2025

MARCO NORMATIVO DE IT

ES0903 - Estándar de Desarrollo API Agencia de Sistemas de Información

Noviembre 2025

Índice

1 Introducción

El presente documento tiene por finalidad definir los requisitos que deben cumplir todas las APIs del GCABA en su diseño, desarrollo e implementación, así como también establecer la interacción con los distintos actores involucrados en el proceso de desarrollo de dichas APIs. Corresponde mencionar que el presente Estándar de desarrollo de API se complementa con los Estándares de Desarrollo (ASI) y Seguridad (ASI), los cuáles se enfocan en los aspectos estructurales de las aplicaciones y su entorno de operación.

Se deja constancia que las definiciones utilizadas son aplicables para las APIs desarrolladas según especificaciones de las distintas reparticiones y agencias de la GCABA.

Con los siguientes estándares se busca homogeneizar la experiencia entre los diversos activos del Estado, para facilitar la comprensión y utilización, primando la usabilidad, accesibilidad y experiencia.

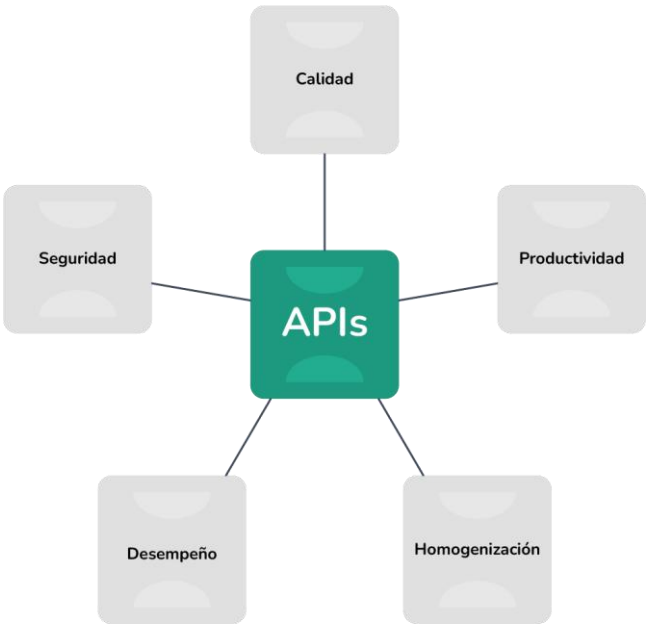
> ***IMPORTANTE**: Todos los sistemas que se desarrollen en el ámbito de los proyectos de GCABA deben respetar la totalidad de los criterios aquí descriptos. Si para algún proyecto en particular, se requiere un acuerdo diferente para alguno de los puntos detallados en este documento, el mismo tiene que quedar establecido por contrato, y este último deberá contar con la aprobación de la Agencia de Sistema de Información.*

2 Marco Normativo de TI

Toda solución de software deberá cumplir con lo expresado en el Marco Normativo de TI del GCABA, publicado en el boletín oficial del día 08-11-2013, Resolución 177-ASINF-2013, Resolución 239-ASINF/2014 y N° 12/ASINF/17. Dicha documentación se encuentra disponible en https://buenosaires.gob.ar/agencia-de-sistemas-de-informacion/estandares-de-la-agencia

3 Requisitos

Las APIs deben cumplir con los estándares establecidos en este documento a fin de lograr los siguientes objetivos:



4 Lineamientos

Estos lineamientos tienen como objetivo lograr una verdadera API RESTful.

Excepciones a tener en cuenta:

- Incluir el número de versión de la API en la URL. No se debe aceptar ninguna petición que no especifique el número de versión.
- No traducir al español lo que DEBE estar en inglés.

