Callbacks y APIs

Trabajar con APIs

{desafío} latam_

Utilizar elementos de programación asíncrona para resolver un problema simple distinguiendo los diversos mecanismos para su implementación acorde al lenguaje Javascript.

Utilizar el objeto XHR y la API Fetch para el consumo de una API externa y su procesamiento acorde al lenguaje Javascript.

{desafío} latam_

- Unidad 1:
  ES6+ y POO
- Unidad 2:
  Herencia
- Unidad 3:
  Callbacks y APIs

Te encuentras aquí

¿Qué aprenderás en esta sesión?

- *Describe qué es una API y su importancia para el desarrollo web distinguiendo las APIs de browser y APIs de terceros.*
- *Utiliza la API XHR para la obtención de información desde un servidor para resolver un problema simple acorde al lenguaje Javascript.*
- *Utiliza la API Fetch para la obtención de información desde un servidor para resolver un problema simple acorde al lenguaje Javascript.*
- *Codifica un programa que despliega datos de una respuesta JSON accediendo a elementos del DOM utilizando lenguaje Javascript para resolver un problema.*

{desafío} latam_

Comenta con tus palabras:

¿Tienes alguna idea de lo que es o para qué sirve una API?

/* Qué es una API */

Qué es una API

- API: Interfaz de Programación de Aplicaciones (Application Programming Interface).
- Es una especificación formal sobre cómo un módulo de un software se comunica o interactúa con otro.
- Una de las principales funciones de las API es poder facilitar el trabajo a los desarrolladores, ahorrar tiempo y dinero.

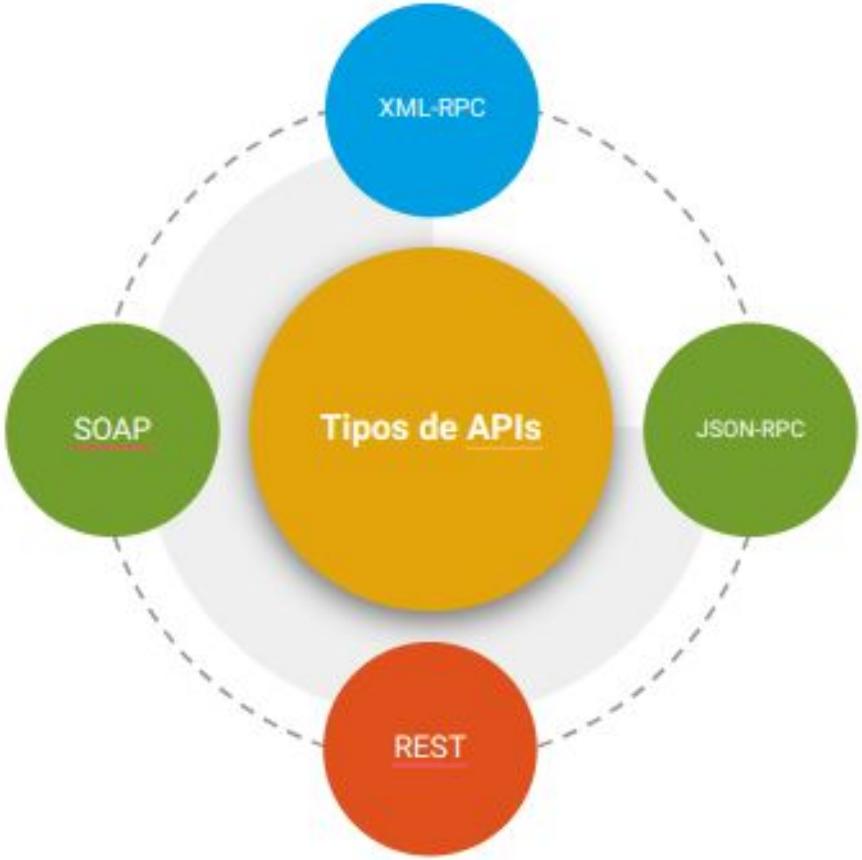
/* Para qué sirve una API */

Para qué sirve una API

Una de las principales funciones de las API es poder facilitar el trabajo a los desarrolladores, ahorrar tiempo y dinero. Por ejemplo, si estás creando una aplicación que es una tienda en línea, no es necesario crear desde cero un sistema de pagos u otro sistema para verificar si hay stock disponible de un producto. Para ello, se puede utilizar una API de servicio de pago ya existente, por ejemplo PayPal, además pedir al distribuidor una API que permita saber la cantidad de productos o inventario que existen.

/* Tipos de APIs de servicios web */

Tipos de APIs de servicios web



/* SOAP (Simple Object Access Protocol) */

SOAP (Simple Object Access Protocol)

- Sirve para que dos procesos puedan comunicarse intercambiando datos XML.
- Se usa para exponer servicios web a través del protocolo HTTP.
- Soporta sólo XML como formato de datos.

