Material descargable

egg

API Testing

¿Qué es una API?

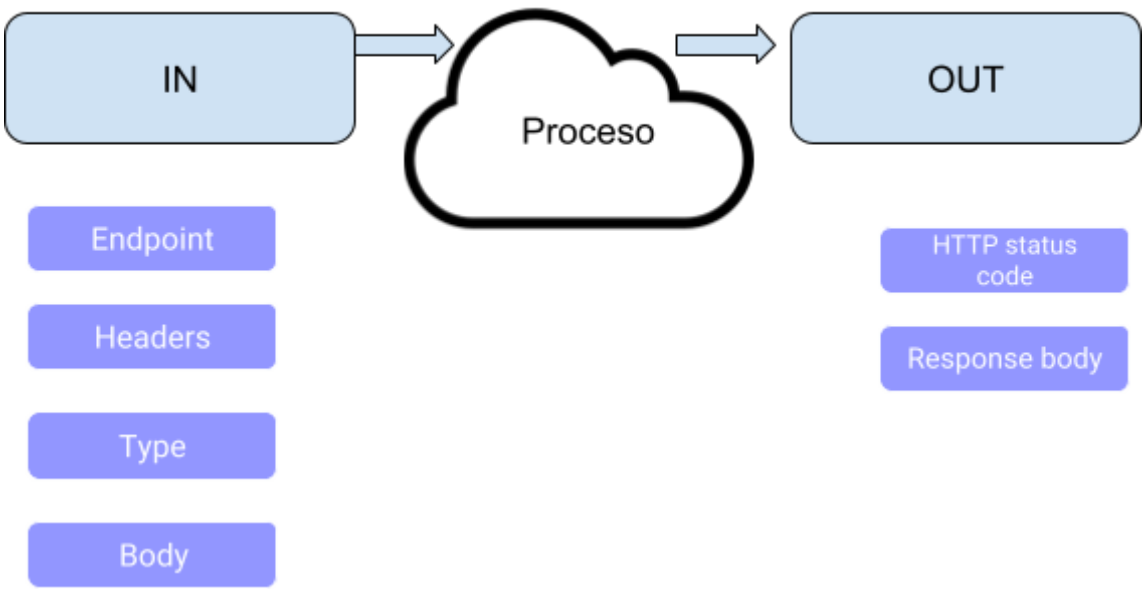
Una **API (*Interfaz de Programación de Aplicaciones*)** es un conjunto de funciones y reglas que permiten que distintos programas se comuniquen entre sí a través de solicitudes y respuestas estructuradas.

Las API funcionan como puntos de acceso que facilitan la integración entre aplicaciones, servicios o plataformas, permitiendo el intercambio de datos y la ejecución de funciones de manera controlada y segura. Gracias a esto, los desarrolladores pueden incorporar funcionalidades de otras aplicaciones sin necesidad de programarlas desde cero.

Existen diferentes tipos de API según su accesibilidad:

- **Públicas:** Están disponibles para cualquier desarrollador y pueden utilizarse libremente o bajo ciertas restricciones.
- **Privadas:** Solo pueden ser utilizadas por aplicaciones o usuarios autorizados dentro de una organización o ecosistema específico.

1. ¿Cómo funciona una API?



El funcionamiento de una API sigue una serie de pasos que permiten la comunicación entre aplicaciones. A continuación, se detallan los elementos clave involucrados en este proceso:

Punto de entrada:

- Es la dirección a la que se envían las solicitudes para interactuar con la API.
- Generalmente, es una **URL** específica o una ruta de acceso que define el recurso al que se quiere acceder.

Cabeceras (*Headers*):

- Son metadatos incluidos en la solicitud o respuesta HTTP.
- Contienen información como el tipo de contenido (*Content-Type*), autenticación, compresión, entre otros.

Cuerpo de la solicitud (*Body*):

- Incluye los datos que se envían a la API.
- Puede estar en distintos formatos, como **JSON**, **XML** o **texto plano**, según la implementación de la API.

Procesamiento:

- La API recibe la solicitud y realiza las acciones necesarias, como acceder a bases de datos, ejecutar cálculos o interactuar con otros servicios.

Código de estado:

- Es un número de tres dígitos en la respuesta HTTP que indica el resultado de la solicitud.
- Algunos códigos comunes son:
    - **200 OK:** La solicitud se procesó con éxito.
    - **400 Bad Request:** Hubo un error en la solicitud.
    - **401 Unauthorized:** Falta autenticación o no es válida.
    - **404 Not Found:** El recurso solicitado no existe.
    - **500 Internal Server Error:** Error en el servidor.

Respuesta:

- Contiene los datos devueltos por la API, que pueden ser la información solicitada, una confirmación de la acción realizada o un mensaje de error en caso de problemas.

2. Endpoints, queries y params

Un **endpoint** es un punto de acceso único dentro de una API, al que se puede acceder a través de una **URL** (*Uniform Resource Locator*). Es la dirección específica donde se envían solicitudes y desde donde se reciben respuestas. Cada **endpoint** está diseñado para ejecutar una función específica, como recuperar, crear, actualizar o eliminar recursos.

Para personalizar las solicitudes y obtener respuestas más específicas, se pueden incluir **parámetros** en la URL. Estos varían según el tipo de solicitud HTTP (*GET, POST, PUT, DELETE, etc.*) y su propósito. Existen dos tipos principales de parámetros:

- **Parámetros de Ruta (*Path Parameters*):**
  - Se utilizan para identificar un recurso específico dentro de la API.
  - Se incluyen directamente en la URL, generalmente como parte de la estructura del **endpoint**.
  - Ejemplo**: /usuarios/123** → Aquí, 123 es un parámetro de ruta que representa el ID de un usuario específico.

