UTN FACULTAD REGIONAL CÓRDOBA
INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Desarrollo de Software
Notas de Clase

Clase 09

Arquitectura de APIs (*)

Punto de partida (*)

La arquitectura de software define, de manera abstracta, los componentes que llevan a cabo una tarea de computación, sus interfaces y la comunicación entre ellas. Toda arquitectura debe ser implementable en una arquitectura física, que consiste simplemente en determinar qué computadora tendrá asignada una determinada tarea.

Un patrón arquetípico (*)

Las arquitecturas de referencia ayudan a organizar y alinear el trabajo de distintos equipos bajo un patrón arquetípico que engloba tareas de un mismo tipo.

¿Qué es una arquitectura de referencia? (*)

Las arquitecturas de referencia están formadas por estructuras e integraciones recomendadas de productos y servicios de TI para crear una solución. Estas arquitecturas incorporan las mejores prácticas y lineamientos de un determinado sector y se pueden aplicar a diferentes contextos. Una arquitectura de referencia se anticipa y da respuesta a las preguntas más frecuentes que puedan surgir. En consecuencia, ayuda a los equipos a evitar los errores y los retrasos que se podrían producir sin el uso de un conjunto probado de mejores prácticas y soluciones.

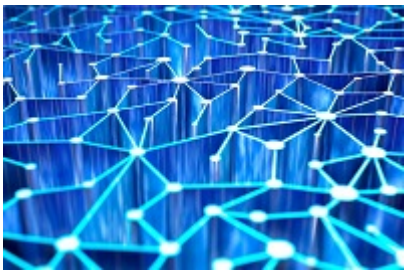
Importancia de la selección de una arquitectura de referencia (*)

La selección de una arquitectura de referencia adecuada es muy importante, ya que son los cimientos sobre los que todo el desarrollo se monta. Al igual que con la construcción de un edificio, si los cimientos no son buenos, la construcción de todo el edificio será problemática, llegando incluso a impedir que se termine el edificio. Elegir la mejor arquitectura para solucionar el problema puede garantizar un desarrollo ágil y mantenible, y un producto duradero en el tiempo.

Arquitectura monolítica (*)

Una arquitectura monolítica se refiere al clásico diseño unificado de un software, donde todo está acoplado y unido en un solo elemento. De ahí su nombre, que hace referencia a un monolito (monumento de piedra de una sola pieza).

El software monolítico está diseñado para ser autocontenido, todos sus componentes están interconectados y son interdependientes entre sí.



En esta arquitectura, fuertemente acoplada, cada componente y sus elementos asociados tienen que estar presentes para que el código se ejecute, ya que esta arquitectura concentra toda funcionalidad en un solo punto.

¿Cuáles son los principales beneficios de este tipo de arquitectura?

- Los programas son fáciles de desarrollar.
- El despliegue y la ejecución del software son muy sencillos.
- El costo de desarrollo es bajo en comparación con otras arquitecturas.

Arquitectura de microservicios (*)

Los microservicios son todo lo opuesto a la arquitectura monolítica, ya que los procesos se distribuyen en múltiples partes, más pequeñas e independientes.

Una de las maneras de implementar este patrón es la distribución de tareas en bloques separados. Estos bloques se llaman “componentes de servicios” y agrupan de uno a varios componentes enfocados en realizar una tarea o un área de negocio de la aplicación.

