




10 MEJORES PRÁCTICAS PARA EL DISEÑO DE APIS REST



UTILIZA URLS SEMÁNTICAS



Las URLs deben ser intuitivas y representar claramente el recurso que manejan. Evita utilizar verbos en las URLs y opta por sustantivos.

- Incorrecto: `/obtenerProductos``
- Correcto: `/productos``

Las URLs semánticas no solo hacen que la API sea más fácil de entender, sino que también ayudan en la documentación automática y en las pruebas.

MÉTODOS HTTP CORRECTOS

Cada operación en tu API debe corresponderse con un método HTTP adecuado:


- GET: Para recuperar información.
Ejemplo: 'GET /productos' devuelve todos los productos.
- POST: Para crear un nuevo recurso.
Ejemplo: 'POST /producto' para agregar un nuevo producto.
- PUT: Para actualizar un recurso existente, o crearlo si no existe. Ejemplo: 'PUT /productos/123' actualiza el producto con ID 123.
- PATCH: Para actualizar solo una parte de los recursos. Ejemplo: 'PATCH /productos/123' actualiza algunos atributos del producto con ID 123.
- DELETE: Para eliminar un recurso.
Ejemplo: 'DELETE /productos/123' elimina el producto con ID 123.

Utilizar los métodos adecuados asegura que tu API sea predecible y siga estándares, facilitando su uso



CÓDIGOS DE ESTADO HTTP


Los códigos de estado HTTP informan al cliente sobre el resultado de la solicitud:

- 
- 200 OK: Solicitud exitosa.
 - 201 Created: Un nuevo recurso ha sido creado.
 - 400 Bad Request: La solicitud contiene un error de sintaxis.
 - 401 Unauthorized: El cliente no está autenticado.
 - 404 Not Found: No se encuentra el recurso solicitado.
 - 500 Internal Server Error: Error en el servidor.

Estos serían los 6 códigos de estado principales, que proporcionan información precisa y ayuda en la depuración y manejo de errores.



DOCUMENTACIÓN



Una buena documentación es esencial para que otros desarrolladores (y tú en el futuro) podáis entender cómo interactuar con tu API. Para ello puedes utilizar herramientas como Swagger. A la hora de documentar, estos serían los puntos clave:


- Documenta todos los endpoints con ejemplos.
- Describe los parámetros de entrada y salida.
- Proporciona ejemplos de peticiones y respuestas.
- Incluye los códigos de estado que pueden ser devueltos.

Esto reduce considerablemente la curva de aprendizaje y facilita la utilización e integración de tu API.



VALIDACIÓN DE DATOS

Para evitar errores y asegurar la integridad de los datos, valida las entradas usando las librerías adecuadas. En Java, por ejemplo, puedes usar 'Bean Validation' con anotaciones tipo:



```
@NotNull  
@Size(min = 2, max = 30)  
private String nombre;
```

- Validación en el Cliente: Asegura que los datos ingresados sean correctos antes de enviarlos al servidor.
- Validación en el Servidor: Revalida los datos recibidos, para asegurar que sean seguros y correctos.

La validación en ambos extremos protege tu API de datos corruptos y ataques maliciosos



VERSIONADO DE LA API

Las APIs cambian con el tiempo, y es importante permitir a los clientes seguir utilizando versiones anteriores mientras migran a la nueva versión.



Una forma común de versionar es incluir la versión en la URL:

- Ejemplo: `'/v1/productos'`, `'/v2/productos'`

Esto permite realizar cambios significativos en la API sin afectar a los clientes que utilizan versiones anteriores.

PAGINACIÓN Y FILTROS



Cuando tienes grandes colecciones de recursos, sirve solo una parte a la vez para mejorar el rendimiento y la experiencia de usuario:


- **Paginación:** Utiliza parámetros como 'page' y 'limit' para dividir los datos: 'GET /productos?page=1&limit=50'.
- **Filtros:** Permite a los usuarios filtrar los resultados, por ejemplo, 'GET /productos?status=activo'.

Implementar paginación y filtros es crucial para la escalabilidad y mejorar significativamente la experiencia del usuario.

RESPUESTA CONSISTENTE




La consistencia en las respuestas es un aspecto fundamental para asegurar que los desarrolladores puedan interactuar con tu API de manera sencilla y predecible. Mejores prácticas para una respuesta consistente

- 
- Estructura de respuesta uniforme: Todas las respuestas, ya sean de éxito o de error, deben seguir una estructura uniforme. Por ejemplo, en lugar de devolver directamente los datos del recurso, encapsula la información en una estructura común: `{"data": { "id": 1, "nombre": "Producto 1", "precio": 100.0}}`
 - Metadatos adicionales: Incluye metadatos útiles, como el número total de registros cuando se devuelve una lista de recursos
 - Formato de respuesta JSON: Asegúrate de que todas las respuestas sigan un formato común, como JSON, que es el estándar más utilizado
 - Evita cambiar el formato de la respuesta entre endpoints: Mantén una estructura coherente en todas las rutas y acciones.
 - Claridad y simplicidad: La respuesta debe ser clara y fácil de interpretar. Evita añadir información innecesaria o excesiva

CONTROL DE ERRORES




Mejores prácticas para el manejo de errores:

- 
- Códigos de estado HTTP significativos:
 - 400 Bad Request: Datos de entrada inválidos.
 - 401 Unauthorized: No autenticado.
 - 403 Forbidden: Sin autorización para acceder al recurso.
 - 404 Not Found: El recurso no existe.
 - 500 Internal Server Error: Error del servidor.
 - Mensajes de error detallados: Ofrece mensajes que describan claramente el problema, pero sin revelar información sensible
 - Estructura de respuesta clara: Utiliza una estructura predecible para los errores, como { "error": { "message": "...", "code": ... } }
 - Logueo y monitoreo de errores: Implementa un sistema de logueo que registre los errores en los sistemas internos para que puedan ser rastreados y solucionados de manera eficiente.

AUTENTICACIÓN Y AUTORIZACIÓN



Mejores prácticas :

- 
- OAuth 2.0: Permite delegar la autenticación a través de un servidor confiable (como Google o Facebook) sin compartir credenciales directamente. También es ideal para aplicaciones que requieren autenticación basada en tokens.
 - Token JWT (JSON Web Tokens): OAuth a menudo usa tokens JWT para autorizar peticiones. Estos tokens son compactos, auto-contenidos y firmados digitalmente, lo que permite verificar la autenticidad sin consultar una base de datos.
 - Scopes: Son permisos específicos que definen qué acciones puede realizar el token.
 - Roles: Asociados a un grupo de permisos más amplios como "admin", "editor", "viewer".
 - HTTPS: Siempre utiliza HTTPS para encriptar las comunicaciones entre el cliente y el servidor.
 - Expiración de Tokens: Configura tiempos de expiración para los tokens, de modo que los usuarios deban autenticarse nuevamente después de un período.



¿TE HA RESULTADO ÚTIL?

¡No olvides guardarlo para más adelante, darle like,
o compartir para que a más gente pueda serle útil!

