme se especifica en los acuerdos de nivel de servicio, las políticas de control de acceso y los procesos operativos definidos.	AAJ, AAX, AJ, AX
[RSG-119]	No repudio: toda transacción procesada o acción realizada por las API DEBE garantizar el no repudio mediante la auditoría, autorización y autenticación adecuadas, rutas seguras y servicios y mecanismos de no repudio.	AAJ, AAX, AJ, AX
[RSG-120]	Autenticación, autorización, auditoría: los usuarios, los sistemas, las API o los dispositivos que participen en transacciones o acciones importantes DEBEN ser autenticados, autorizados mediante servicios de control de acceso basados en roles o atributos y mantener la separación de tareas. Además, todas las acciones DEBEN ser registradas y el nivel de autenticación debe aumentar en función del riesgo asociado a la información.	AAJ, AAX, AJ, AX
[RSG-121]	En el desarrollo de las API DEBEN tenerse debidamente en cuenta las amenazas, los casos de uso malicioso, las técnicas de codificación segura, la seguridad de la capa de transporte (TLS) y las pruebas de seguridad, sobre todo: – los PUT y POST: qué modificaciones de los datos internos podrían utilizarse para atacar o transmitir información errónea; – los DELETE: podrían utilizarse para eliminar el contenido de un almacén de recursos internos; – los métodos permitidos de la lista blanca: para asegurar que los métodos HTTP permitidos sean debidamente restringidos y que otros devuelvan un código de respuesta adecuado; y – los ataques más conocidos deberían considerarse durante la fase de modelado de amenazas del proceso de diseño para asegurar que el riesgo de amenaza no aumente. DEBEN tenerse en cuenta las amenazas y medidas de mitigación definidas en la lista de los diez riesgos más críticos de OWASP.	AAJ, AAX, AJ, AX
[RSG-122]	En el desarrollo de las API DEBERÍAN seguirse las normas y las mejores prácticas que se enumeran a continuación: – las mejores prácticas de codificación segura: *OWASP Secure Coding Principles* (Principios de Codificación Segura de OWASP); – la seguridad de las API Rest: *REST Security Cheat Sheet* (Lista de Recomendaciones Prácticas sobre Seguridad REST); – las entradas de escape y la protección de secuencias de comandos en sitios cruzados: *OWASP XSS Cheat Sheet* (Lista de Recomendaciones	AAJ, AAX, AJ, AX

	Prácticas de OWASP sobre Secuencias de Comandos en Sitios Cruzados); – la prevención de ataques de inyección SQL: *OWASP SQL Injection Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Inyección SQL), *OWASP Parameterization Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Parametrización); y – la seguridad de la capa de transporte: *OWASP Transport Layer Protection Cheat Sheet* (Lista de Recomendaciones Prácticas de OWASP sobre Seguridad de la Capa de Transporte).	
[RSG-123]	DEBEN realizarse pruebas de seguridad y evaluaciones de vulnerabilidad para garantizar que las API son seguras y resistentes a las amenazas. Para ello PUEDEN aprovecharse las pruebas de seguridad de aplicaciones estáticas y dinámicas (SAST/DAST), las herramientas de gestión automatizada de vulnerabilidades y las pruebas de penetración.	AAJ, AAX, AJ, AX
[RSG-124]	Los servicios protegidos DEBEN proporcionar puntos finales HTTP solo mediante TLS 1.2, o una versión superior, con un paquete de cifrado que incluya el protocolo Elliptic-curve Diffie–Hellman (ECDH) para el intercambio de claves.	AAJ, AAX, AJ, AX
[RSG-125]	Cuando se consideran los protocolos de autenticación, DEBERÍA utilizarse una confidencialidad directa perfecta para garantizar la seguridad del transporte. NO DEBERÍAN permitirse los algoritmos criptográficos inseguros ni la compatibilidad hacia atrás con SSL 3 y TLS 1.0/1.1.	AAX, AAJ
[RSG-126]	Para una máxima seguridad y confianza, DEBERÍA establecerse una VPN Ipsec de sitio a sitio para proteger mejor la información transmitida a través de redes inseguras.	AAX, AAJ
[RSG-127]	La aplicación consumidora DEBERÍA validar la cadena de certificados TLS al realizar peticiones a recursos protegidos, incluida la comprobación de la lista de revocación de certificados.	AAX, AAJ
[RSG-128]	Los servicios protegidos DEBERÍAN utilizar solo certificados válidos emitidos por una autoridad certificadora de confianza.	AAX, AAJ
[RSG-129]	Los *tokens* DEBERÍAN ser firmados utilizando algoritmos de firma segura que cumplan con el estándar de firma digital (DSS) FIPS 186-4. DEBERÍAN considerarse los algoritmos de firma digital de Rivest, Shamir y Adleman (RSA) o de curva elíptica (ECDSA).	AAX, AAJ
[RSG-130]	La autenticación anónima DEBE utilizarse solo cuando los clientes y la aplicación que están utilizando acceden a información o funciones con un nivel de sensibilidad bajo que no deberían requerir autenticación, como la información pública.	AAJ, AAX, AJ, AX
[RSG-131]	NO DEBE permitirse la autenticación con nombre de usuario y contraseña o con contraseña *hash* [generada mediante una función *hash* o función resumen].	AAJ, AAX, AJ, AX
[RSG-132]	Si un servicio está protegido, DEBERÍA usarse el protocolo OpenID Connect.	AAX, AAJ
[RSG-133]	Cuando se utiliza un JSON Web Token (JWT), el secreto JWT DEBERÍA tener una entropía alta para aumentar el factor de trabajo de los ataques de fuerza bruta; los *tokens* TTL y RTTL DEBERÍAN ser lo más cortos posibles; y la información sensible NO DEBERÍA almacenarse en la carga útil JWT.	AAX, AAJ
[RSG-134]	En las peticiones `POST y PUT`, los datos sensibles DEBERÍAN ser transferidos en el cuerpo de la petición o mediante los encabezados de la petición.	AAX, AAJ
[RSG-135]	En las peticiones `GET`, los datos sensibles DEBERÍAN ser transferidos en un encabezado HTTP.	AAX, AAJ
[RSG-136]	Para minimizar la latencia y reducir el acoplamiento entre los servicios protegidos, la decisión de control de acceso DEBERÍA ser tomada localmente por los puntos finales REST.	AAX, AAJ
[RSG-137]	DEBERÍAN utilizarse claves de API en los servicios protegidos y públicos para evitar que se sobrecargue a sus proveedores de servicios con múltiples peticiones (ataques de denegación de servicio). En el caso de los servicios protegidos, PUEDEN utilizarse claves de API para la monetización (planes adquiridos), la aplicación de políticas de calidad de servicio y la supervisión.	AAX, AAJ
[RSG-138]	Las claves de API PUEDEN combinarse con el encabezado de petición HTTP `user-agent` para determinar si el usuario es una persona o un agente de software, conforme a lo especificado en la Norma RFC 7231 del IETF.	AAX, AAJ
[RSG-139]	El proveedor de servicios DEBERÍA devolver junto con los encabezados de respuesta HTTP el estado de uso en el momento. Se PUEDEN devolver los siguientes datos de respuesta:	AAX, AAJ