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-enc
oding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>.
```

{desafío} latam_

/* REST (Representational State Transfer) */

REST (Representational State Transfer)

- Interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON.

- La arquitectura REST define algunas restricciones o características para ser utilizada, como el caso de: Cliente/Servidor, Sin estado, Caché, Sistema de capas, Interfaz uniforme y Código bajo demanda.

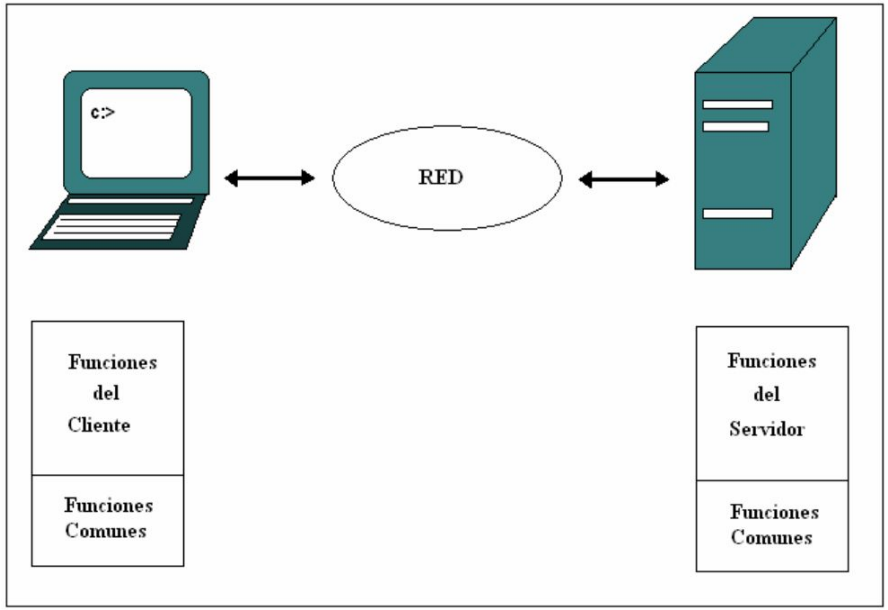
Conversemos, ¿Para qué necesitamos una API?

{desafío} latam_

/* Cliente/Servidor */

Cliente/Servidor

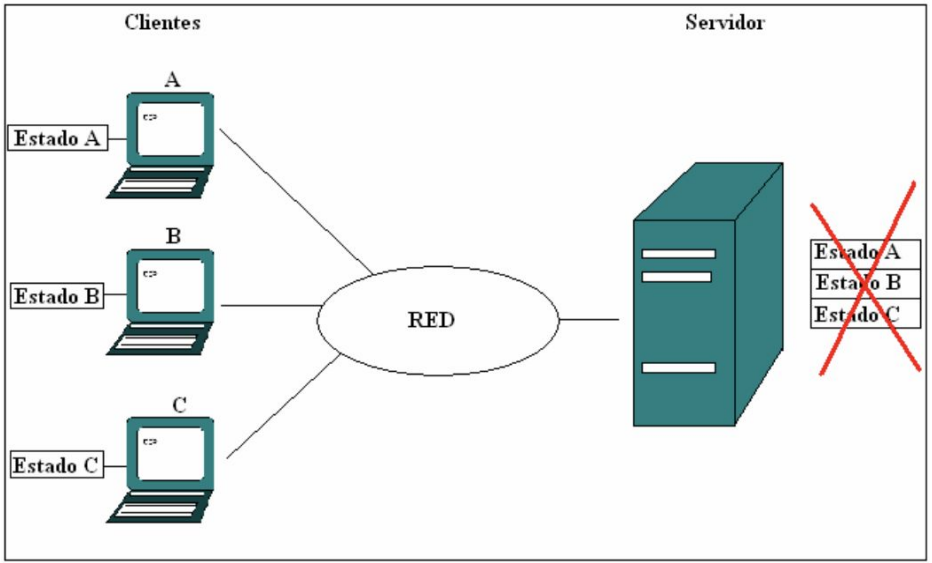
Corresponde a la separación de Cliente/Servidor, cada uno debe manejar sus recursos, el cliente debe manejar la interfaz y el servidor el almacenaje de los datos. Con esto se mejora la portabilidad de las interfaces de usuario y escalabilidad para los componentes de los servidores, ya que no deben preocuparse por lo que pasa en la interfaz de usuario.



Fuente: http://bibing.us.es

/* Sin estado */

Sin estado



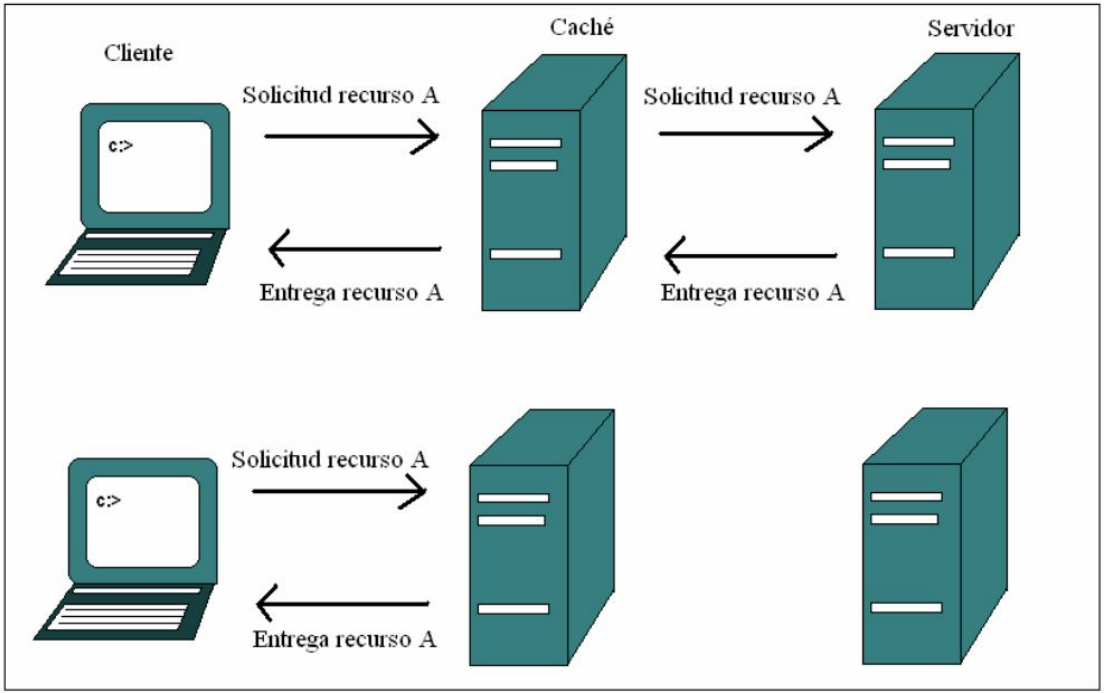
Fuente: http://bibing.us.es

Cada petición que se realice desde el cliente al servidor, deberá contener toda la información necesaria para que el servidor comprenda lo que se está solicitando y no deberá haber ningún dato guardado con anterioridad y en el que pueda revisarlos.

/* Caché */

Caché

