APIs: (interfaz de Programación de Aplicaciones)
Paso intermedio entre cliente y servidor.

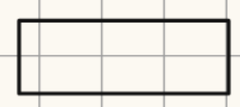
Solicitud: cliente solicita a servidor a través de la API. Para obtener info. o realizar acciones.

Respuesta: servidor procesa solicitud y envía una respuesta al cliente. La respuesta puede contener los datos solicitados o un mensaje indicando si la solicitud fue exitosa o no.

- Endpoints: direcciones URL a las que se envían las solicitudes.
- Métodos HTTPS: GET, POST, PUT, DELETE para las diferentes operaciones.
- Formato de datos: json, XML
- Documentación

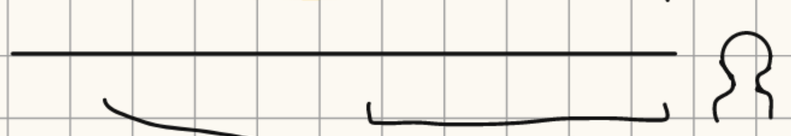
BASES DE DATOS, SQL y API:

servidor



base de datos

API: interfaz para acceder a los datos de la API.



usuario

solicitudes mediante protocolo HTTP.

formato de respuesta: archivo JSON o XML.

SQL: lenguaje para interactuar con la base de datos.

POLITICAS DE LANZAMIENTO DE APIs

privado: solo se pueden usar internamente → le da a las empresas un mayor control sobre las APIs.

de partners: se comparten con partners empresariales específicos, cosa que puede ofrecer flujos de ingresos adicionales, sin comprometer la calidad.

público: Todos tienen acceso a las APIs. Esto permite que terceros desarrollen apps que interactúan con tu API, y puede ser un recurso para innovar.

SOAP: Protocolo de Acceso a Objetos Simples

APIs diseñadas con SOAP usan XML para el formato de los mensajes.
las solicitudes son con HTTP o SMTP.

ventaja: facilita que apps que funcionen en entornos distintos o que estén escritas en lenguajes distintos compartan info.

REST: Transferencia de Estado Representacional

las API web que funcionan con las limitaciones de arquitectura REST se llaman: RESTful

diferencia: SOAP es un protocolo, REST un estilo de arquitectura.

Una API es RESTful si cumple:

Arquitectura cliente - servidor: aquí REST compuesta por clientes, servidores y recursos y administra solicitudes con HTTP.

Sin estado: contenido de los clientes no se almacena en el servidor entre solicitudes.
La info del estado de la sesión está en el cliente.

Capacidad de caché: el almacenamiento en caché puede eliminar la necesidad de algunas interacciones cliente - servidor.

Sistema en capas: las interacciones cliente - servidor pueden estar mediadas por capas adicionales.
pueden ofrecer funcionalidades adicionales como: equilibrio de carga, cachés compartidos o seguridad

Código de demanda (opcional): los servidores pueden extender las funciones de un cliente transfiriendo código ejecutable.

Interfaz uniforme: - identificación de recursos en las solicitudes: los recursos se identifican en las solicitudes y se separan de las representaciones que se devuelven al cliente.

- administración de recursos mediante representaciones: clientes reciben archivos que representan los recursos. Estas representaciones deben tener la info. suficiente como para poder ser modificadas o eliminadas.

- mensajes autodescriptivos: cada mensaje que se devuelve al cliente contiene la info. suficiente para describir cómo debe procesar la info.

- hipermedios es el motor del estado de la aplicación: después de acceder a un recurso, el cliente REST debe ser capaz de descubrir mediante hipervínculos todas las otras acciones que están dispo. actualmente

APIs conocidas: -OpenAPI: diseño de APIs de REST sin depender de un lenguaje
-GraphQL: lenguaje de consulta y entorno del lado del servidor, alternativa a REST.
prioridad: brindar los datos solicitados y nada más.

SOA: arquitectura orientada al servicio
ESB: bus de servicios empresariales
Arquitecturas de microservicios

APIs vs webhooks
→ función de devolución de llamadas
se basa en protocolo HTTP para que dos API se comuniquen mediante eventos (ligeros)
sirven para: - recibir pequeñas cantidades de datos de otras aplicaciones
- activar flujos de trabajo de automatización en entornos de GitOps.

API "inversa" o "push": El servidor se encarga de comunicación en vez de el cliente.
Envía solicitud POST de HTTP cuando los datos están dispo.
→ no son APIs y necesitan de una