	– límite de tasa (*rate limit*): máximo número de peticiones por minuto permitido en el sistema; – límite de tasa restante (*rate limit remaining*): número de peticiones restantes (-1 indica que se ha superado el límite); y – restablecimiento del límite de tasa (*rate limit reset*): tiempo (en segundos) necesario para que se restablezca el número máximo de peticiones permitidas.	
[RSG-140]	El proveedor de servicios DEBERÍA devolver el código de estado `429 Too Many Requests` si se reciben demasiadas peticiones en poco tiempo.	AAX, AAJ
[RSG-141]	Las claves de API DEBEN ser revocadas si el cliente incumple el contrato de uso, conforme a lo especificado por la oficina de propiedad intelectual.	AAJ, AAX, AJ, AX
[RSG-142]	La transferencia de las claves de API DEBERÍA realizarse mediante encabezados HTTP personalizados. NO DEBERÍA realizarse mediante parámetros de consulta.	AAX, AAJ
[RSG-143]	Las claves de API DEBERÍAN generarse de forma aleatoria.	AAX, AAJ
[RSG-144]	Los certificados seguros y de confianza DEBEN ser emitidos por una autoridad certificadora de confianza mutua mediante un proceso de establecimiento de confianza o una certificación cruzada.	AAJ, AAX, AJ, AX
[RSG-145]	DEBERÍAN utilizarse certificados compartidos entre el cliente y el servidor, por ejemplo X.509, para mitigar los riesgos de seguridad de la identidad propios de los sistemas sensibles y las acciones privilegiadas.	AAJ, AAX, AJ, AX
[RSG-146]	Para servicios con un alto nivel de privilegio, la autenticación mutua bidireccional entre el cliente y el servidor DEBERÍA utilizar certificados para ofrecer una mayor protección.	AAX, AAJ
[RSG-147]	DEBERÍA implementarse la autenticación multifactor para mitigar los riesgos de la identidad en aplicaciones con un perfil de alto riesgo, en sistemas que procesen información muy sensible o en acciones privilegiadas.	AAX, AAJ
[RSG-148]	Si la API REST es pública, el valor del encabezado HTTP `Access-Control-Allow-Origin` DEBE ser '*'.	AAJ, AAX, AJ, AX
[RSG-149]	Si la API REST está protegida, DEBERÍA usarse el intercambio de recursos de origen cruzado (CORS), si es posible. De lo contrario, PUEDE utilizarse JSONP como alternativa, pero solo para peticiones `GET`, por ejemplo, cuando el usuario accede con un navegador antiguo. NO DEBERÍA utilizarse `iframe`.	AAX, AAJ

[Sigue el Anexo II]

ANEXO II

VOCABULARIO DE PI RELATIVO A LA ARQUITECTURA REST**ERROR! BOOKMARK NOT DEFINED.**

Versión 1.1

Revisión aprobada por el Comité de Normas Técnicas de la OMPI (CWS)
en su décima sesión, celebrada el 25 de noviembre de 2022

1. En el Cuadro 5 se proporciona vocabulario de PI, a saber, ejemplos de parámetros de petición de servicios básicos RESTful. Es posible que las oficinas de PI tengan que desarrollar peticiones más complejas y cargas útiles de respuesta variadas en función de sus necesidades operativas. Los parámetros que figuran en el cuadro son ejemplos de elementos de la Norma ST.96, en *lower camel case*, utilizados para las respuestas JSON. El diccionario completo de datos de PI y los esquemas XML de PI de la Norma ST.96 pueden consultarse en: https://www.wipo.int/standards/en/st96/v5-0/.

