**República Argentina - Poder Ejecutivo Nacional**
2017 - Año de las Energías Renovables

**Anexo**

**Número:** IF-2017-00277685-APN-SSGA#MM

CIUDAD DE BUENOS AIRES
Viernes 6 de Enero de 2017

**Referencia:** Anexo Pautas Técnicas de Interoperabilidad

---

**ANEXO**

**Pautas Técnicas de Interoperabilidad de Sistemas**

**I.- Introducción**

Las presentes Pautas Técnicas de Interoperabilidad tienen por objeto brindar un marco de intercambio de información entre las entidades y jurisdicciones que integran el Sector Público Nacional, en el marco del Decreto Nro. 1273/2016.

Las interfaces de programación de aplicaciones (APIs) exponen las funciones internas de un sistema al mundo exterior. Esto permite que los sistemas compartan información sin requerir que los desarrolladores tengan que compartir el código de las aplicaciones, ni entender las complejidades internas de cada sistema.

**II.- Alcances**

A los fines de facilitar el intercambio de información entre las entidades y jurisdicciones que componen el Sector Público Nacional, de acuerdo a lo dispuesto en el Decreto Nro. 1273 del 19 de diciembre de 2016, se establecen las siguientes Pautas de Interoperabilidad. Las entidades y jurisdicciones detalladas en el artículo 8 de la Ley Nro. 24.156 deberán intercambiar la información desarrollando APIs siguiendo los siguientes lineamientos.

Estas Pautas de Interoperabilidad son de aplicación para todas las entidades y jurisdicciones que integran la Administración Pública Nacional o Entidades de derecho Público vinculadas o dependientes de aquélla (en adelante, organizaciones) que participan en el intercambio de asientos registrales, ya sea para la prestación de servicios directos a los ciudadanos, como de cara al intercambio de información con otros órganos.

**III.- Arquitectura REST**

Las Pautas de Interoperabilidad se basan en la arquitectura REST. Dicha arquitectura especifica un conjunto de principios de arquitectura de Software que definen la interacción entre distintos componentes dentro de un sistema de hipermedia distribuido. En particular, se implementará arquitectura REST sobre el protocolo HTTP.

Los principios generales a los que deberán ajustarse son:

- Arquitectura cliente-servidor: Un cliente realiza peticiones a un servidor, el cual devuelve respuestas.
- Stateless: Cada petición que se realiza será independiente de la siguiente.
- Cacheable: Una vez realizada la primera petición, el resto podrán ser obtenidas del cache en caso de no haber sufrido cambios.
- Sistema por capas: Para toda petición originada de un cliente al servicio no será distinguible si está siendo dirigida directo al

servidor o a un sistema de chache o balanceador.

- Identificación del recurso: Se deberá indicar a través de una URI, la misma debe cumplir con el RFC 3986.
- Operaciones uniformes: Los métodos serán los especificados en HTTP (GET, PUT, POST, DELETE) provistos en el RFC 2616.
- Formato o representación del recurso: El tipo de contenido deberá ser del tipo JSON, el mismo se encuentra definido en el RFC 7159 https://tools.ietf.org/html/rfc7159

**IV.- Pautas en el diseño**

• Se debe crear una URI para cada recurso. Estos deben ser sustantivos, no verbos.

No se debe utilizar: http://www.ejemplo.com.ar/entidades/obtenerentidad?id=001

Se debe utilizar: http://www.ejemplo.com.ar/entidades/001

Use los verbos HTTP (GET, POST, PUT, DELETE) para funcionar en las colecciones y elementos.

- Ponga el número de versión en la URL, por ejemplo: http://ejemplo.gob.ar/v1.0/path/to/resource
- Especificar campos opcionales como una lista separada por coma.
- Para indicar el formato de respuesta utilizar el campo content-type del header siendo por defecto el formato JSON. Por ejemplo: XML: Content-Type: application/xml JSON: Content-Type: application/json; charset=utf-8
- El formato DEBE ser del tipo: api/v2.0/resource/{id}
- Utilizar sub recursos para indicar relaciones.

Si un recurso está relacionado a otro, se deberá representar como un sub recurso del mismo. Ejemplo: /organismos/{id del organismo}/empleados/{id_empleado}

No obstante es válido y muchas veces necesario que exista, siguiendo con el ejemplo, /empleados como top-level uri dentro de la API.

En el caso de que una representación incluya una lista de objetos relacionados debe incluir links a las URI de cada objeto. De otra forma sólo se puede navegar el árbol de relación teniendo información externa a los resultados que indique dónde ir a buscar cada recurso.

