Programación IV

Profesora: Cynthia Estrada
Fecha: 20/10
Tema: *Documentación API*

Integrantes:

- Díaz Rossini Juan José
- Gallo Genaro
- Navarro Victor Leandro

🧩 Documentación API

---

1. ¿Qué es una documentación API?

La **documentación de una API** es una guía técnica que explica **cómo interactuar correctamente con una API (Interfaz de Programación de Aplicaciones)**.
En otras palabras, es como el **manual de instrucciones** que permite a otros desarrolladores entender cómo usar los servicios que ofrece esa API, sin necesidad de revisar o modificar su código interno.

Esta documentación suele incluir:

- Una descripción general sobre qué hace la API.
- Los distintos **endpoints** o rutas disponibles.
- Qué **métodos HTTP** usa (GET, POST, PUT, DELETE, etc.).
- Qué **parámetros** o datos debe recibir.
- Qué **respuestas** devuelve (por ejemplo, en formato JSON).
- Ejemplos de uso y posibles **errores**.

Ejemplo:
Si la API tiene un endpoint `/usuarios`, la documentación debe indicar:

- Qué método se usa (`GET /usuarios` para listar, `POST /usuarios` para crear, etc.).
- Qué datos se deben enviar (por ejemplo, nombre, correo).
- Qué datos devuelve (como id, nombre, email).
  De esa manera, cualquier programador puede utilizar la API sin necesidad de conocer su código

fuente.

---

2. ¿Quién escribe la documentación API?

Generalmente, la documentación de una API puede ser elaborada por distintos perfiles dentro de un equipo:

- **Desarrolladores:**
  Son quienes crean la API y conocen su lógica interna, por lo que pueden detallar su funcionamiento técnico con precisión.

- **Redactores técnicos (Technical Writers):**
  Se especializan en escribir de forma clara, estructurada y comprensible para todo tipo de lector técnico. Su tarea es transformar la jerga del programador en un texto didáctico.

- **Equipos de QA o soporte técnico:**
  Aportan ejemplos reales, pruebas y casos comunes de uso. Gracias a su experiencia con usuarios, ayudan a detectar posibles confusiones o errores frecuentes.

👉 En los equipos más organizados, la documentación se crea de forma **colaborativa**, combinando el conocimiento técnico del desarrollador con la claridad comunicativa del redactor.

---

3. Función detallada de cada parte de una documentación API

Una buena documentación API está dividida en secciones, y cada una cumple una función específica:

- **Introducción:**
  Explica qué es la API, cuál es su propósito y en qué contexto se usa (por ejemplo: API para gestionar usuarios, productos o pagos).

- **Autenticación:**
  Describe cómo conectarse de forma segura a la API, ya sea mediante **tokens, claves API, OAuth** u otro método de validación.

- **Endpoints o recursos:**
  Presenta todas las rutas disponibles (por ejemplo `/usuarios`, `/productos`, `/pedidos`) indicando qué acciones se pueden realizar en cada una (listar, crear, actualizar, eliminar).

- **Parámetros y respuestas:**
  Detalla qué datos necesita cada endpoint, qué tipo de datos son (string, number, boolean) y qué información devuelve la API al cliente.

- **Ejemplos de uso:**
  Muestra ejemplos reales de peticiones (en cURL, JavaScript, Python, etc.), lo que facilita entender la implementación.

- **Códigos de error:**
  Informa qué puede salir mal (por ejemplo, “404 Not Found” si el recurso no existe, o “401 Unauthorized” si el token no es válido).

- **Versionado:**
  Indica la versión actual de la API (v1, v2, etc.) y las diferencias o mejoras entre versiones.

---

4. ¿Por qué es importante documentar una API?

La documentación es **fundamental** porque sin ella, los desarrolladores no podrían entender cómo interactuar correctamente con la API.
Una API sin documentación es como una **máquina sin manual de uso**: puede funcionar perfectamente, pero nadie sabrá cómo operarla.

Justificación:

- **Reduce el tiempo de aprendizaje:** Los nuevos desarrolladores pueden integrarse rápidamente al proyecto.

- **Facilita la integración:** Otros sistemas o aplicaciones externas pueden conectarse fácilmente.

- **Aumenta la mantenibilidad:** Es más fácil actualizar o modificar la API sin romper funcionalidades existentes.

- **Evita errores:** Al estar todo explicado, se minimizan los malentendidos entre desarrolladores.

- **Mejora la calidad del software:** Todo el equipo sabe exactamente cómo deben comportarse los endpoints y qué resultados esperar.

---

5. Ventajas de contar con documentación API

Tener una buena documentación trae numerosos beneficios, tanto para el equipo de desarrollo como para los usuarios externos:

- **Rapidez de integración:** Los programadores entienden rápidamente cómo consumir la API.

- **Menor carga de soporte técnico:** Si está bien explicada, los usuarios necesitan menos asistencia.

- **Estandarización:** Todos siguen las mismas convenciones y formatos, evitando confusiones.

- **Facilita el mantenimiento:** Los cambios futuros pueden realizarse de forma más segura.