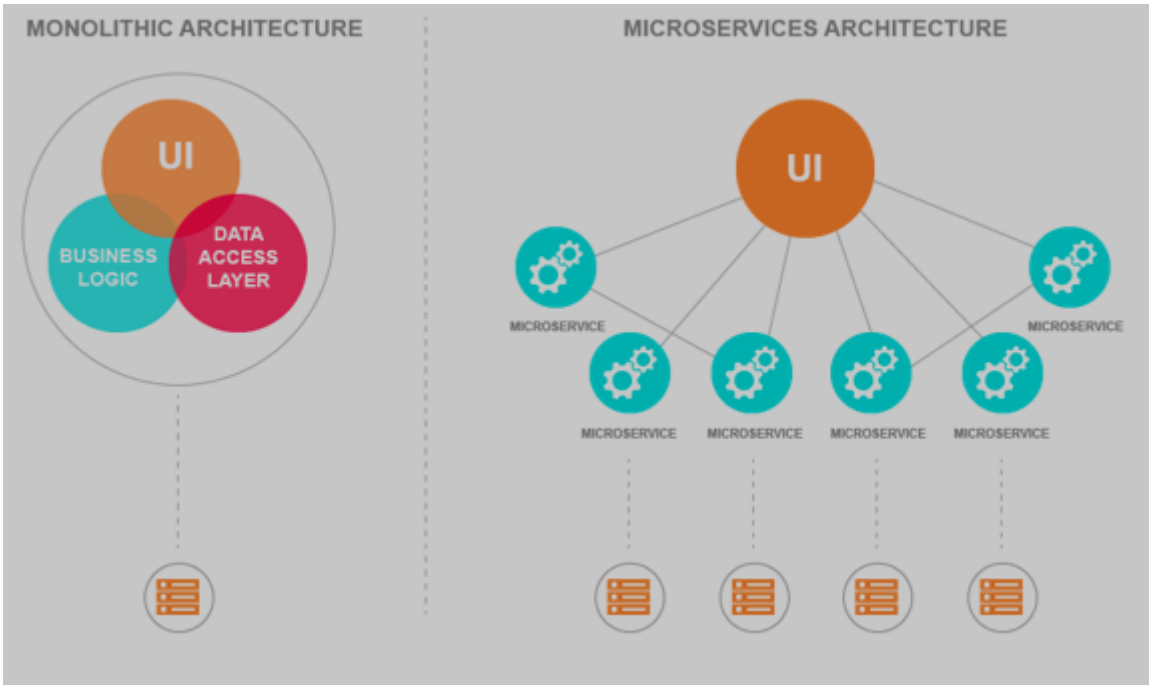
Estos componentes de servicios están completamente desacoplados entre sí y funcionan de forma totalmente independiente, lo que permite que cualquiera de ellos pueda ser reemplazado en tiempo real sin afectar el funcionamiento de la aplicación.

¿Qué es un microservicio? (*)

Un microservicio es un componente independiente y autónomo de un sistema o aplicación más amplio. Un microservicio se encarga de implementar una funcionalidad completa de negocio. Esta funcionalidad es privada y se puede exponer al exterior a través de una API.

El software construido en base a microservicios se puede descomponer en varias partes funcionales independientes. De esta manera, cada uno de estos servicios podrá ser desplegado, modificado y redesplegado sin comprometer los otros aspectos funcionales de la aplicación.

Esto trae como resultado que, en caso de necesitarlo, solo tendremos que modificar un par de servicios en lugar de redesplegar toda la aplicación por completo nuevamente.



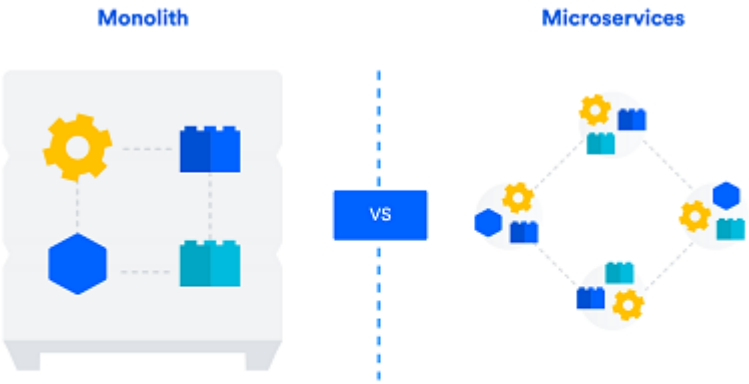
¿Cómo funcionan los microservicios? (*)

La arquitectura de microservicios es un método de desarrollo que sirve para estructurar tanto una aplicación como una combinación de servicios.

Son componentes autónomos e independientes de un sistema más amplio.

Cada microservicio se encarga de implementar una funcionalidad completa de negocio.