Una URi que apunta a un sub-recurso (ej, /organismos/{id_organismo}/empleados/ 123 puede devolver la representación del sub-recurso o simplemente utilizar un redirect http para indicar el top-level que lo devuelve (ej HTTP 307 moved temporarily)

- Categorizar los recursos de acuerdo a las acciones que los clientes pueden realizar sobre los mismos, por ejemplo una representación del recurso o modificación.
- Para las representaciones, se deben utilizar el verbo HTTP GET.
- Para las modificaciones, se deben utilizar los verbos HTTP POST, PUT y/o DELETE.
- Los request GET no deben tener efectos sobre el recurso al que apuntan. Sólo deben devolver su representación. Por lo tanto, invocar al recurso no debería tener como resultado modificarlo.
- Manejo de errores y respuestas usando códigos HTTP estándar

Usar los códigos de estado HTTP para indicar el resultado de las invocaciones a la API. Los más importantes son los siguientes:

- 200 ,201,204
- 304, 307
- 400,401,403,404,422
- 500

Las respuestas de errores DEBEN incluir los códigos de estados HTTP, mensaje para el desarrollador, mensaje para el usuario final, código de error interno, enlaces con documentación al respecto. Por ejemplo:

```
{
   "status" : 400,
   "developerMessage" : "Descripción clara del problema. Sugerencias de cómo resolver el problema.",
   "userMessage" : "Este es el mensaje para el usuario final.",
   "errorCode" : "[444444|exceptionName]",
   "moreInfo" : "http://www.ejemplo.gob.ar/developer/path/to/help/for/444444, http://drupal.org/node/444444",
}
```

- Mantener en minúsculas la URI.
- Usar UTF-8 y aceptar caracteres “especiales” como comillas, tildes, símbolos.
- Esconder las extensiones de las tecnologías utilizadas (.php, .asp, .jsp, etc).
- Se deberá utilizar como encoding UTF8, y deberá ser informado en la cabecera (Content-Type: application/json; charset=utf-8)
- Se deberán utilizar fechas según ISO 8601 en UTC

  Para solo fechas, el formato debe ser 2016-01-27.

  Para fechas completas, el formato debe ser 2016-01-27T10:00:00Z.

- Una API que retorna JSON DEBE usar el content type adecuado al tipo de datos y encoding que devuelve. El recomendado es:

  Content-Type: application/json; charset=utf-8

**Ejemplos válidos de URLs**

Lista de artículos:

```
GET http://www.ejemplo.gob/api/v1.0/articulos
```

Filtrando con query string:

```
GET http://www.ejemplo.gob/api/v1.0/articulos?year=2016&sort=desc
```

Un artículo en formato JSON:

```
GET http://www.ejemplo.gob/api/v1.0/articulos/1234
```

Todos los comentarios de un artículo en particular:

```
GET http://www.ejemplo.gob/api/v1.0/articulos/1234/comentarios
```

Especificar campos opcionales en una lista separada por coma:

```
GET http://www.ejemplo.gob/api/v1.0/articulos/1234?fields=title,body
```

Agregar un comentario a un artículo específico:

```
POST http://ejemplo.gob/api/v1.0/articulos/1234/comentarios
```

**Ejemplos NO válidos de URLs:**

- Sustantivos singulares:

  ```
  http://www.ejemplo.gob/api/v1.1/articulo
  http://www.ejemplo.gob/api/v1.1/articulo/1234
  ```

- Verbo en la URL:

  ```
  http://www.ejemplo.gob/api/v1.1/articulo/1234/create
  ```

- Filtro fuera del query string

  ```
  http://www.ejemplo.gob/api/v1.1/articulos/2016/desc
  ```

**V.- Verbos HTTP**

Los verbos HTTP, o métodos, se deben utilizar en el cumplimiento de sus definiciones de la norma HTTP/1.1 .

Este es un ejemplo de cómo deben ser los verbos HTTP para crear, leer, actualizar y eliminar las operaciones en un contexto particular:

| Método HTTP | POST | GET | PUT | DELETE |
|---|---|---|---|---|
| Operación | CREATE | READ | UPDATE | DELETE |
| /articulos | Crea nuevo artículo | Lista de artículos | Error | Elimina todos los artículos |
| /articulos/1234 | Error | Muestra el artículo 1234 | Si existe, actualiza el artículo; sino, devuelve error. | Borra 1234 |

**VI.- Soporte JSON**

- Las respuestas DEBEN ser objetos JSON (no arrays). Usar un array para retornar resultados limita la capacidad de incluir metadata sobre resultados, y limita la capacidad de las API's para agregar top-level keys en el futuro.