- **Aumenta la adopción:** Si es una API pública, una buena documentación atrae más desarrolladores y empresas interesadas en usarla.

---

6. Elementos esenciales que debe incluir una documentación API

1. **Descripción general:** Qué hace la API y su objetivo principal.
2. **Guía de autenticación:** Cómo acceder (tokens, claves, permisos).
3. **Lista de endpoints:** Incluye la URL completa, el método HTTP (GET, POST, PUT, DELETE), su descripción y ejemplos de uso.
4. **Ejemplos de solicitudes y respuestas:** Muestran exactamente cómo enviar datos y qué esperar a cambio (en formato JSON o XML).
5. **Códigos de estado HTTP:** Explicación clara de cada código (200 OK, 404 Not Found, 500 Server Error, etc.).
6. **Errores comunes y soluciones:** Ayuda a los desarrolladores a resolver fallas rápidamente.
7. **Información de versión:** Indica si hay actualizaciones, mejoras o endpoints obsoletos.
8. **Casos prácticos:** Ejemplos aplicados a situaciones reales o simulaciones de uso.

---

7. Cómo estructurar correctamente una documentación API

La estructura debe seguir un **orden lógico y progresivo**, de lo general a lo específico:

1. **Introducción general** – Contexto, propósito y visión general.
2. **Requisitos previos** – Librerías, dependencias o entornos necesarios.
3. **Autenticación y seguridad** – Cómo acceder de manera segura.
4. **Lista de endpoints** – Detallando:
    - Método HTTP (GET, POST, PUT, DELETE).
    - URL del recurso.
    - Parámetros requeridos y opcionales.
    - Ejemplo de solicitud.
    - Ejemplo de respuesta.
    - Posibles errores.
5. **Guía paso a paso** – Cómo usar la API en un flujo real.
6. **Ejemplos de código** – Implementaciones en distintos lenguajes.

7. **Historial de versiones (Changelog)** – Registra los cambios y mejoras.

De esta manera, cualquier persona —técnica o no— puede comprender la API desde cero hasta su uso avanzado.

---

8. Herramientas para crear y mantener documentación API

Existen múltiples herramientas diseñadas para simplificar la creación, actualización y visualización de documentación:

- **Swagger (OpenAPI):**
  Permite crear documentación interactiva donde se pueden probar las peticiones directamente desde el navegador. Usa formato YAML o JSON.

- **Postman:**
  Además de probar endpoints, permite **generar documentación automática** basada en las colecciones creadas.

- **Redoc:**
  Crea documentación visual y navegable a partir de especificaciones OpenAPI.

- **Apiary:**
  Ofrece un entorno para diseñar, documentar y testear APIs REST. Ideal para equipos colaborativos.

- **Slate:**
  Genera documentación estática con un diseño moderno, limpio y fácil de leer.

- **Docusaurus:**
  Creado por Facebook, se usa para construir portales de documentación completos, ideal si la API forma parte de un ecosistema mayor.

Estas herramientas no solo ayudan a **documentar**, sino también a **mantener la coherencia** y **facilitar las pruebas** de los endpoints.

---

9. Mejores prácticas para mantener una documentación profesional

- **Claridad ante todo:** Explicar conceptos de forma simple y sin tecnicismos innecesarios.

- **Ejemplos funcionales:** Siempre mostrar peticiones y respuestas reales.

- **Actualización constante:** Cada cambio en la API debe reflejarse en la documentación.

- **Formato consistente:** Mantener la misma estructura, estilo y nombres en todo el documento.

- **Errores documentados:** Incluir mensajes de error, causas y soluciones.

- **Uso de herramientas estándar:** Swagger, Postman o Redoc para facilitar la comprensión.
- **Multilenguaje:** Ofrecer ejemplos en varios lenguajes de programación.
- **Enlaces útiles:** Referencias a recursos externos o repositorios de ejemplo.

---

10. Ejemplo práctico en JavaScript

Endpoint: `GET /api/usuarios`
Descripción: Devuelve la lista completa de usuarios registrados en la base de datos.

```
// Ejemplo: obtener lista de usuarios desde la API
fetch('https://miapi.com/api/usuarios', {
  method: 'GET',
  headers: {
    'Authorization': 'Bearer TU_TOKEN_AQUI',
    'Content-Type': 'application/json'
  }
})
  .then(response => {
    if (!response.ok) throw new Error('Error en la solicitud: ' + response.status);
    return response.json();
  })
  .then(data => {
    console.log('Lista de usuarios:', data);
  })
  .catch(error => {
    console.error('Error:', error.message);
  });
```

Respuesta esperada (JSON):

```
[
  { "id": 1, "nombre": "Juan Pérez", "email": "juanperez@gmail.com" },
  { "id": 2, "nombre": "María Gómez", "email": "maria@gmail.com" }
]
```

Códigos de respuesta:

- `200 OK`: Lista obtenida correctamente.
- `401 Unauthorized`: Token no válido o faltante.
- `500 Internal Server Error`: Error interno del servidor.