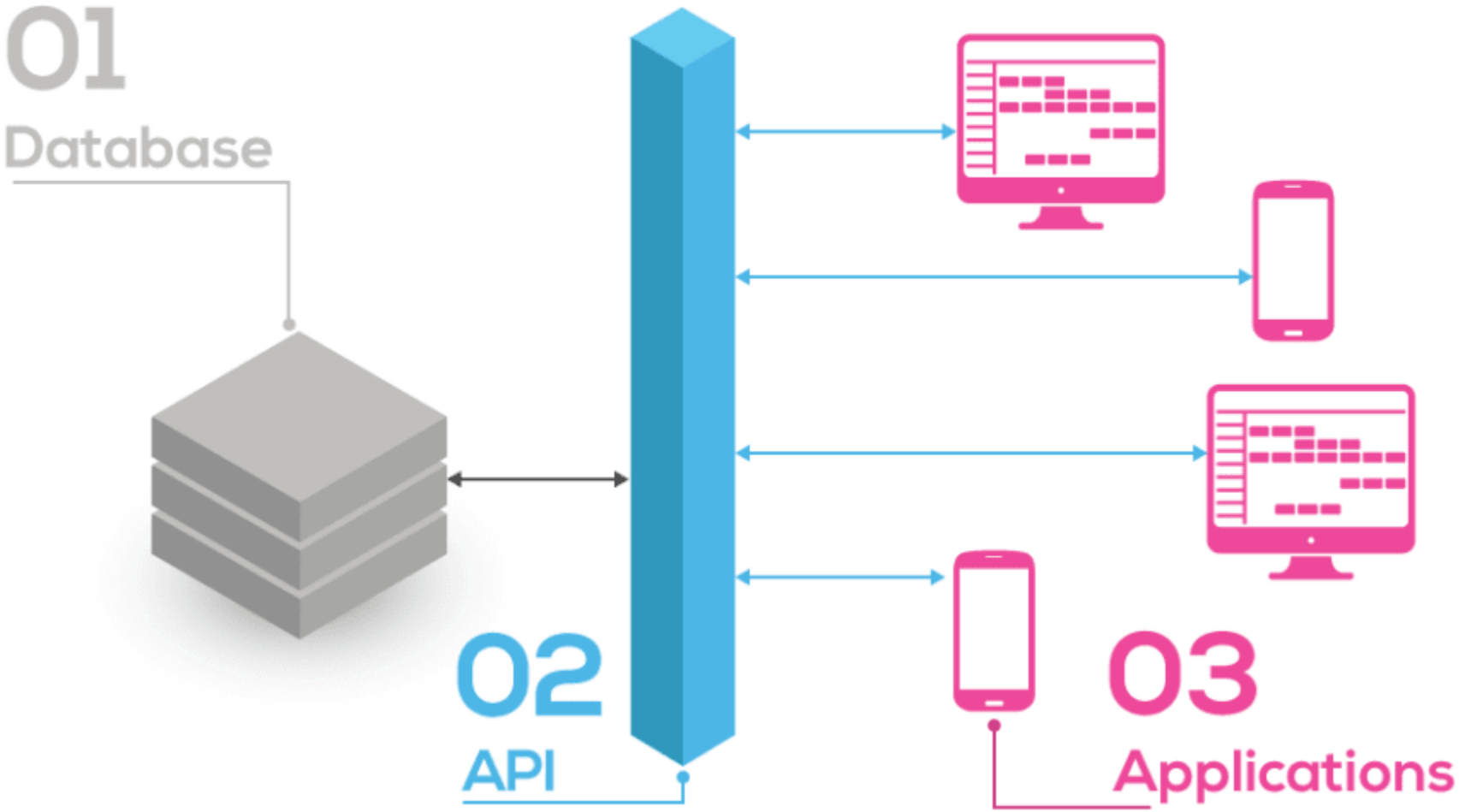

{API}

Qué es una API?

Las API, o interfaces de programación de aplicaciones, son un conjunto de funciones a través del cual dos software`s pueden comunicarse entre sí sin ninguna intermediación humana. Es un punto de entrada abstracto a una pieza de software con una descripción de la interfaz abierta y su comportamiento.

- Aplicación: por aplicación, nos referimos a un servicio con el que un desarrollador quiere interactuar. Este servicio puede ser un flujo de datos meteorológicos, una aplicación para compartir imágenes o un portal de datos abierto.
- Interfaz: la interfaz es la puerta de entrada al servicio. Debe atravesar esa puerta para interactuar con las capacidades del servicio (por ejemplo, filtrar datos meteorológicos para una ciudad, publicar imágenes en Instagram ...)
- Programa: El programa es un conjunto de procedimientos codificados escritos por un desarrollador. El programa está diseñado para interactuar con la aplicación. Por ejemplo, el programa puede enviar una dirección postal para obtener coordenadas (piense en Airbnb o Google Map).