¿Por qué puede resultar beneficioso trabajar con microservicios? (*)



- Agilidad: Cada microservicio se puede diseñar, desarrollar e implementar con independencia de otros microservicios. Esto aporta agilidad, dado que es más fácil implementar nuevas versiones de los microservicios con frecuencia.
- Diversidad: Con un sistema compuesto de múltiples servicios colaborativos, como el caso del enfoque de microservicios, podemos decidir utilizar diferentes lenguajes de programación y tecnologías dentro de cada servicio.
- Productividad: Esto nos permite elegir la herramienta adecuada para cada tipo de trabajo, en lugar de tener que elegir una estandarizada, es decir, una talla única para todo.
- Independencia: Cada equipo puede administrar, desarrollar, implementar y escalar su servicio de forma independiente a los demás equipos.

¿Qué es una API? (*)

La abreviatura API significa "interfaz de programación de aplicaciones".

Hablando de componentes de software, una API es la interfaz pública de un componente que permite comunicarse con él para hacer uso del mismo. En el contexto de arquitecturas de microservicios, una API describe una interfaz de servicio expuesta a través del protocolo HTTP.

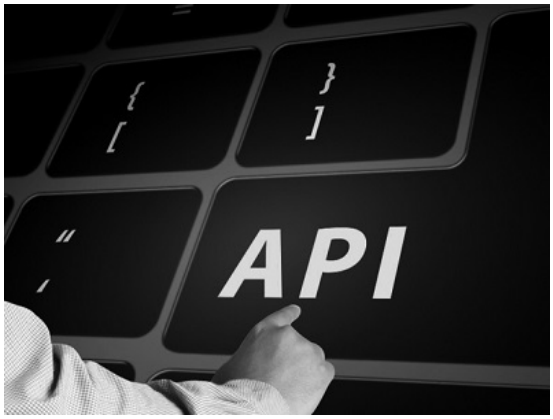
El propósito de una API es intercambiar datos entre diferentes sistemas.

Las APIs le permiten al usuario final utilizar aplicaciones, programas y softwares consultando, cambiando y almacenando datos de diferentes sistemas, como si fuera una especie de traductor.

Un ejemplo de API es el conjunto de funciones que permite a distintos sistemas el envío de notificaciones a clientes por diferentes medios (correo electrónico, sms o mensajes al celular, notificaciones Push, etc.). Las notificaciones Push, por ejemplo, son muy utilizadas en los dispositivos móviles para informar sobre actualizaciones de las apps, novedades, hitos, etc.

Desde una perspectiva de negocio ...	**Desde una perspectiva técnica ...**

Desde una perspectiva de negocio ...	Desde una perspectiva técnica ...
	
Las APIs exponen y extienden el negocio.	Es un contrato y una vía de acceso que permite la interoperabilidad entre sistemas.

Para el consumidor de la API	El proveedor de la API
	
La misma no es más que una descripción de la interfaz y un endpoint (o conjunto de URLs). El idioma es el protocolo HTTP.	se tienen en cuenta todos los detalles de la infraestructura de implementación de la misma.

Características que debe tener una API (*)

- Extensibilidad: Su funcionalidad debe ser fácil de extender. Las personas siempre van a requerir más funcionalidades:
- Predictibilidad y consistencia:Usar una API no debería tener comportamientos sorpresa. Si bien puede afectar por perfiles diferentes, el comportamiento debe ser consistente para todos.
- Errores claros: Evitar fallos. Contemplar todas las validaciones necesarias.

Algunos usos de APIs (*)

- Usar Google Maps en aplicaciones.

- Integrar videos de Youtube en una página web.
- Consultar el clima en Google.
- Loguearte en una página web a través de tus cuentas de redes sociales.
- Crear bots para chats automáticos (hoy se utilizan en la mayoría de las empresas para poder dar respuesta las 24 horas).

APIs y Microservicios: una relación funcional (*)

- Los microservicios pueden ser un medio para implementar el backend de servicio expuesto por una API.
- Los microservicios generalmente dependen de las API como un medio estándar de comunicación entre ellos en una red interna.

