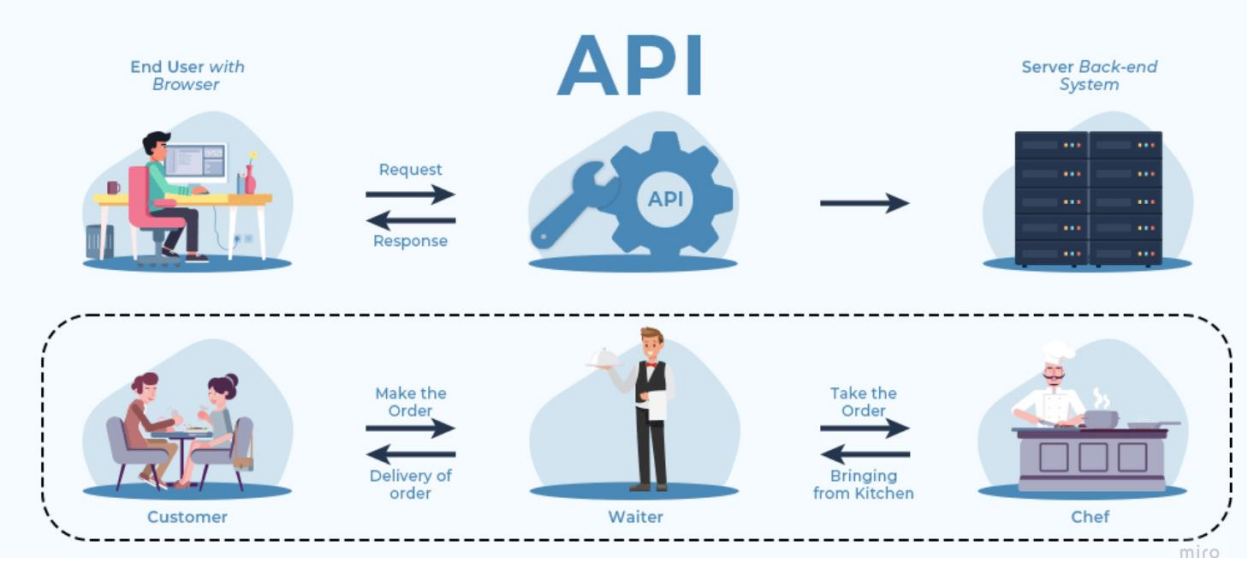
API

¿Qué es una API?

(Application Programming Interface) / *(Interfaz de programación de aplicaciones)*

Las API son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos.



Por ejemplo, el sistema de software del **instituto de meteorología** contiene datos meteorológicos diarios.

La aplicación meteorológica de su teléfono “habla” con este sistema a través de las API y le muestra las actualizaciones meteorológicas diarias en su teléfono.

Funciones de API

- **API de SOAP**

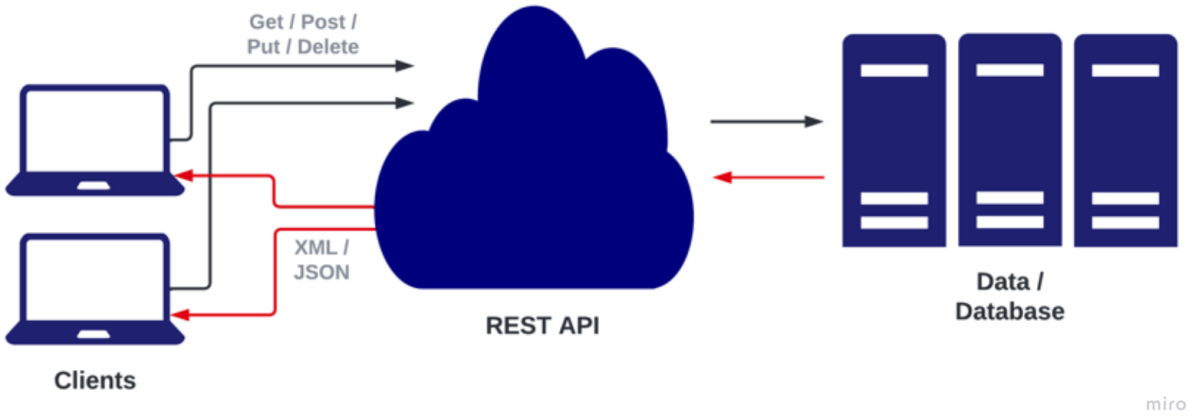
Estas API utilizan el protocolo simple de acceso a objetos. El cliente y el servidor intercambian mensajes mediante XML. Se trata de una API menos flexible que era más popular en el pasado.