Diagrama API



Qué hace una API?

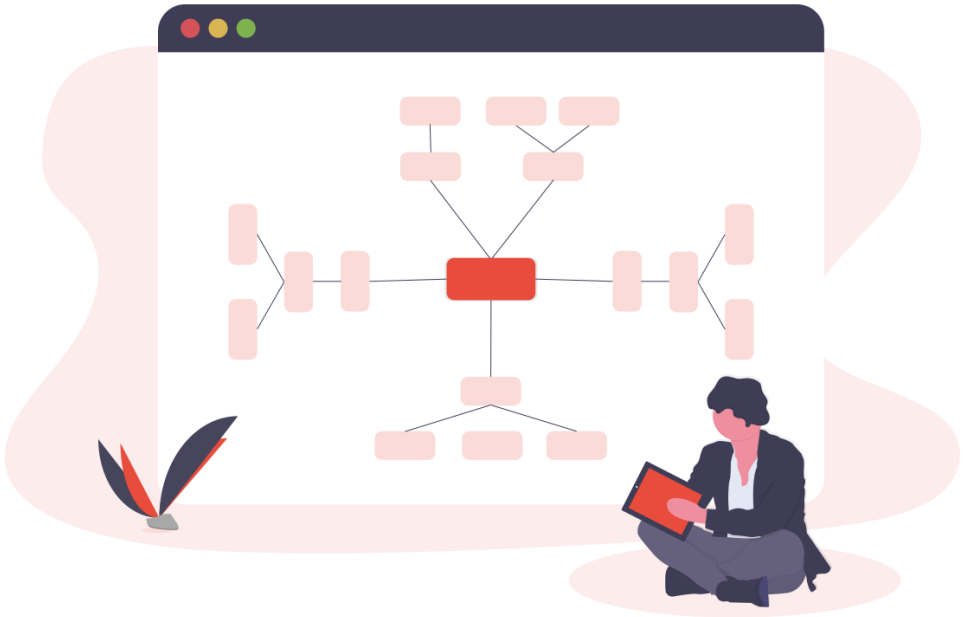
Una API permite a un desarrollador acceder a un servicio, por eso se dice que una API expone un servicio., el servicio puede venir en muchas formas y formas: flujos de datos en tiempo real (por ejemplo, Twitter), mapas (OpenStreetMap), publicación de una imagen (Instagram).
Los desarrolladores escriben programas que consumen estas API.

API como metáfora

Metafóricamente una API seria equiparable a el servicio eléctrico que proveer electricidad para que una lampara funciones, el cliente (desarrolladores) solo necesitan conocer la forma de conectar la lampara a la corriente eléctrica, si el voltaje y amperaje de la lampara es apta para con el servicio eléctrico (protocolos de peticiones).


REST {API}

- **REST** ES un estilo arquitectónico utilizado para describir la arquitectura web.
- **REST** ES independiente del protocolo.
- **REST** Trata sobre la arquitectura web (REST != API).
- **REST** NO ES un patrón de diseño.
- **REST** NO ES UN ESTÁNDAR. Sin embargo, los estándares se pueden utilizar para implementar REST.



Qué es REST

Transferencia de estado representacional es un estilo arquitectónico que fue definido por Roy Thomas Fielding en su tesis doctoral "Estilos arquitectónicos y el diseño de arquitecturas de software basadas en red".

El nombre "Transferencia de estado representacional" pretende evocar una imagen de cómo se comporta una aplicación web bien diseñada: una red de páginas web (una máquina de estado virtual), donde el usuario progresa a través de la aplicación seleccionando enlaces (transiciones de estado) , lo que hace que la página siguiente (que representa el siguiente estado de la aplicación) se transfiera al usuario y se procese para su uso.

Roy Thomas Fielding

Ventajas de REST

Al construir servicios basados en los principios de REST, uno está efectivamente construyendo servicios que son más amigables con la web. Esto se debe a que REST es un estilo arquitectónico que describe la arquitectura web, lo cual te ayudara a construir servicios que son mas:

- escalables
- de confianza
- flexibles
- portátiles

Restricciones arquitectónicas de REST

1. Cliente-Servidor: Lograr una alta cohesión y un acoplamiento flojo para mejorar la portabilidad y flexibilidad y evolucionar los sistemas de forma independiente.

2. No almacenar estados: Toda la información requerida para comprender una solicitud debe estar contenida en la Solicitud. ser autónoma nos da beneficios como:

   - Visibilidad.
   - Confiabilidad.
   - Escalabilidad.

   (desventaja: rendimiento)

3. Cache: Mejora la eficiencia en la red, establece que un servidor debe incluir datos adicionales en la respuesta para indicar al cliente si la solicitud se puede almacenar en caché y durante cuánto tiempo (desventaja: confiabilidad).