[Nota editorial: En el futuro, está previsto proporcionar un enlace a una lista más exhaustiva de vocabulario de PI relativo a REST para XML y JSON, que se actualizará de forma continua a medida que se vayan desarrollando elementos y vocabulario de PI.]

Cuadro 5. Ejemplos de vocabulario de las API, en *lower camel case,* sobre cada actividad, siguiendo los esquemas XSD de la Norma ST.96

Dominio de actividad	Nombre del recurso	Nombre del parámetro	Descripción
TODO	`/trademarks` `/patents` `/designs`	`st13ApplicationNumber`	Número de solicitud de PI presentada en el formato establecido en la Norma ST.13 de la OMPI, a saber, una cadena de varios valores, incluidos el número de solicitud nacional, el tipo de PI y el país/organización.
TODO	`/trademarks` `/patents` `/designs`	`applicationNumber`	Número de solicitud de PI presentada en el formato de la oficina nacional.
MÚLTIPLE	`/trademarks` `/designs`	`internationalRegistrationNumber`	Número del registro internacional correspondiente a los derechos de PI. En el caso de las marcas, este tema es competencia del Sistema de Madrid, y, en el caso de los dibujos y modelos industriales, es competencia del Sistema de La Haya.
TODO	`/trademarks` `/patents` `/designs`	`availableDocument`	Entrada de documento único relevante para los criterios de búsqueda proporcionados al servicio DocList de la API.
TODO	`/trademarks` `/patents` `/designs`	`sortingCriteria`	Criterio de ordenación utilizado por el servicio DocList de la API.
TODO	`/trademarks` `/patents` `/designs`	`receivingOfficeCode`	Código de la oficina de PI, en el formato establecido en la Norma ST.2 de la OMPI.
TODO	`/trademarks` `/patents` `/designs`	`receivingOfficeDate`	Fecha de recepción en la oficina de PI.
Marcas	`/trademarks`	`registrationDate`	Fecha de registro en la oficina de PI.
		`applicationDate`	Fecha de la solicitud.
		`markCurrentStatusCode`	Código correspondiente a la situación jurídica en que se encuentra la solicitud.

		markCurrentStatusDate	Fecha de consulta de la situación jurídica en que se encuentra la solicitud.
Patentes	/patents	filingDate	Fecha de presentación de la solicitud.
		grantPublicationDate	Fecha de publicación de la concesión.
		fileReferenceIdentifier	Número de referencia de los solicitantes.
		applicationBodyStatus	Situación del organismo solicitante.
		statusEventData	Datos sobre una incidencia relativa a la situación jurídica de una solicitud de patente específica.
		keyEventCode	Código relativo a una actividad general de alto nivel que abarca los casos más generales e importantes de una categoría.
Dibujos y modelos industriales	/designs	applicationDate	Fecha de presentación de la solicitud.
		designApplicationCurrentStatus	Categoría de la situación jurídica en que se encuentra la solicitud de registro de dibujos o modelos.
		designApplicationCurrentStatusDate	Fecha de consulta de la situación jurídica en que se encuentra la solicitud.

2. En el Cuadro 6 que figura a continuación se especifican los parámetros de consulta técnica que se deberían aplicar a todos los servicios de las API REST.

Cuadro 6. Vocabulario técnico de las API

Parámetro de consulta/ruta	Parámetro Valor Tipo de datos	Restricción	Formato	Descripción	Norma de diseño
format	cadena		type/subtype; parameter=value conforme a la Norma RFC 7231, sección 3.1.1.1. (Tipo de medios)	Se utiliza para la negociación del tipo de contenido (debe darse preferencia a un encabezado de petición HTTP)	[RSG-19]
v	cadena		v%, donde % es un entero positivo	Se utiliza para el versionado de servicios (debe darse preferencia a la indicación de la versión mediante un segmento de la ruta del URL)	[RSG-64]
limit	entero	positivo	limit=10	Tamaño de página utilizado para la paginación	[RSG-73]
offset	entero	positivo; el valor por defecto es 0	offset=5	Desplazamiento utilizado para la paginación	[RSG-73]
sort	lista de cadenas separadas por comas	valores posibles: – asc – desc	sort=key1:asc,key2:desc	Criterios multiatributos de ordenación	[RSG-74] a [RSG-76]

`expand`	lista de cadenas separadas por comas		`expand=key1,key2`	Se utiliza para expandir el cuerpo del contenido devuelto	[RSG-77]
`count`	booleano	el valor por defecto es `false`	`count=true`	Devuelve el número de elementos de una colección (puede ser mediante una función *inline*)	[RSG-81]
`apiKey`	cadena		`apiKey=abcdef12345`	Se utiliza para indicar una clave de API web (debe darse preferencia a un encabezado HTTP)	[RSG-137] a [RSG-138]

[Sigue el Anexo III]

ANEXO III

DIRECTRICES DE LAS API WEB RESTFUL Y MODELO DE CONTRATO DE SERVICIO

Versión 1.1

Revisión aprobada por el Comité de Normas Técnicas de la OMPI (CWS)
en su décima sesión, celebrada el 25 de noviembre de 2022

1. En el Anexo III se ofrecen dos modelos de especificaciones de API conformes a la norma que tienen por objeto orientar a las Oficinas de PI que deseen desarrollar servicios web conformes a esta norma. A continuación y en los Apéndices A y B se ofrecen detalles sobre los dos modelos de ejemplo.