Conceptos REST (*)

¿Qué es REST? (*)

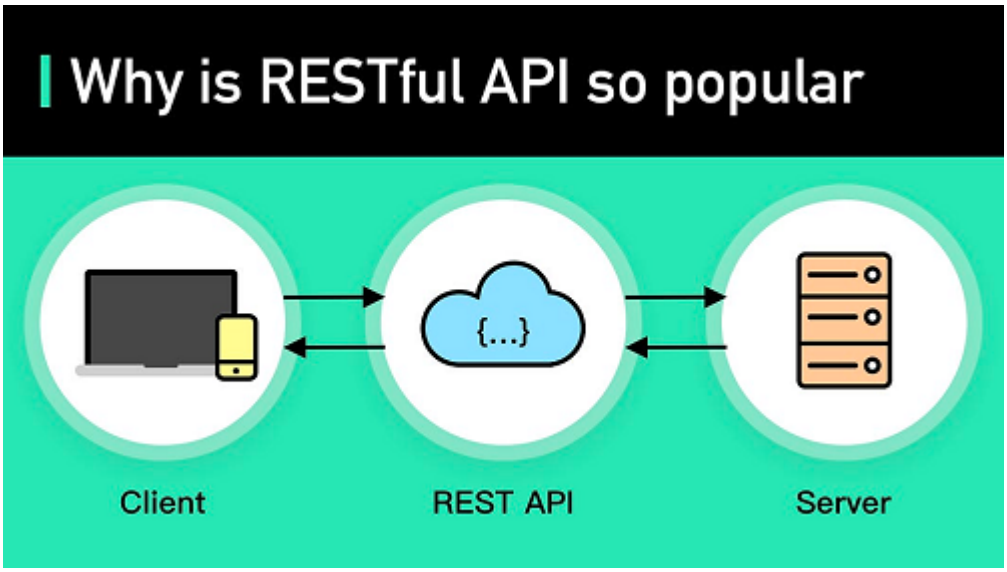
En año 2000, Roy Fielding definió REST como un estilo arquitectónico web basado en el protocolo HTTP, que impone condiciones sobre cómo debe funcionar una API.

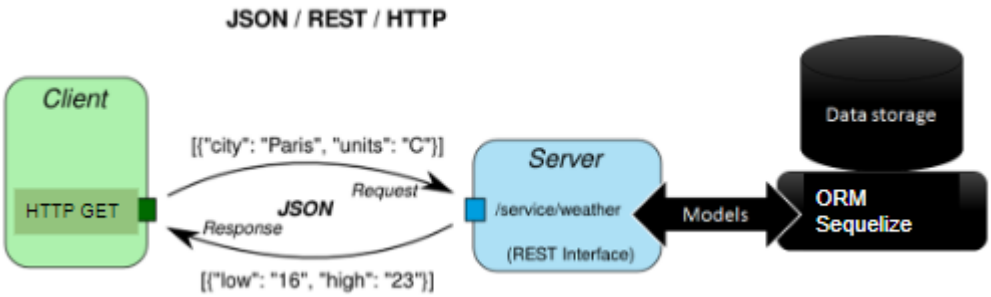
Principios REST (*)

- **Cliente / Servidor:** Definen un interface de comunicación entre ambos separando completamente las responsabilidades entre ambas partes.
- **Sin Estado:** Cada petición que se realiza a ellos es completamente independiente de la siguiente.
- **Cache:** El contenido de los servicios REST se puede cachear de tal forma que una vez realizada la primera petición al servicio, pero el resto no sea necesario procesarla.
- **Arquitectura en Capas:** El servidor puede disponer de varias capas para su implementación. Esto ayuda a mejorar la escalabilidad, el rendimiento y la seguridad.
- **Interfaz Uniforme:** Esta restricción indica que cada recurso del servicio REST debe tener una única dirección, “URI”.

