

# Manifiesto de Codificación para Desarrollos de Software - APIs REST

## -Principios Generales

### 1. Simplicidad y Claridad

El código debe ser fácil de entender, sin sobrecomplicar los procesos. Evitar soluciones complejas cuando una más sencilla pueda ser efectiva.

### 1. Consistencia

El código debe seguir convenciones y patrones consistentes para que sea más fácil de mantener y comprender

por otros desarrolladores. Las herramientas como linters y formateadores automáticos son recomendables para mantener la consistencia.

### 1. Modularidad y Reusabilidad

El código debe ser estructurado en módulos pequeños, reutilizables y fáciles de probar. Las funcionalidades

comunes deben ser encapsuladas en servicios o bibliotecas reutilizables.

## - Estructura de la API

### 1. Versionado de la API

Todas las APIs deben tener un esquema de versionado claro. Se recomienda incluir la versión en la URL, como

GET /api/v1/usuarios.

### - Uso de Métodos HTTP

Las APIs deben usar los métodos HTTP estándar de manera coherente:

1. GET: Para obtener datos.
2. POST: Para crear recursos.
3. PUT: Para actualizar recursos.
4. DELETE: Para eliminar recursos.

### 2.3 Codificación de Respuestas HTTP Las respuestas deben usar códigos de estado HTTP adecuados:

200 OK: Solicitud exitosa. 201 Created: Recurso creado. 204 No Content: Solicitud exitosa sin contenido. 400 Bad Request: Solicitud mal formada. 401 Unauthorized: Falta de autenticación. 403 Forbidden: Prohibido. 404 Not Found: Recurso no encontrado. 500 Internal Server Error: Error del servidor. 2.4 Formato de Datos Las APIs deben usar JSON como formato de intercambio de datos. Si se requiere otro formato, debe ser explícitamente documentado.

3. Diseño de Endpoints 3.1 Convenciones de Nombres Los nombres de los endpoints deben ser coherentes y reflejar los recursos que representan:

/usuarios (en plural para listas de recursos). /usuarios/{id} (para acceder a un recurso específico). 3.2 Uso de Parámetros en la URL y Query Parameters Utiliza parámetros de ruta (/usuarios/{id}) para recursos individuales y parámetros de consulta (/usuarios?edad=30) para filtrar, ordenar y paginar los resultados.

3.3 Manejo de Errores Las APIs deben devolver errores de manera estandarizada. La respuesta debe incluir un código de estado HTTP adecuado y un cuerpo de respuesta con un mensaje descriptivo del error:

json Copiar código {

```
"error": "Usuario no encontrado",  
"message": "No se ha encontrado un usuario con el ID proporcionado",  
"status": 404
```

} 4. Seguridad 4.1 Autenticación y Autorización Las APIs deben requerir autenticación a través de métodos seguros, como JWT (JSON Web Tokens) o OAuth 2.0. No deben almacenarse credenciales en texto claro.

4.2 CORS (Cross-Origin Resource Sharing) Las políticas de CORS deben ser configuradas para permitir el acceso seguro desde los orígenes que se necesiten, y se deben evitar configuraciones demasiado abiertas.

4.3 Protección contra Inyecciones Se deben aplicar medidas de seguridad como la validación de entrada, uso de consultas parametrizadas y escaneo de vulnerabilidades de inyección SQL, XSS, CSRF, entre otros.

5. Documentación 5.1 Documentación Automática de la API Las APIs deben estar documentadas usando herramientas como Swagger/OpenAPI para generar documentación interactiva y siempre actualizada. Los endpoints, parámetros y respuestas deben estar descritos claramente.

5.2 Comentarios en el Código Aunque el código debe ser claro, los comentarios deben usarse para explicar “por qué” se hace algo, no “qué” se hace, ya que el segundo debe ser evidente en un código bien escrito.

6. Pruebas 6.1 Cobertura de Pruebas Es fundamental que las APIs tengan una buena cobertura de pruebas unitarias, de integración y de aceptación. Las pruebas deben ser automatizadas siempre que sea posible.

6.2 Pruebas de Seguridad Las APIs deben ser sometidas a pruebas de seguridad regulares, como pruebas de penetración y revisiones de vulnerabilidades.

7. Manejo de Logs 7.1 Uso de Logs Las APIs deben registrar eventos importantes para facilitar la depuración y la auditoría. Los logs deben contener información relevante pero no deben incluir información sensible como contraseñas o tokens.

7.2 Niveles de Logs Utilizar niveles adecuados para los logs, como:

DEBUG: Información detallada para depuración. INFO: Información sobre el flujo normal de la

aplicación. WARN: Advertencias de posibles problemas. ERROR: Errores que ocurren durante la ejecución. 8. Desempeño 8.1 Optimización de Consultas Se deben evitar consultas ineficientes que puedan causar cuellos de botella. El uso de índices en las bases de datos y la paginación en respuestas con grandes volúmenes de datos es fundamental.

8.2 Escalabilidad El diseño de la API debe permitir una fácil escalabilidad, tanto horizontal como vertical, utilizando técnicas como la cacheización de resultados y la distribución de carga.

[←Regresar](#)

From:  
<http://wiki.adacsc.co/> - Wiki

Permanent link:  
<http://wiki.adacsc.co/doku.php?id=ada:howto:sicoferp:factory:new-migracion-sicoferp:manifiestoback&rev=1733776029>

Last update: **2024/12/09 20:27**