2. Cabe señalar que los modelos de ejemplo se han elaborado con un planteamiento híbrido de los enfoques *contract-first* y de *code-first*.

Primer modelo de ejemplo: DocList

3. El primero de los modelos de ejemplo está inspirado en el servicio web del sistema Office Open Portal Dossier (OPD), provisto del mismo nombre, de las Oficinas de la Cooperación Pentalateral (IP5)[17]. El servicio DocList de las API proporciona una lista de los documentos de patente asociados a como mínimo un número de solicitud o de publicación.

Segundo modelo de ejemplo: Situación jurídica de las patentes

4. El segundo de los modelos de ejemplo es una API sobre la situación jurídica de las patentes, que proporciona el historial de incidencias relativas a la situación jurídica de una solicitud determinada o información detallada sobre una incidencia concreta relativa a la situación jurídica.

[Siguen los apéndices A y B del Anexo III]

[17] Las Oficinas de la Cooperación Pentalateral (IP5) son la Administración Nacional de Propiedad Intelectual de China (CNIPA), la Oficina Europea de Patentes (OEP), la Oficina Japonesa de Patentes (JPO), la Oficina Surcoreana de Propiedad Intelectual (KIPO) y la Oficina de Patentes y Marcas de los Estados Unidos de América (USPTO).

APÉNDICE A

MODELO DE EJEMPLO DOCLIST

1. En el Apéndice A se proporciona un enlace a un fichero ZIP que incluye el documento de requisitos en el que se describen los formatos de petición y respuesta, la especificación YAML y los componentes XSD.

2. El Apéndice A puede consultarse en:
https://www.wipo.int/edocs/mdocs/cws/en/cws_8/cws_8_2-appendixa.zip

APÉNDICE B

MODELO DE EJEMPLO SOBRE LA SITUACIÓN JURÍDICA DE LAS PATENTES

1. En el Apéndice B figura un enlace a un fichero ZIP que incluye la especificación de la API proporcionada en RAML, datos de ejemplo y listas de enumeración de la Norma ST.96 de la OMPI.

2. El Apéndice B puede consultarse en:
https://www.wipo.int/edocs/mdocs/cws/en/cws_8/cws_8_2-appendixb.zip

[Sigue el Anexo IV]

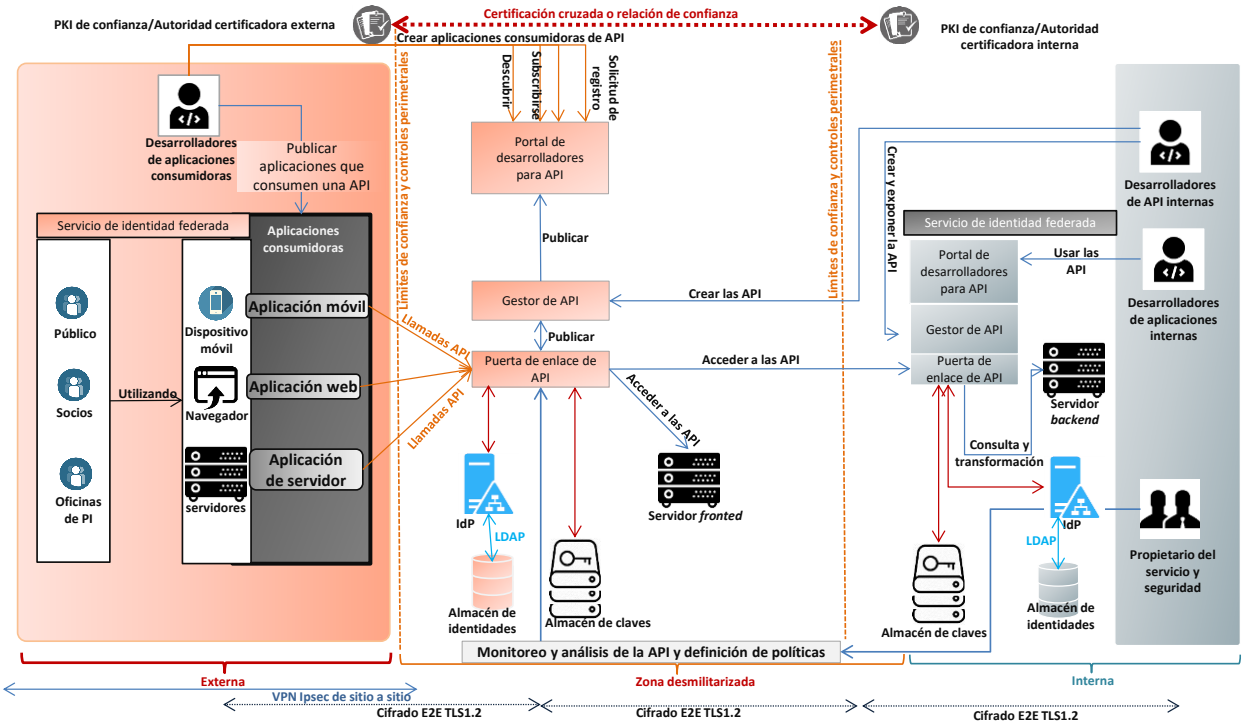
ANEXO IV

MEJORES PRÁCTICAS DE ARQUITECTURA DE SEGURIDAD DE ALTO NIVEL

Versión 1.1

Revisión aprobada por el Comité de Normas Técnicas de la OMPI (CWS)
en su décima sesión, celebrada el 25 de noviembre de 2022