Armar una documentación detallada:
https://repositorio-ce-asi.buenosaires.gob.ar/

- Para el diseño y documentación de la misma utilizar herramientas como:
    - Swagger
    - Raml

5 Idioma

Bajo ningún concepto se debe traducir lo que DEBE estar en inglés. A continuación, se detallan algunos ejemplos:

Ejemplo válido:

- http://www.buenosaires.gob.ar/api/v1/articulos?year=2016&sort=desc

Ejemplo NO válido:

- http://www.buenosaires.gob.ar/api/v1/articulos?anio=2016&orden=desc

Esta restricción también incluye:

- **Nombres de los parámetros de consulta (query parameters):** Como *year, sort, filter, page, limit, etc.*
- **Valores específicos de ciertos parámetros:** Como *asc* o *desc* para ordenamiento, ciertos códigos de estado o categorías predefinidas.

6 URLs RESTful

Lineamientos generales:

- Una URL identifica un recurso.
- Las URLs DEBEN incluir sustantivos, no verbos.
- Usar sustantivos en plural solamente para consistencia (no sustantivos en singular).
- Usar los verbos HTTP apropiados (*GET, POST, PUT, PATCH, DELETE*) para operar en las colecciones y elementos.
- No se debe necesitar ir más allá de resource/identifier/resource.
- Indicar solo el número de versión mayor en la URL. Esta URL cambiará en el momento que el contrato cambie. Por ejemplo: http://ejemplo.gob.ar/v1/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 puede ser: *api/v2/resource/{id}*

Ejemplos válidos de URLs:

- Obtener la lista de artículos:
  - GET http://www.buenosaires.gob/api/v1/articulos
- Filtrar con query string:
  - GET http://www.buenosaires.gob/api/v1/articulos?year=2016&sort=desc
- Obtener un artículo en formato JSON:
  - GET http://www.buenosaires.gob.ar/api/v1/articulos/1234
- Agregar un comentario a un artículo específico:
  - POST http://www.buenosaires.gob.ar/api/v1/articulos/1234/comentarios

Ejemplos NO válidos de URLs:

- Sustantivos singulares:
  - http://www.buenosaires.gob.ar/articulo
- Filtro fuera del query string:
  - http://www.buenosaires.gob/articulos/2016/desc
- Formato de número de versión:
  - GET http://www.buenosaires.gob.ar/api/v1.0/articulos/1234

7 Verbos HTTP

Los verbos HTTP (o métodos), se deben utilizar en el cumplimiento de sus definiciones de la norma 1.1 / HTTP. Se comparte 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/PATCH	DELETE
Operación	CREATE	READ	UPDATE	DELETE
/artículos	Crea nuevo articulo	Lista de artículos	Error	Elimina todos los artículos
/artículos/1234	Error	Muestra articulo 1234	Si existe, actualiza el artículo; sino devuelve error.	Borra 1234

8 Soporte JSON

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

No usar claves impredecibles: realizar el parsing de una respuesta JSON donde las claves son impredecibles es difícil y genera malestar a los clientes.

Usar *guión_bajo* para las claves: diferentes lenguajes usan diferentes convenciones. JSON usa *guión_bajo*, no *camelCase*.

Rever estándar Json**

Más info en json.org

Respuestas

- No incluir valores en las claves.
- La metadata solamente debe contener propiedades directas a la respuesta, no propiedades relacionadas a la información de la respuesta.

Ejemplo válido
Sin valores en claves:

```
"tags": [
{"id": "125", "name": "Ciudadano"},
{"id": "834", "name": "Servicios"}
],
```

Ejemplo NO válido
Con valores en claves:

```
"tags": [
  {"125": "Ciudadano"},
  {"834": "Servicios"}
],
```

9 Formato de fecha

Usar ISO 8601, en UTC.

- Para utilizar solo fechas, el formato debe ser: **2016-01-27**.
- Para fechas completas, el formato debe ser: **2016-01-27T10:00:00Z.**
    - 2016-01-27T10:00:00Z
        - año, mes, día
        - hora, minutos, segundos
        - UTC

Puede obtenerse más información en The 5 laws of API dates and times

10 Manejo de errores

Las respuestas de errores DEBEN incluir los códigos de estado HTTP, mensaje para el desarrollador, mensaje para el usuario final, código de error interno y enlaces con más información para los desarrolladores.