- Se puede incluir un parámetro especial para devolver los datos en otros formatos como XML, CSV u otro.
  GET http://www.ejemplo.gob/api/v1.0/articulos?format=csv
  GET http://www.ejemplo.gob/api/v1.0/articulos.csv

- No usar claves impredecibles. Realizar el parsing de una respuesta JSON donde las claves son impredecibles o cambiantes es complejo.

- Usa guión bajo para las claves. En formato JSON usar por ejemplo guión_bajo y no camelCase.

Más info en json.org

**VII.- Documentación**

Para documentar los Servicios WEB REST se debe incluir una introducción funcional que describa qué hace la API, y para cada servicio lo siguiente:

- Nombre
- URI

Ejemplo de uso incluyendo invocación con *curl* y la respuesta del ejemplo.

- Descripción de los parámetros obligatorios y opcionales
- Valores por default
- Códigos de error.

**Ejemplo de documentación**

**Listar Reparticiones**

Obtiene la lista de reparticiones

Uso: curl "http://ejemplo.gob.ar/api/v1/reparticiones" -H "Authorization: tokendeautenticacion"

El comando anterior devuelve una respuesta JSON como :


```
{
 items: [
        {
           "id": 1,
           "name": "reparticion1",
           "propiedad": "valor"
        },
        {
           "id": 2,
           "name": "reparticion2",
           "propiedad": "valor"
        },
]
}
```


Este endpoint devuelve las reparticiones según los parámetros de filtro

**HTTP Request**

GET http://example.com/api/kittens

Parámetros del request:

| Parametro | Default | Descripción |
|---|---|---|
| incluir_externos | false | Si está en true, la respuesta incluye reparticiones externas. |
| activas | true | Si está en false, la respuesta incuirá reparticiones que no están activan en el sistema |

Nota — Este servicio no requiere autenticación.

También se admitirá utilizar herramientas de documentación como:

- **SWAGGER**
- **MkDocs**
- **Aglio**

Documentar los ejemplos de la forma mas genérica posible, contemplando mas de un lenguaje de ser posible (por ejemplo Python y Java)

**VII.- Seguridad**

Con el objetivo de ofrecer escalabilidad, y generar servicios WEB stateless (sin estado) y aptos para aplicaciones móviles, se deberá implementar seguridad a través de los siguientes lineamientos:

**Autenticación Basada en Tokens**

Se implementará a partir de JSON WEB TOKEN (JWT) explicado en el RFC 7519 utilizando para asegurarlo los estándares JWS y JWE (RFC 7515 y 7516) utilizándo como mínimo el algoritmo HMAC

**Protocolo de comunicación segura**

La implementación de los servicios WEB RESTFul deberá ser implementada en https para ofrecer encriptación.

**VIII.- Versiones**

- No liberar una versión de una API sin su número de versión.
- Los números de versión deben abarcar dos niveles de versión: x.x
- Las versiones DEBEN ser enteros, no decimales, con el prefijo ‘v’.
- Dar soporte al menos una versión anterior a la actual.
- Ejemplos: Válido: v1.0, v2.1, v3.5 No válido: v-1.1, v1.2.5, 1.3.3

**IX.- Casos particulares**

Para algunos tipos de intercambio de información deben tenerse especial cuidado en la arquitectura de las APIs para que sean efectivas. Algunos casos particulares que pueden requerir complementar REST con otros protocolos son:

**a) Transferencias Masivas.** Cuando el cliente requiera tener una copia propia de un set de datos muy grande por motivos válidos (implementar un motor de búsqueda, hacer procesos de inteligencia de negocio, etc). Si la API no puede transmitir la información de forma eficiente o escalable (para el cliente como para el servidor) puede ser necesario implementar otros métodos.

**b) Notificaciones inmediatas.** Las APIs REST funcionan a pedido del cliente. En el caso de que el cliente deba tener las últimas modificaciones o novedades de forma inmediata se pueden implementar métodos tipo callback (si el servidor lo permite) o tecnologías de streaming de datos, publisher subscriber, etc. De ser posible deben mantenerse sobre HTTP (utilizando websockets, Server Sent Events,etc)

**c) Transacciones costosas**. Las transacciones que disparan operaciones muy costosas deben implementarse de forma asíncrona, evitando bloquear a los sistemas que las ejecutan.

Martelo María José
Subsecretaria
Subsecretaría de Gestión Administrativa
Ministerio de Modernización