1. La arquitectura de seguridad define los servicios y mecanismos que deberían aplicarse para cumplir con las políticas y normas establecidas, al tiempo que proporciona un marco para lograr una mayor normalización y automatización de la seguridad. Los servicios y mecanismos básicos del marco de seguridad de la API (el portal de desarrollo, el gestor de API y la puerta de enlace de API) permiten agrupar funciones. Esas funciones o servicios pueden proporcionarse mediante aplicaciones discretas, desarrollo de código personalizado o productos COTS, o bien aprovechando las tecnologías existentes que pueden configurarse con ese fin. Algunas de las funciones pueden solaparse o combinarse en uno o más productos dependiendo del proveedor utilizado.



2. La arquitectura de seguridad recomendada DEBERÍA incluir los siguientes servicios y mecanismos de seguridad para la API:

- un portal API web para proporcionar funciones como *API Discovery*, *API Analytics*, y acceso a especificaciones y descripciones, incluidos acuerdos de nivel de servicio, redes sociales y preguntas frecuentes;
- un gestor de API web para facilitar la administración y gestión centralizadas de los catálogos de API, la gestión del registro y de la incorporación de varias comunidades de desarrolladores de API, la gestión del ciclo de vida de la API, la aplicación de perfiles de seguridad predefinidos y la gestión del ciclo de vida de las políticas de seguridad;
- una puerta de enlace de API web para proporcionar capacidades de automatización de la seguridad, incluidas, entre otras, protección centralizada contra amenazas, autenticación de API centralizada, autorización, registro, aplicación de políticas de seguridad, cifrado de mensajes, monitorización y análisis;

- un servicio de monitoreo y análisis de la API web para proporcionar funciones como la monitorización avanzada de servicios de API, el análisis, uso de perfiles para líneas de base de seguridad, cambios de uso y demanda;
- un almacén de credenciales para proporcionar capacidades de almacenamiento seguro de, entre otras cosas, claves de API, secretos y certificados;
- una autoridad certificadora de confianza para emitir certificados seguros e instaurar la confianza entre las distintas oficinas;
- un sistema de información de seguridad y gestión de eventos (SIEM) para permitir la correlación de los registros de seguridad y el análisis y monitoreo de seguridad avanzados;
- un proveedor de identidad (IdP) para gestionar las identidades almacenadas en los directorios LDAP y activar la autenticación; y
- un producto de análisis de aplicaciones web que realiza análisis de seguridad regulares basados en una línea base de seguridad confiable como el OWASP Top 10.

[Sigue el Anexo V]

ANEXO V

CÓDIGOS DE ESTADO HTTP

Versión 1.1

Revisión aprobada por el Comité de Normas Técnicas de la OMPI (CWS)
en su décima sesión, celebrada el 25 de noviembre de 2022

1. Es importante que cada respuesta corresponda a un código de estado HTTP apropiado y que se utilicen los códigos HTTP estándares. Además de utilizarse un código de estado apropiado, el cuerpo de la respuesta HTTP debería incluir una descripción útil y concisa del error. Las respuestas deberían ser específicas y claras para que los consumidores puedan llegar fácilmente a una conclusión cuando utilicen la API.

2. Los códigos de estado HTTP se definen conforme a la Norma RFC 7231. A continuación figura una lista de los códigos de estado que debería devolver una API, según corresponda.

3. Se definen las siguientes categorías de códigos de estado de respuesta:

- códigos con formato 1xx: respuestas informativas. Comunican información a nivel de protocolo de transferencia;
- códigos con formato 2xx: respuestas correctas. Indican que la petición del cliente se procesó correctamente;
- códigos con formato 3xx: respuestas de redirección. Indican que el cliente debe realizar alguna otra acción para poder completar su petición;
- códigos con formato 4xx: respuestas de error del cliente. Esta categoría de códigos de estado indica que el cliente ha cometido un error; y
- códigos con formato 5xx: respuestas de error del servidor. Indican que se ha producido un error ocasionado por el servidor.

4. En el siguiente cuadro se especifican los códigos de estado HTTP y los documentos RFC del IETF conexos como referencias.