Por ejemplo:

```
{
    "status": 400,
    "developerMessage": "Detallar una descripción clara del
problema. Proveer a los desarrolladores sugerencias de cómo
resolver sus problemas.",
    "userMessage": "Este es el mensaje para el usuario final.",
    "errorCode": "444444",
```

```
    "moreInfo":
"http://www.ejemplo.gob.ar/developer/path/to/help/for/444444,
http://drupal.org/node/444444",
}
```

Usar estos 3 simples códigos de respuesta indicando (1) éxito, (2) fallo debido a un problema del cliente, (3) fallo debido a un problema del servidor:

```
200 - OK
400 - Bad Request
500 - Internal Server Error
```

10.1 Listado de códigos de estado

1×× Informational

100 Continue
101 Switching Protocols
102 Processing

2×× Success

200 OK
201 Created
202 Accepted
203 Non-authoritative Information
204 No Content
205 Reset Content
206 Partial Content
207 Multi-Status
208 Already Reported
226 IM Used

3×× Redirection

300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
305 Use Proxy
307 Temporary Redirect
308 Permanent Redirect

4×× Client Error

400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed

406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 Conflict
411 Length Required
412 Precondition Failed
413 Payload Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
418 I'm a teapot
421 Misdirected Request
422 Unprocessable Entity
423 Locked
424 Failed Dependency
426 Upgrade Required
428 Precondition Required
429 Too Many Requests
431 Request Header Fields Too Large
444 Connection Closed Without Response
451 Unavailable For Legal Reasons
499 Client Closed Request

5×× Server Error

500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported
506 Variant Also Negotiates
507 Insufficient Storage
508 Loop Detected
510 Not Extended
511 Network Authentication Required
599 Network Connect Timeout Error

11 UTF-8

Se debe utilizar UTF-8

Esperar caracteres acentuados o comillas en la salida de la API, aun cuando no se esperen.

Una API debe informar a los clientes de esperar UTF-8 mediante la inclusión de una notación de caracteres en la cabecera *Content-Type* para las respuestas.

Una API que retorna JSON DEBE usar: *Content-Type: application/json; charset=utf-8*

12 Versiones

- Nunca liberar la versión de una API sin su correspondiente número.
- Los números de versión deben abarcar tres niveles de versión: X.Y.Z
    a. **X= Versión Mayor** debe ser incrementada por cualquier cambio no compatible con la versión anterior de la API.
    b. **Y= Versión Menor** debe ser incrementado si se introduce nueva funcionalidad compatible con la versión anterior.
    c. **Z= Versión Patch** debe ser incrementado cuando se introducen solo arreglos compatibles con la versión anterior.
- Los números correspondientes a las versiones mayor, menor y patch son enteros no negativos y se incrementan en 1, comenzando en 1.
- Las versiones deben ser expresadas en números enteros, no decimales, con el prefijo 'v'.
- Dar soporte al menos hasta una versión anterior a la actual.
- Ejemplos:
    - Válido: v1.0.0, v2.1.0, v3.5.0
    - No válido: v-1.1.0

13 Límite de registros

- Si el límite no está especificado, retornar resultados con un valor por defecto.
- Por ejemplo, para obtener registros de 51 a 75, hacer lo siguiente:
    - http://APIrenaper.buenosaires.gob.ar/dni?limit=25&offset=50
    - offset=50 significa, 'evitar los primeros 50 registros'
    - limit=25 significa, 'retornar un máximo de 25 registros'

La información sobre los límites de registros y totales disponibles DEBEN ser incluidos en la respuesta. Por ejemplo:

```
{
    "metadata": {
        "resultset": {
            "count": 225874,
            "offset": 25,
            "limit": 25
        }
    },
    "results": []
}
```