- **Parámetros de Consulta (*Query Parameters*):**
  - Permiten filtrar, ordenar o personalizar la respuesta del servidor.
  - Se añaden al final de la URL, después de un signo de interrogación (?). Si hay más de un parámetro, se separan con el símbolo &.
  - Ejemplo**: /usuarios?edad=25&pais=Argentina** → En este caso, edad=25 y pais=Argentina son parámetros de consulta que filtran la lista de usuarios según la edad y el país.

3. Tipos de peticiones

En el contexto de las API, las solicitudes HTTP permiten interactuar con los recursos del servidor. A continuación, se presentan las peticiones más comunes, alineadas con las operaciones del modelo **CRUD** (*Create, Read, Update, Delete*):

GET ***Lectura de Datos (Read)***	Se utiliza para solicitar datos de un recurso específico. Es una operación de **solo lectura**, por lo que **no modifica** los datos en el servidor.	• **Segura**: No altera el estado del servidor. • **Idempotente**: Puede repetirse sin generar cambios en los datos. • No debe incluir un cuerpo de solicitud.

POST ***Creación de Recursos (Create)***	Se utiliza para **enviar datos** al servidor y crear un nuevo recurso. Comúnmente se emplea para **formularios** o **datos estructurados**.	• **No segura**: Modifica el estado del servidor. • **No idempotente**: Múltiples solicitudes pueden generar múltiples recursos. • **Debe incluir** un cuerpo de solicitud con los datos a enviar.
PUT ***Actualización Total de Recursos (Update)***	Se utiliza para **actualizar completamente** un recurso existente.	• **No segura**: Modifica el estado del servidor. • **Idempotente**: Repetir la misma solicitud no genera cambios adicionales. • **Debe incluir** un cuerpo de solicitud con todos los datos del recurso, incluso si no cambian.
DELETE ***Eliminación de Recursos (Delete)***	Se utiliza para **eliminar** un recurso del servidor.	• **No segura**: Modifica el estado del servidor. • **Idempotente**: Si el recurso ya fue eliminado, repetir la solicitud no genera errores. • **No debe incluir** un cuerpo de solicitud.

🚀 **¿Quieres profundizar más?** Existen otras peticiones HTTP con funcionalidades adicionales. Te invitamos a investigar sobre:

- **PATCH** → Para actualizaciones parciales.
- **HEAD** → Para obtener solo los encabezados de una respuesta.
- **OPTIONS** → Para conocer los métodos soportados por un endpoint.

4. Códigos de respuesta HTTP

Los códigos de estado HTTP son respuestas estándar que los servidores web envían a los clientes para indicar el resultado de una solicitud. Se dividen en cinco categorías principales, cada una con un propósito específico. Conocerlas facilita la interpretación de las respuestas del servidor y permite diagnosticar posibles errores de manera más eficiente.

Las principales clases de códigos HTTP son:

- **1xx - Informativos**: Indican que la solicitud fue recibida y el servidor continúa procesándola.
- **2xx - Éxito:** La solicitud fue recibida, comprendida y procesada correctamente.
- **3xx - Redirección :** Indican que la solicitud debe dirigirse a un nuevo recurso o ubicación.
- **4xx - Errores del Cliente:** Representan problemas en la solicitud realizada por el cliente, como un recurso inexistente o falta de autenticación.
- **5xx - Errores del Servidor:** Indican fallos en el servidor que impiden completar la solicitud correctamente.

A continuación, encontrarás un listado detallado con los códigos más comunes en cada categoría.

¿Qué es API Testing?

El **API Testing** es el proceso de evaluación de una **Interfaz de Programación de Aplicaciones (API)** para garantizar su correcto funcionamiento, fiabilidad y cumplimiento de los requisitos establecidos. Su propósito es verificar la funcionalidad de la API enviando solicitudes con distintos parámetros y datos de entrada, asegurando que las respuestas obtenidas sean precisas y conformes con las especificaciones esperadas.

Además de la validación funcional, el API Testing puede incluir:

- **Pruebas de rendimiento** 📊: Evaluación de la respuesta de la API bajo distintas cargas de trabajo.
- **Pruebas de seguridad** 🔒: Identificación de posibles vulnerabilidades que puedan comprometer la integridad de los datos.

En términos generales, este tipo de prueba implica la ejecución de solicitudes **HTTP** (*GET, POST, PUT, DELETE*, etc.) y la posterior validación de las respuestas.

Pasos para realizar API Testing

Un proceso adecuado para probar una API debe incluir las siguientes etapas:

1. **Comprender el funcionamiento de la API**: Revisar la documentación para conocer los endpoints disponibles, los parámetros requeridos y los formatos de respuesta.
2. **Definir una combinación adecuada de parámetros**: Establecer diferentes configuraciones de solicitud para evaluar distintos escenarios de prueba.
3. **Ejecutar las pruebas**: Realizar llamadas a la API con los parámetros definidos y analizar su comportamiento.
4. **Verificar los resultados**: Comparar las respuestas obtenidas con los valores esperados para determinar su precisión y coherencia.
5. **Reportar comportamientos inesperados**: Documentar cualquier error o desviación respecto al resultado esperado para su corrección.

El **API Testing** es una práctica esencial para garantizar la calidad, estabilidad y seguridad de una API antes de su implementación en entornos de producción.