Valor	Descripción	Referencia
100	`Continue` (Continuar)	Norma RFC 7231, sección 6.2.1
101	`Switching Protocols` (Cambiando de protocolos)	Norma RFC 7231, sección 6.2.2
102	`Processing` (Procesando)	Norma RFC 2518
103	`Early Hints` (Indicaciones tempranas)	Documento RFC 8297
104-199	`Unassigned` (No asignado)	
200	`OK` (Todo es correcto)	Norma RFC 7231, sección 6.3.1
201	`Created` (Creado)	Norma RFC 7231, sección 6.3.2
202	`Accepted` (Aceptado)	Norma RFC 7231, sección 6.3.3
203	`Non-Authoritative Information` (Información no autorizada)	Norma RFC 7231, sección 6.3.4
204	`No Content` (Sin contenido)	Norma RFC 7231, sección 6.3.5
205	`Reset Content` (Restablecer el contenido)	Norma RFC 7231, sección 6.3.6
206	`Partial Content` (Contenido parcial)	Norma RFC 7233, sección 4.1
207	`Multi-Status` (Multiestado)	Norma RFC 4918
208	`Already Reported` (Ya transmitido)	Documento RFC 5842

209-225	`Unassigned` (No asignado)	
226	`IM Used` (Utilizado IM)	Norma RFC 3229
227-299	`Unassigned` (No asignado)	
300	`Multiple Choices` (Opciones multiples)	Norma RFC 7231, sección 6.4.1
301	`Moved Permanently` (Movido permanentemente)	Norma RFC 7231, sección 6.4.2
302	`Found` (Encontrado)	Norma RFC 7231, sección 6.4.3
303	`See Other` (Ver otro)	Norma RFC 7231, sección 6.4.4
304	`Not Modified` (No modificado)	Norma RFC 7232, sección 4.1
305	`Use Proxy` (Uso de proxy)	Norma RFC 7231, sección 6.4.5
306	En desuso	Norma RFC 7231, sección 6.4.6
307	`Temporary Redirect` (Redireccionamiento temporal)	Norma RFC 7231, sección 6.4.7
308	`Permanent Redirect` (Redireccionamiento permanente)	Norma RFC 7538
309-399	`Unassigned` (No asignado)	
400	`Bad Request` (Petición incorrecta)	Norma RFC 7231, sección 6.5.1
401	`Unauthorized` (No autorizado)	Norma RFC7235, sección 3.1
402	`Payment Required` (Pago requerido)	Norma RFC 7231, sección 6.5.2
403	`Forbidden` (Prohibido)	Norma RFC 7231, sección 6.5.3
404	`Not Found` (No encontrado)	Norma RFC 7231, sección 6.5.4
405	`Method Not Allowed` (Método no permitido)	Norma RFC 7231, sección 6.5.5
406	`Not Acceptable` (No aceptable)	Norma RFC 7231, sección 6.5.6
407	`Proxy Authentication Required` (Autenticación de proxy requerida)	Norma RFC7235, sección 3.2
408	`Request Timeout` (Tiempo de espera agotado para la petición)	Norma RFC 7231, sección 6.5.7
409	`Conflict` (Conflicto)	Norma RFC 7231, sección 6.5.8
410	`Gone` (Ya no disponible)	Norma RFC 7231, sección 6.5.9
411	`Length Required` (Longitud requerida)	Norma RFC 7231, sección 6.5.10
412	`Precondition Failed` (Condición previa fallida)	Norma RFC 7232, sección 4.2, y Norma RFC 8144, sección 3.2
413	`Payload Too Large` (Carga útil demasiado grande)	Norma RFC 7231, sección 6.5.11
414	`URI Too Long` (URI demasiado largo)	Norma RFC 7231, sección 6.5.12
415	`Unsupported Media Type` (Tipo de medios no compatible)	Norma RFC 7231, sección 6.5.13, y Norma RFC7694, sección 3
416	`Range Not Satisfiable` (Rango no permitido)	Norma RFC 7233, sección 4.4
417	`Expectation Failed` (Expectativa fallida)	Norma RFC 7231, sección 6.5.14
418-420	`Unassigned` (No asignado)	
421	`Misdirected Request` (Petición mal dirigida)	Norma RFC 7540, sección 9.1.2
422	`Unprocessable Entity` (Entidad no procesable)	Norma RFC 4918
423	`Locked` (Bloqueado)	Norma RFC 4918
424	`Failed Dependency` (Dependencia fallida)	Norma RFC 4918

425	`Unassigned` (No asignado)	
426	`Upgrade Required` (Actualización requerida)	Norma RFC 7231, sección 6.5.15
427	`Unassigned` (No asignado)	
428	`Precondition Required` (Condición previa requerida)	Norma RFC 6585
429	`Too Many Requests` (Demasiadas peticiones)	Norma RFC 6585
430	`Unassigned` (No asignado)	
431	`Request Header Fields Too Large` (Campos del encabezado de petición demasiado largos)	Norma RFC 6585
432-450	`Unassigned` (No asignado)	
451	`Unavailable For Legal Reasons` (No disponible por razones legales)	Norma RFC 7725
452-499	`Unassigned` (No asignado)	
500	`Internal Server Error` (Error interno del servidor)	Norma RFC 7231, sección 6.6.1
501	`Not Implemented` (No implementado)	Norma RFC 7231, sección 6.6.2
502	`Bad Gateway` (Puerta de enlace incorrecta)	Norma RFC 7231, sección 6.6.3
503	`Service Unavailable` (Servicio no disponible)	Norma RFC 7231, sección 6.6.4
504	`Gateway Timeout` (Tiempo de espera agotado para la puerta de enlace)	Norma RFC 7231, sección 6.6.5
505	`HTTP Version Not Supported` (Versión HTTP no compatible)	Norma RFC 7231, sección 6.6.6
506	`Variant Also Negotiates` (Variante también negocia)	Documento RFC 2295
507	`Insufficient Storage` (Almacenamiento insuficiente)	Norma RFC 4918
508	`Loop Detected` (Bucle detectado)	Documento RFC 5842
509	`Unassigned` (No asignado)	
510	`Not Extended` (Extensiones requeridas)	Documento RFC 2774
511	`Network Authentication Required` (Autenticación de la red requerida)	Norma RFC 6585
512-599	`Unassigned` (No asignado)	