Ejemplos de Peticiones y Respuestas

- GET /artículos
- GET /articulos/[id]
- POST /articulos/[id]/comentarios

GET /articulos

Ejemplo: http://www.buenosaires.gob.ar/api/v1/articulos

Respuesta:

```
{
    "metadata": {
        "resultset": {
            "count": 123,
            "offset": 0,
            "limit": 10
        }
    },
    "results": [
        {
            "userId": 1,
            "id": 1,
            "title": "sunt aut facere repellat occaecati",
            "body": "quia et suscipit recusandae expedita."
        },
        {
            "userId": 2,
            "id": 2,
            "title": "qui est esse",
            "body": "est rerum tempore vitae sequi nihil dolor."
        }
    ]
}
```

GET /artículos/[id]

Ejemplo: http://www.buenosaires.gob.ar/api/v1/articulos/[id]

Respuesta:

```
{
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat occaeati excepturi optio",
    "body": "quia et suscipit suscipit consequuntur expedita."
}
```

POST /articulos/[id]/comentarios

Ejemplo: Crear – POST http:// www.buenosaires.gob.ar /api/v1/articulos/[id]/comentarios

Cuerpo de la solicitud:

```
{
    "postId": 1,
    "id": 1,
    "name": "id labore ex et quam laborum",
    "email": "pedro@ejemplo.com",
    "body": "laudantium enim quasi est quidem ipsam eos."
}
```

14 Datos de prueba

Cada recurso debe aceptar un parámetro 'mock' en el servidor de prueba. Pasando este parámetro, se debe devolver una respuesta de datos simulada (sin pasar por el backend).

La implementación de esta función en la primera etapa del desarrollo asegura que la API exhibirá un comportamiento coherente.

Nota: Si el parámetro 'mock' está incluido en una solicitud para el entorno de producción, debe mostrar un error.

15 Seguridad

Cualquier API que se desarrolle debe usar HTTPS encryption (TLS/SSL). HTTPS provee:

- **Seguridad**. El contenido de las peticiones está encriptado a través de Internet.
- **Autenticidad.** Una garantía más fuerte de que un cliente se comunica con la API real.
- **Privacidad.** Privacidad mejorada para las aplicaciones y usuarios que usan la API. Las cabeceras HTTP y los parámetros query string (entre otras cosas) serán encriptadas.
- **Compatibilidad.** Más amplia compatibilidad del lado del cliente. Para solicitudes CORS a la API para trabajar en los sitios web HTTPS - para no ser bloqueado en forma de contenido mixto - esas peticiones deben ser a través de HTTPS.

HTTPS debe estar configurado aplicando las mejores prácticas, incluyendo cifrado que soporte forward secrecy y Seguridad de transporte HTTP estricta (HTTP Strict Transport Security).

Para APIs existentes que corren sobre HTTP, el primer paso es agregar soporte HTTPS y actualizar la documentación aclarando que es la configuración por defecto, usándolo en los ejemplos, etc.

Luego, evaluar la posibilidad de deshabilitar o redireccionar a peticiones HTTP.

16 Claves API

client_id y client_secret

OpenID

app_id y app-key

Es importante que las APIs tengan la forma de poder identificar qué aplicación quiere acceder a los recursos. Para esto, se utiliza una clave que va junto con el request.

Ejemplo API's Infraciones:

https://servicios.gcba.gob.ar/SAI-InfraccionesApi/api/actas-varios/600200001/resultado

header client_id: abc88c4d5b212345667897bd5ab094

header client_secret: abce4191234567898452a146a

Resultado de esta identificación:

- Previene peticiones de usuarios anónimos.
- Previene que datos sensibles sean expuestos.
- Se puede aplicar *rate limiting* dependiendo el cliente.
- Previene vulneraciones de la seguridad de los datos

17 CORS

Para que los clientes puedan usar una API desde el front de una aplicación, la API DEBE tener habilitado CORS.

