API REST

Se conoce como **API** (*Application Programming Interface*), o Interfaz de programación de Aplicaciones al conjunto de rutinas, funciones y procedimientos (métodos) que permite utilizar recursos de un software por otro, sirviendo como una capa de abstracción o intermediario.

La arquitectura REST

La arquitectura **REST** (*Representational State Transfer*) trabaja sobre el protocolo HTTP. Por consiguiente, los procedimientos o **métodos de comunicación** son los mismos que HTTP, siendo los principales: GET, POST, PUT, PATCH y DELETE. Otros métodos que se utilizan en REST API son OPTIONS y HEAD.

Otro componente de un REST API es el **HTTP Status Code**, que le informa al cliente o consumidor del API que debe hacer con la respuesta recibida.

Por lo general y mejor práctica, el cuerpo (Body) de la ***respuesta de un API es una estructura en formato jSON***.

Las principales ***ventajas*** del uso de una API REST son:

- **Separación** entre el cliente y el servidor: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos.
- **Visibilidad**, **fiabilidad** y **escalabilidad**: la separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas.
- La API REST siempre es **independiente** del tipo de plataformas o lenguajes: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando

Llamadas a la API REST

Las llamadas a una API REST permiten realizar operaciones CRUD (Crear, Leer, Actualizar y Borrar) en recursos de una base de datos, usando peticiones HTTP:

1. **Crear (POST)**: Envía el recurso a añadir en la petición.
2. **Leer (GET)**: Devuelve todos los registros de una entidad o uno específico si se proporciona un `id` en la URL.
3. **Actualizar (PUT y PATCH)**:
    - **PUT** actualiza todos los datos de un recurso.

    - **PATCH** actualiza solo los campos específicos indicados, aunque no siempre es soportado.
4. **Eliminar (DELETE)**: Elimina el recurso especificando la entidad y el `id` en la URL.

Cada operación usa una URL para identificar el recurso y un método HTTP que define la acción (por ejemplo, `GET` para leer o `POST` para crear), más el código de estado.

Clientes REST

Para interactuar con una API REST, se emplean **clientes REST**, herramientas que facilitan la comunicación entre un cliente y un servidor RESTful mediante peticiones HTTP. Los clientes REST cumplen con los principios REST, proporcionando una interfaz simplificada para manejar los recursos y métodos HTTP (GET, POST, PUT, DELETE) y presentando una experiencia más amigable para el desarrollador.

Funcionamiento básico de un cliente REST:

1. **Solicitud del cliente**: Se envía una petición HTTP al servidor RESTful indicando el recurso y el método HTTP a utilizar.
2. **Procesamiento en el servidor**: El servidor interpreta y procesa la solicitud, realiza la acción correspondiente y genera una respuesta.
3. **Respuesta del servidor**: El servidor responde con el código de estado y los datos solicitados.
4. **Interpretación del cliente**: El cliente interpreta la respuesta para verificar el resultado y procesa los datos si es necesario.

Para probar APIs REST, una opción popular es **Postman**, que permite realizar peticiones HTTP mediante una interfaz intuitiva. Un ejemplo común es la API pública de Rick and Morty, donde el método GET permite obtener información sobre los personajes.

Para realizar peticiones como POST, PUT o DELETE, es necesario especificar los datos en formato JSON en el cuerpo de la solicitud. Sin embargo, muchas APIs públicas solo permiten el método GET para evitar modificaciones de datos.

JSON Server

JSON Server es una herramienta rápida y sencilla para crear APIs REST locales con datos en formato JSON. Es útil para el desarrollo y pruebas, ya que permite simular peticiones REST sin depender de servidores externos, facilitando el trabajo desde el front-end mientras el back-end aún está en desarrollo. Al finalizar el desarrollo, basta con cambiar las URLs locales por las de producción en el servidor real.

Instalación y Configuración Básica:

- Se instala mediante `npm install -g json-server`, requiriendo tener Node.js.
- JSON Server usa un archivo `db.json` como base de datos simple en formato JSON, donde cada propiedad representa una entidad (similar a una tabla) y cada objeto dentro de los arrays representa un registro.

Ejemplo de Inicio y Uso:

- Iniciar el servidor con `json-server --watch db.json`, lo que habilita las rutas de la API en `http://localhost:3000`.
- Realizar peticiones como `GET /posts/1` devuelve el registro correspondiente en formato JSON.

Características Principales:

- Soporta operaciones **CRUD** (GET, POST, PUT, PATCH, DELETE) y guarda automáticamente los cambios en `db.json`.
- Las peticiones deben incluir un `Content-Type: application/json` en su encabezado cuando envíen datos.
- Admite filtros en peticiones GET basadas en propiedades distintas de la `id`.

Con JSON Server y Postman, es posible simular una API completa para pruebas, incluso realizar cambios en los datos, lo cual no es posible en la mayoría de las APIs públicas.