[Sigue el Anexo VI]

ANEXO VI

TÉRMINOS DE REPRESENTACIÓN

Versión 1.1

Revisión aprobada por el Comité de Normas Técnicas de la OMPI (CWS) en su décima sesión, celebrada el 25 de noviembre de 2022

Término	Definición	Tipo de datos
Amount (cuantía)	Valor monetario.	Número
Category (categoría)	División o subconjunto específicamente definido en un sistema de clasificación en el que todos los elementos comparten el mismo concepto de taxonomía.	Cadena
Code (código)	Combinación de uno o más números, letras o caracteres especiales, que se sustituye por un significado concreto. Representa valores finitos y predeterminados o formato libre.	Cadena
Date (fecha)	Punto concreto en el tiempo expresado mediante el año, el mes y el día.	Cadena
Directory (directorio)	Siempre precedido por `PATH`.	Cadena
Document (documento)	Un objeto grande de caracteres (CLOB) es un tipo de datos específico de casi todas las bases de datos. Básicamente, un CLOB es un puntero a un texto almacenado fuera del cuadro en un bloque dedicado. Se utiliza para documentos XML. Está compuesto por información textual del Registro Internacional de Marcas que se está intercambiando. Las etiquetas XML identifican los elementos de datos relacionados con dicha información. El equipo de desarrollo del apoyo a la tecnología y la innovación de Madrid puede definir el atributo XML_DOC como un CLOB, un puntero a los datos etiquetados almacenados fuera del cuadro en un bloque dedicado.	Cadena
Identifier (identificador)	Combinación de uno o más números enteros, letras o caracteres especiales que identifican de manera inequívoca una aplicación concreta de un objeto comercial, pero que puede no tener un significado fácilmente definible.	Cadena
Indicator (indicador)	Señal de presencia, ausencia o requisito de algo. Los valores recomendados son "`Y`", "`N`" y "`?`", si fuera necesario.	Booleano
Measure (medida)	Valor numérico que se determina midiendo un objeto y que se expresa mediante la unidad de medida especificada. `MeasureType` se utiliza para representar un tipo de magnitud física como la temperatura, la longitud, la velocidad, la anchura, el peso, el volumen o la latitud de un objeto. Más concretamente, `MeasureType` debe usarse para medir las propiedades intrínsecas o físicas de un objeto percibido como un conjunto.	Número
Name (nombre)	Designación de un objeto expresada mediante una palabra o una frase.	Cadena
`Number` (número)	Cadena de numerales o caracteres alfanuméricos que expresan una etiqueta, un valor, una cantidad o una identificación.	Número, cadena

Término	Definición	Tipo de datos
`Percent` (por ciento)	Número que representa una parte de un conjunto, que se dividirá por 100.	Número
`Quantity` (cantidad)	Número contado de unidades no monetarias que puede contener fracciones. `Quantity` se utiliza para representar un número de cosas contadas. `Quantity` debe utilizarse para contar o cuantificar las propiedades simples de un objeto percibido como un compuesto, una colección o un contenedor. `Quantity` debería siempre expresar un número de cosas contadas, y la propiedad será del tipo total, enviada, cargada o almacenada. `QuantityType` debería utilizarse para componentes que requieren información sobre la unidad; y `xsd:nonNegativeInteger` debería utilizarse para componentes contables que no requieren información sobre la unidad.	Número
`Rate` (índice)	Cantidad o cuantía medida con respecto a otra cantidad o cuantía.	Número
`Text` (texto)	Cadena de caracteres sin formato, por lo general en forma de palabras (incluye abreviaturas y comentarios).	Cadena
`Time` (tiempo)	Designación de un punto cronológico concreto en un período.	Fecha
`DateTime` (fecha y hora)	Fecha y hora en que se produce un evento.	Fecha
`URI`	Identificador uniforme de recursos que identifica la ubicación del archivo.	Cadena

[Sigue el Anexo VII]