Para el más simple y común caso de uso, donde toda la API entera deba ser accesible desde el navegador, habilitar CORS es tan simple como incluir esta cabecera HTTP en todas las respuestas:

Access-Control-Allow-Origin: *

Access-Control-Allow-Origin: http://example.com

Access-Control-Allow-Methods: PUT, POST, DELETE

- **Access-Control-Allow-Origin:** ¿qué origen está permitido?
- **Access-Control-Allow-Credentials:** ¿también se aceptan solicitudes cuando el modo de credenciales es incluir (include)?
- **Access-Control-Allow-Headers:** ¿qué cabeceras pueden utilizarse?
- **Access-Control-Allow-Methods:** ¿qué métodos de petición HTTP están permitidos?
- **Access-Control-Expose-Headers:** ¿qué cabeceras pueden mostrarse?
- **Access-Control-Max-Age:** ¿cuándo pierde su validez la solicitud preflight?
- **Access-Control-Request-Headers:** ¿qué header HTTP se indica en la solicitud preflight?
- **Access-Control-Request-Method:** ¿qué método de petición HTTP se indica en la solicitud preflight?

Esto tiene soporte por todos los navegadores modernos y funciona simplemente en todos los clientes JavaScript, como jQuery.

Para una configuración más avanzada, ver la especificación de W3C o la guía de Mozilla.

Por ese motivo el CORS ofrece una solución intermedia, permitiendo hacer excepciones a la prohibición en aquellas situaciones en que las solicitudes de origen cruzado son expresamente requeridas. No obstante, se corre el riesgo de que los administradores web se aprovechen de las wildcards por comodidad, haciendo que la protección de la SOP sea en vano. Por eso, es importante utilizar el CORS solo en casos especiales y configurarlo de la manera más restrictiva posible.

18 Documentación

Todas las APIs deben contar con una documentación clara, accesible, actualizada y centrada en el consumidor.

Requisitos mínimos de la documentación:

- **Nombre de la API**
- **Descripción:** descripción general de su propósito.
- **Versión actual**: de versiones anteriores (cuando aplique).
- **Contacto:** mail@xxxxxx.com
- **Environment:** para cada ambiente (dev, test, prod).
- **Autenticación requerida**, incluyendo mecanismos utilizados (token, OAuth, API key, etc). En caso de utilizar tokens dinámicos, los mismos deben tener una vigencia no mayor a 15 minutos.
- **Listado completo de endpoints** con la siguiente información por cada uno:
    - Método HTTP (GET, POST, etc.)
    - Ruta (/usuarios/{id})
    - Descripción funcional
    - Parámetros de entrada (query, path, header, body)
    - Ejemplo de solicitud (request)

- Ejemplo de respuesta (response), incluyendo posibles códigos de estado (200, 400, 401, etc.)
- Definiciones de errores

- **Esquemas de datos (models)** usados por la API, incluyendo:
  - Campos requeridos
  - Tipos de datos
  - Validaciones
  - Relación entre objetos

Herramientas sugeridas para la documentación:

- **Swagger**

- **RAML**

```
#%RAML 1.0
title: Nombre de tu API
version: v1.0
description: |
  Descripción general del propósito de tu API.
  Aquí puedes explicar qué hace la API, a quién está dirigida y cualquier otra información relevante.

baseUri: |
  {environment}
baseUriParameters:
  environment:
    description: Entorno base de la API.
    type: string
    enum:
      - http://localhost:8080/dev
      - https://test.example.com/api/v1
      - https://api.example.com/v1
    example: https://api.example.com/v1

securitySchemes:
  - bearerAuth:
      type: OAuth 2.0
      description: |
        Esquema de seguridad para autenticación mediante Bearer Token (JWT).
      settings:
        authorizationUri: https://example.com/oauth/authorize
        accessTokenUri: https://example.com/oauth/token
        authorizationGrants: [ authorization_code, implicit ]
        scopes:
          - read: Acceso de lectura
          - write: Acceso de escritura
  - apiKeyAuth:
      type: API Key
      description: |
```

19 Referencias

- White House Web API Standards
- W3C: REST
- 18F API Standards
- Best Practices for Designing a Pragmatic RESTful API