¿Qué es API RESTful? (*)

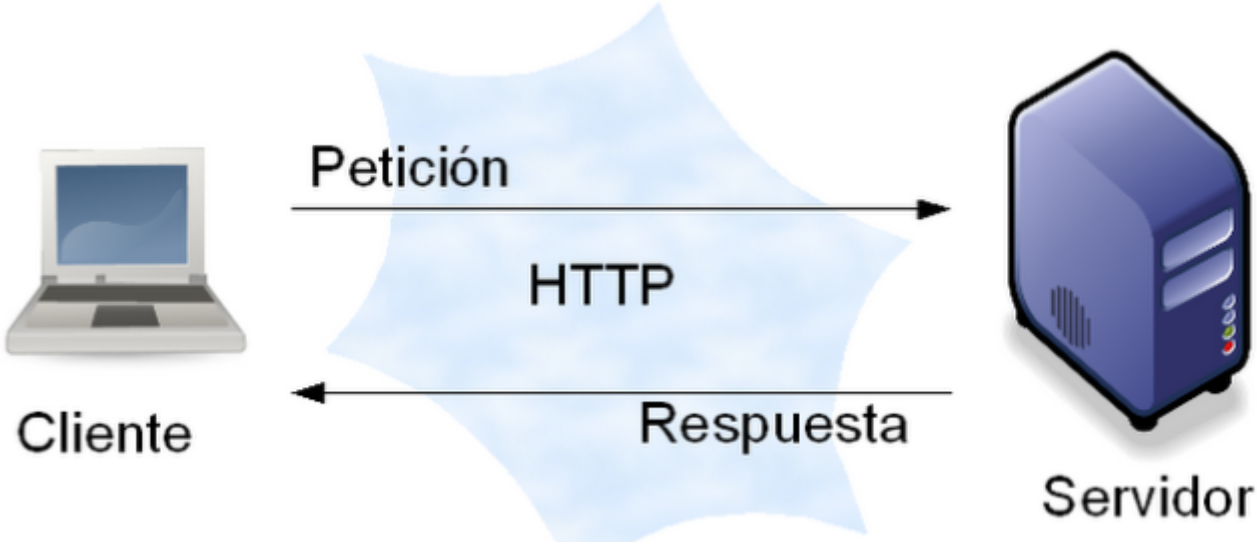
La API RESTful es una interfaz que dos sistemas de computación utilizan para intercambiar información de manera segura a través de una arquitectura Rest. La mayoría de las aplicaciones para empresas deben comunicarse con otras aplicaciones internas o de terceros para llevar a cabo varias tareas.





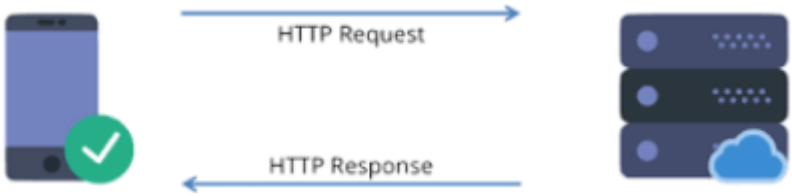
Arquitectura Cliente Servidor (*)

Las RESTFul API funcionan bajo una arquitectura cliente servidor, usando HTTP como protocolo de comunicación.



Protocolo HTTP (*)

HTTP es un protocolo de comunicación utilizado en la web para la transferencia de datos entre diferentes dispositivos y servidores. Se basa en un modelo cliente-servidor. Utiliza diferentes métodos para realizar operaciones en los datos. Utiliza un formato de mensaje simple que consta de una cabecera y un cuerpo.

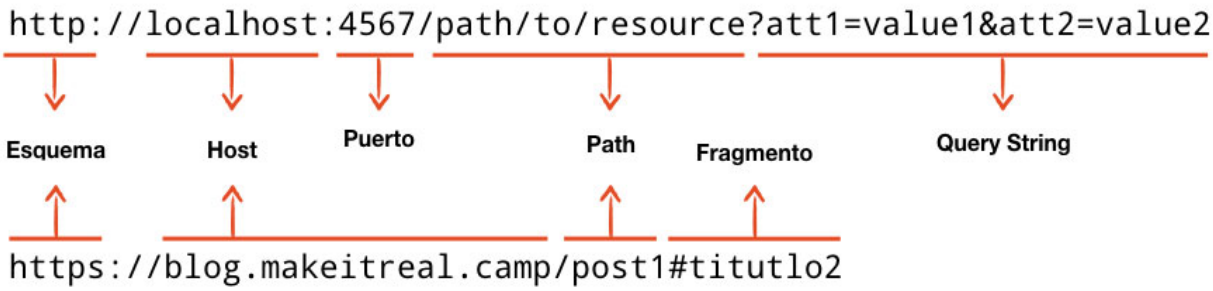


Características del Protocolo HTTP (*)