- **API de REST**

Estas son las API más populares y flexibles que se encuentran en la web actualmente. El cliente envía las solicitudes al servidor como datos. El servidor utiliza esta entrada del cliente para iniciar funciones internas y devuelve los datos de salida al cliente. Veamos las API de REST con más detalle a continuación.

API REST

(Representational State Transfer)
Transferencia de estado representacional



Una API REST (Representational State Transfer) es una API que se ajusta a principios de **diseño específicos**.

Las API REST también se pueden denominar API RESTful.

Aquí te explico cómo funciona un API REST de manera general:

1. **Recursos**: En un API REST, todo es considerado como un recurso. **Un recurso puede ser cualquier cosa que se pueda nombrar, como datos, una colección de datos o incluso servicios**.

Por ejemplo, en una aplicación de redes sociales, los recursos pueden incluir usuarios, publicaciones, comentarios, etc.

1. **URI (Identificador de Recursos Uniforme)**: Cada recurso se identifica mediante un URI único (Uniform Resource Identifier). Este es el camino que se utiliza para acceder al recurso en el servidor. Por ejemplo, **/users** podría ser el URI para acceder a la lista de usuarios en una aplicación.
2. **Operaciones HTTP**: Un API REST utiliza los métodos HTTP estándar para realizar operaciones sobre los recursos. Los métodos HTTP más comunes utilizados en un API REST son:

- **GET**: Se utiliza para recuperar datos de un recurso. Por ejemplo, obtener la lista de usuarios o los detalles de un usuario específico.
- **POST**: Se utiliza para crear un nuevo recurso. Por ejemplo, agregar un nuevo usuario a la base de datos.
- **PUT**: Se utiliza para actualizar un recurso existente. Por ejemplo, actualizar la información de un usuario.
- **DELETE**: Se utiliza para eliminar un recurso. Por ejemplo, eliminar un usuario de la base de datos.
- **PATCH**: Se utiliza para realizar actualizaciones parciales en un recurso.

3. **Representaciones de Recursos**: Los recursos en un API REST pueden tener diferentes representaciones, como JSON, XML, HTML, etc. Por ejemplo, al obtener la lista de usuarios, la respuesta podría ser un JSON que contiene los detalles de cada usuario.
4. **Sin Estado (Stateless)**: Un API REST es sin estado, lo que significa que cada solicitud que realiza el cliente al servidor debe contener toda la información necesaria para comprender y procesar esa solicitud. El servidor no guarda ningún estado de sesión sobre el cliente entre las solicitudes. Esto simplifica la escalabilidad y la confiabilidad del sistema.

Harlem's de las URIS:

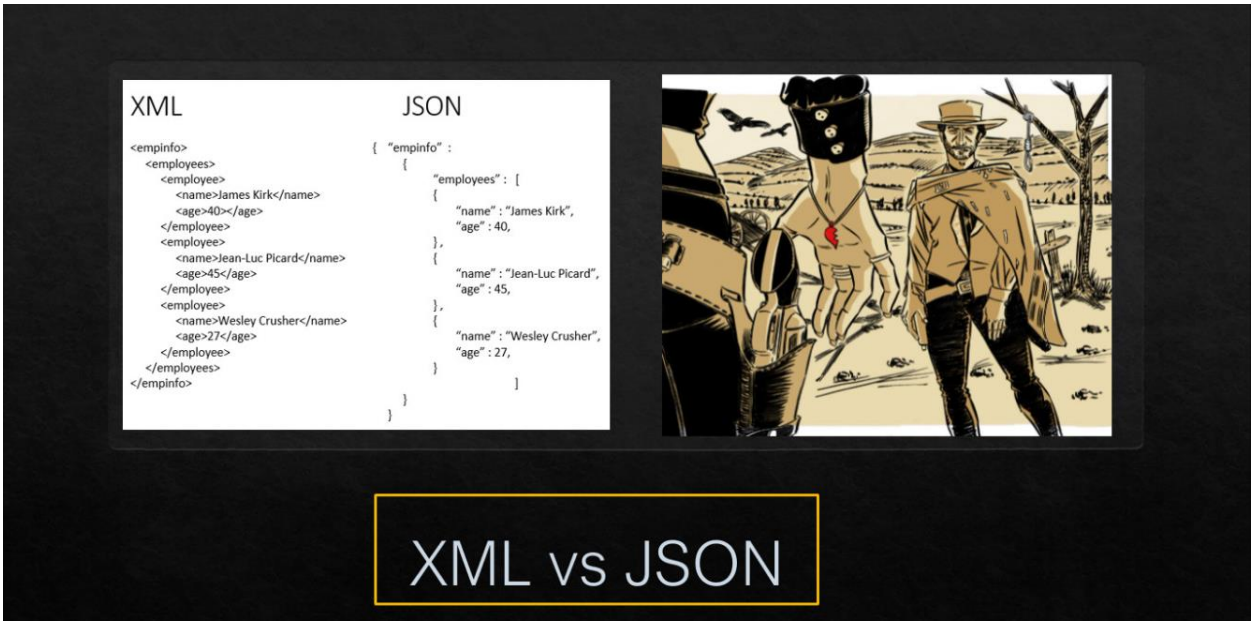
Las URIs (Uniform Resource Identifiers) son cadenas que identifican recursos en la web. La estructura básica de una URI sigue este patrón:

scheme:[//authority]path[?query][#fragment]

1. **Scheme**: Indica el protocolo utilizado para acceder al recurso (por ejemplo, http, https, ftp, etc.).
2. **Authority**: Especifica la autoridad que gestiona el recurso. En el caso de las URIs HTTP, esto generalmente incluye el nombre de dominio y, opcionalmente, el puerto (por ejemplo, www.ejemplo.com, localhost:8080).
3. **Path**: Especifica la ubicación del recurso en el servidor. Es una cadena que identifica la ruta al recurso dentro del servidor.
4. **Query**: Parámetros opcionales que se pasan al recurso. Son pares de nombre-valor que se utilizan para filtrar o personalizar la respuesta del servidor.
5. **Fragment**: Identifica una parte específica del recurso, como un ancla en una página web.

Aquí algunos ejemplos de URIS que podemos encontrar:

JSON



- **Objeto JSON**: Un objeto JSON comienza y termina con llaves {} y contiene pares de **"clave: valor"** separados por comas.

Estructura	Descripción

{
 	
"clave1": "valor1",
 	
"clave2": "valor2",
 	
"clave3": "valor3"
 	
}
	Pares clave-valor: Cada par consiste en una clave seguida de dos puntos : y un valor. Las claves son cadenas de caracteres, y los valores pueden ser de varios tipos de datos, como cadenas de caracteres, números, booleanos, objetos, matrices o null.

{
 	
"nombre": "Juan",
 	
"apellido": "Pérez"
 	
}
	Cadenas de caracteres: Las claves y los valores de tipo cadena de caracteres se colocan entre comillas dobles " ".

{
 	
"edad": 30,
 	
"altura": 1.75
 	
}
	Números: Los números no requieren comillas y pueden ser enteros o decimales.

{
 	
"esEstudiante": true,
 	
"tieneMascota": false
 	
}
	Booleanos: Los valores booleanos son true o false.

{
 	
"persona": {
 	
"nombre": "María",
 	
"edad": 25
 	
}
 	
}
	Objetos anidados: Un valor puede ser otro objeto JSON, lo que permite la anidación de datos.

{
 	
"numeros": [1, 2, 3, 4, 5]
 	
}
 	
{
 	
"Genero_Músical": [“Pop”, “Balada”, “Rock”, “Electrónica”]
 	
}
	Arrays: Los valores también pueden ser matrices (listas) de valores, que se colocan entre corchetes [].

{
 	
"estado": null
 	
}
	Null: Representa un valor nulo.

Validador de JSON en línea:

https://seostudio.tools/es/json-validator

¿Cuáles son los diferentes tipos de API?