ANEXO VII

PUBLICACIÓN DE LA GESTIÓN DEL CICLO DE VIDA DE LAS API

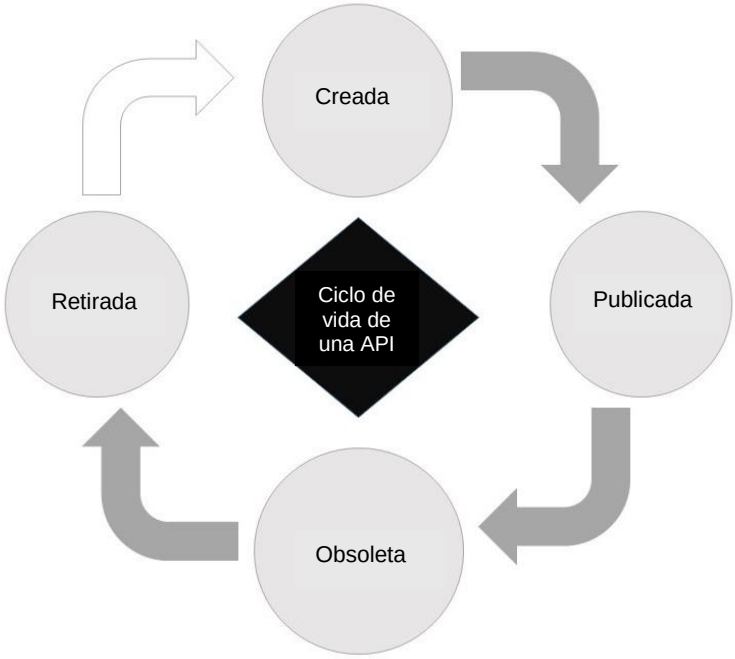
Versión 1.1

Revisión aprobada por el Comité de Normas Técnicas de la OMPI (CWS) en su décima sesión, celebrada el 25 de noviembre de 2022

1. En el presente Anexo se describe brevemente la gestión del ciclo de vida de las API y se indica la información más importante que las oficinas de PI deberían publicar en un documento normativo destinado a que los consumidores de API conozcan la mejor manera de utilizar la API.

2. La gestión del ciclo de vida de una API es un aspecto crítico de una estrategia de API, ya que establece el marco en que se desarrolla el ciclo de vida de la API desde su creación hasta su retirada. Resulta útil a nivel interno para los desarrolladores y equipos de operaciones y a nivel externo para los consumidores de API. Permite a los equipos de desarrollo interno crear una estructura y establecer expectativas para el desarrollo de la API, y a los equipos de operaciones conocer los requisitos de soporte. En cuanto a los consumidores de API, tanto a nivel interno como externo, les proporciona un contrato informal de expectativas para cuando utilicen una API determinada. Estos aspectos quedarán más claros cuando se describa cada una de las etapas del ciclo de vida.

3. Los ciclos de vida de API publicados consisten en procesos sencillos de 4 fases o en procesos más complejos de 10 o más fases. No obstante, en general, los ciclos de vida con más fases se consideran versiones detalladas de los ciclos de vida con menos fases. En el presente documento se aborda un proceso básico en el que la API pasa por 4 estados, suficiente para describir su ciclo de vida: creada -> publicada -> obsoleta -> retirada. Cualquier documento publicado sobre el ciclo de vida de una API debe incluir al menos una descripción de estas cuatro fases que son gestionadas por una oficina de PI.



Creada

4. La creación de una API se centra en su diseño, implementación y documentación. Lo más importante durante la fase de creación es tener en cuenta la finalidad de la API y la estructura general necesaria para que la API pueda funcionar lo mejor posible en el futuro. Lo ideal sería que la API se ajustara a una serie de normas internas y externas, como las recomendaciones formuladas en la presente norma. Si se va a monetizar la API, la estrategia de monetización debería definirse en esta fase.

Publicada

5. Después de crear una API es necesario publicarla. Debería utilizarse una estrategia estándar de desarrollo de versiones y se debería proporcionar documentación que incluya la especificación de la API y peticiones y respuestas de ejemplo (véanse [RSG-64] y [RSG-65]). Una vez publicada, la API puede ser consumida por las aplicaciones. Conviene señalar que durante la fase de publicación se pueden realizar correcciones y mejoras.

Obsoleta

6. En algún momento la API dejará de utilizarse, bien porque sea reemplazada por una versión más reciente o porque deje de ser de interés debido a algún factor externo o interno. Se deberá informar a los consumidores de la API y realizar los preparativos necesarios para retirar la API del catálogo. En esta fase se suelen corregir solo los errores más importantes de la API.

Retirada

7. En esta fase se desactiva la API, lo que debería incluir la imposibilidad de acceder a ella y la eliminación de la API de la plataforma correspondiente. Se debería considerar la posibilidad de prestar soporte extendido o si hay algún caso en que la retirada se pueda retrasar.

8. Las dos últimas fases, relativas a la obsolescencia y la retirada, son las principales que hay que documentar en la gestión del ciclo de vida. Es fundamental que los consumidores de las API conozcan las previsiones establecidas para la API cuando empiezan a utilizarla con el fin de evitar decepciones o problemas cuando se intente eliminar una API del catálogo. Esto debería incluir, por ejemplo, la gestión de las versiones mayores y menores y los plazos de notificación de los cambios. En un nivel más alto, suele haber dos enfoques para declarar como obsoleta o retirar una API: o bien se mantienen algunas versiones previamente establecidas, o bien se conservan las versiones antiguas durante un período de tiempo determinado. También puede utilizarse una combinación de ambos enfoques, pero en el documento publicado sobre el ciclo de vida debe indicarse claramente el número de versiones antiguas que se van a admitir o el período de tiempo que se mantendrán las versiones antiguas.

[Fin del Anexo VII y de la Norma]