4. Interfaz uniforme: proporcionar una interfaz simplificada con mayor visibilidad que puede satisfacer los requisitos de más clientes (desventaja: interfaz sub optima).

    1. Identificar recursos (información que puede ser nombrada).
    2. Manipulación de recursos a través de representaciones (JSON, XML, etc).
    3. Mensajes auto descriptivos (Respuestas del servidor).
    4. Hypermedia como el motor del estado de la aplicación (HATEOAS) (las respuestas deben informar al usuario como proceder).

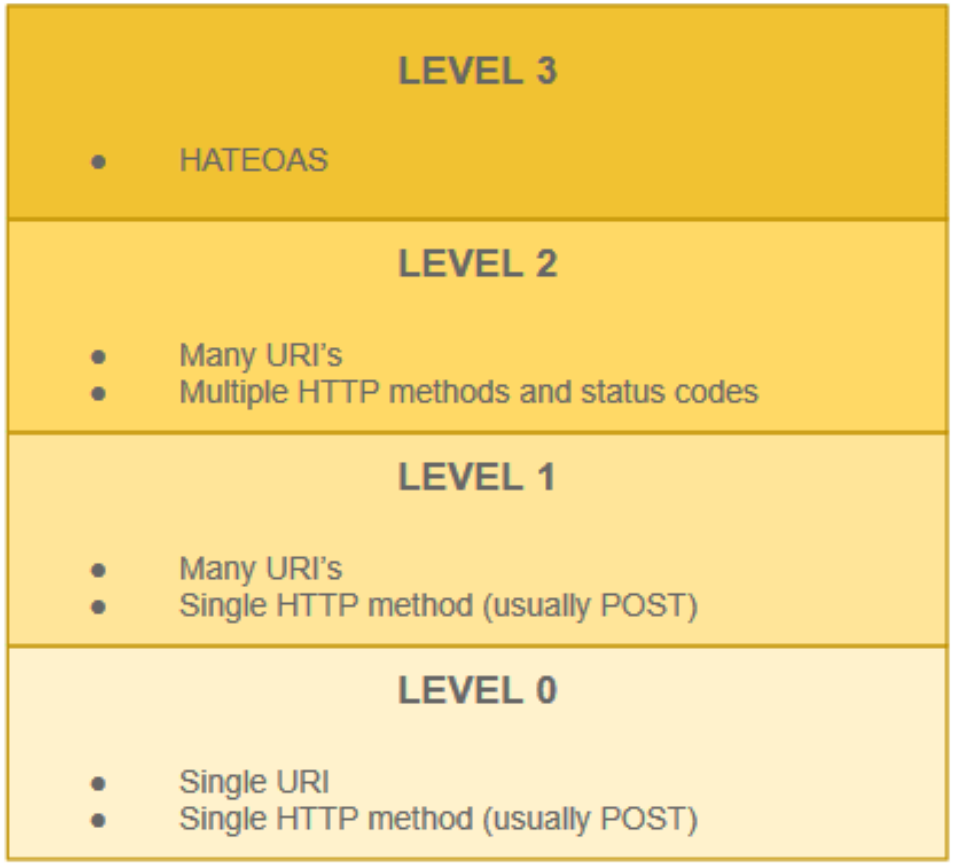
5. Sistemas en capas: El Cliente no puede suponer que se está comunicando directamente con el Servidor.

6. Código bajo demanda: (opcional) Cuando un Cliente realiza una solicitud a un recurso en un Servidor, recibirá el recurso así como el código para ejecutar contra ese recurso.

Modelo de madures de Richardson

Es un modelo de madurez heurístico que se puede utilizar para comprender mejor qué tan maduro es un servicio en términos del estilo arquitectónico REST.

Una API no puede llamarse una API REST a menos que al menos satisfaga un Nivel 3 del RMM. Por lo tanto, sería mejor pensar en las API como una API HTTP que satisface un Nivel 1,2 o 3 en el RMM.



1. Nivel 0: Los servicios en este nivel tienen un único URI y usar un solo HTTP verb (generalmente POST).
2. Nivel 1: Los servicios en este nivel tienen muchos URI`s con un solo protocolo HTTP. La principal diferencia entre el Nivel 0 y el Nivel 1 es que los servicios de Nivel 1 exponen múltiples recursos lógicos en lugar de un solo recurso.
3. Nivel 2: Los servicios en este nivel tienen muchos recursos direccionables por URI`s. Cada recurso direccionable admite múltiples HTTP verbs y códigos de estado HTTP.
4. Nivel 3: admite (HATEOAS). Por lo tanto, las representaciones de un recurso también contendrán enlaces a otros recursos (las acciones que se pueden realizar en relación con el recurso actual).