Las API se clasifican tanto en función de su arquitectura como de su ámbito de uso. Ya exploramos los principales tipos de arquitecturas de API, ahora veamos el ámbito de uso.

API privadas

Estas son internas de una empresa y solo se utilizan para conectar sistemas y datos dentro de la empresa.

API públicas

Están abiertas al público y pueden cualquier persona puede utilizarlas. Puede haber o no alguna autorización y coste asociado a este tipo de API.

API de socios

Solo pueden acceder a ellas los desarrolladores externos autorizados para ayudar a las asociaciones entre empresas.

API compuestas

Estas combinan dos o más API diferentes para abordar requisitos o comportamientos complejos del sistema.

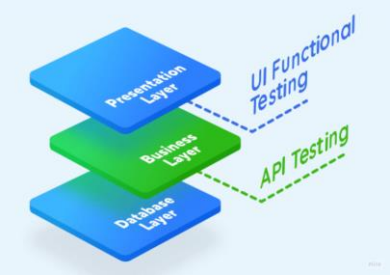
miro

¿Qué son las pruebas de API?

Las estrategias de pruebas de API son similares a otras metodologías de pruebas de software. El objetivo principal es validar las respuestas del servidor. Las pruebas de API incluyen lo siguiente:

- Hacer varias solicitudes a los puntos de conexión de la API para probar el rendimiento.
- Escribir pruebas de unidades para comprobar la lógica empresarial y la corrección funcional.
- Probar la seguridad mediante la simulación de ataques al sistema.

miro



A continuación te muestro algunos tipos de datos que podemos diseñar para testear Apis.

Obtener lista de usuarios (GET /users):

1. **Caso de prueba 1**: Verificar que se devuelva un código de estado HTTP 200 (OK) al realizar una solicitud GET a /users.
2. **Caso de prueba 2:** Verificar que el cuerpo de la respuesta contenga una lista de usuarios en el formato esperado (por ejemplo, en formato JSON).
3. Caso de prueba 3: Verificar que la lista de usuarios devuelta no esté vacía.

Obtener detalles de un usuario específico (GET /users/{id}):

1. **Caso de prueba 1:** Verificar que se devuelva un código de estado HTTP 200 (OK) al realizar una solicitud GET a /users/{id}, donde {id} es un ID de usuario válido.
2. **Caso de prueba 2:** Verificar que el cuerpo de la respuesta contenga los detalles del usuario solicitado en el formato esperado.
3. **Caso de prueba 3:** Verificar que se devuelva un código de estado HTTP 404 (Not Found) al realizar una solicitud GET a /users/{id}, donde {id} es un ID de usuario inválido o inexistente.

Crear un nuevo usuario (POST /users):

1. **Caso de prueba 1**: Verificar que se devuelva un código de estado HTTP 201 (Created) al realizar una solicitud POST a /users con los datos válidos del nuevo usuario en el cuerpo de la solicitud.
2. **Caso de prueba 2:** Verificar que el cuerpo de la respuesta contenga los detalles del nuevo usuario creado.
3. **Caso de prueba 3:** Verificar que se devuelva un código de estado HTTP 400 (Bad Request) si se intenta crear un usuario con datos inválidos o faltantes.

Actualizar los detalles de un usuario existente (PUT /users/{id}):

1. **Caso de prueba 1:** Verificar que se devuelva un código de estado HTTP 200 (OK) al realizar una solicitud PUT a /users/{id}, donde {id} es un ID de usuario válido, con los nuevos datos del usuario en el cuerpo de la solicitud.
2. **Caso de prueba 2:** Verificar que el cuerpo de la respuesta contenga los detalles actualizados del usuario.
3. **Caso de prueba 3:** Verificar que se devuelva un código de estado HTTP 404 (Not Found) si se intenta actualizar un usuario con un ID inválido o inexistente.

Eliminar un usuario (DELETE /users/{id}):

1. **Caso de prueba 1:** Verificar que se devuelva un código de estado HTTP 204 (No Content) al realizar una solicitud DELETE a /users/{id}, donde {id} es un ID de usuario válido.
2. **Caso de prueba 2:** Verificar que se devuelva un código de estado HTTP 404 (Not Found) si se intenta eliminar un usuario con un ID inválido o inexistente.