- **Basado en solicitudes y respuestas**
- **Independencia de la plataforma**
- **Simplicidad**
- **Flexibilidad**
- **Escalabilidad**

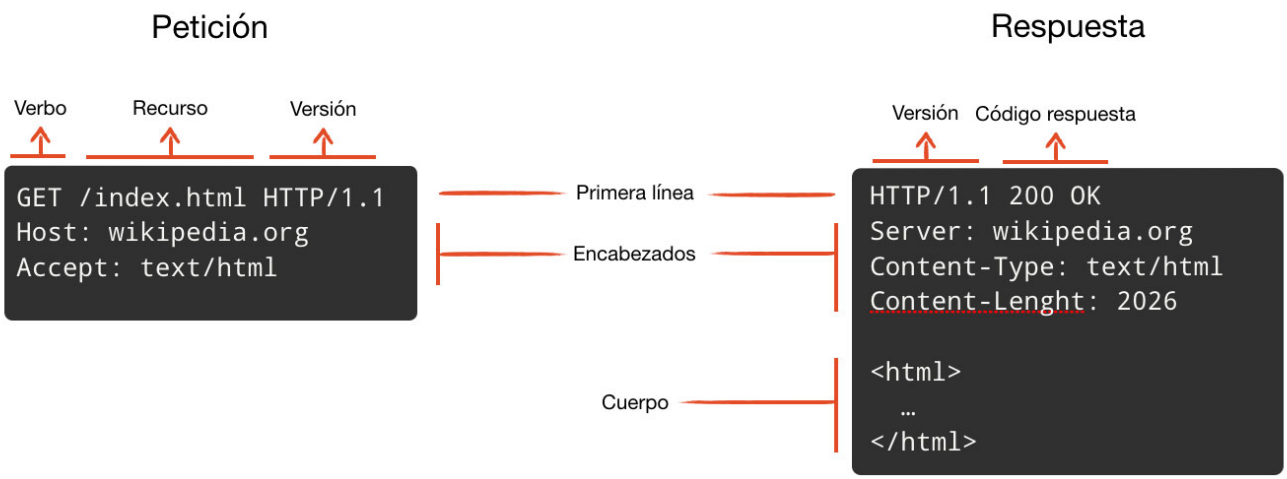
Estructura URL HTTP (*)

Un URL (Uniform Resource Locator) se utiliza para ubicar un recurso en Internet. Los URLs no solo se pueden utilizar para el protocolo HTTP, se utilizan en muchos otros protocolos.



- **Esquema:** El esquema define el protocolo a utilizar, para HTTP puede ser http o https (el protocolo seguro de HTTP).
- **Host:** La IP o el nombre del servidor que se quiere acceder (p.e. 127.0.0.1, localhost, google.com, www.google.com, etc.).
- **Puerto:** El puerto en el que está escuchando el servidor HTTP. Si se omite se asume que es el 80.
- **Path:** La ruta al recurso que se quiere acceder.
- **Query String:** Contiene información adicional para el servidor en forma de propiedades (atributo=valor). Las propiedades se separan por &.
- **Fragmento:** La referencia a una ubicación interna del documento.

Request (Petición) y Response (Respuesta) (*)


Petición
Respuesta
Verbo
Recurso
Versión
Versión
Código respuesta
GET /index.html HTTP/1.1
Host: wikipedia.org
Accept: text/html
Primera línea
Encabezados
Cuerpo
HTTP/1.1 200 OK
Server: wikipedia.org
Content-Type: text/html
Content-Lenght: 2026
<html>
…
</html>