Las peticiones que se realicen podrán o no ser “cacheables”, dependerá si el cliente le otorga el permiso al servidor para obtener estas respuestas en caso de hacer las mismas solicitudes.



Fuente: http://bibing.us.es

/* Interfaz uniforme */

Interfaz uniforme

La idea de utilizar una interfaz uniforme nace de poder estandarizar solicitudes de datos, por lo que no sería necesario hacer solicitudes de tipo createUser, updateProduct, removeTicket, etc. La interfaz de REST tiene un diseño eficiente para el manejo de grandes volúmenes de datos y con esto se optimiza la web.

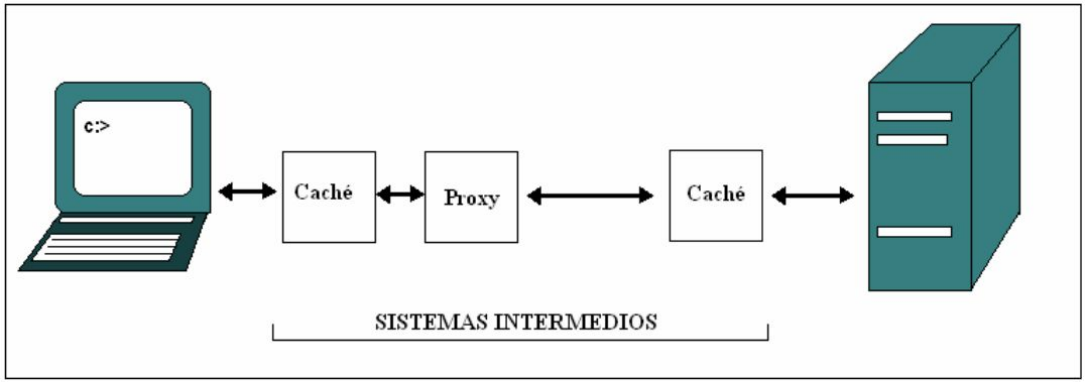
Para este estándar se establecen algunas restricciones:

- Identificación de recursos.
- Manipulación de recursos a través de sus representaciones.
- Mensajes auto-descriptivos.
- Hipermedios como el motor del estado de la aplicación.

/* Sistema de capas */

Sistema de capas

La idea de tener un sistema de capas es aumentar la escalabilidad de las aplicaciones, separando por componentes y limitando su comportamiento a lo que debe realizar para complementar todo el sistema.



Fuente: http://bibing.us.es

/* Código bajo demanda */

Código bajo demanda

Este ítem es opcional y permite a los clientes descargar el código y ejecutarlo en forma de scripts. Esto le da al cliente la posibilidad de extender la cantidad de funcionalidades de la aplicación.

/* API REST */

API REST

- El término API REST significa utilizar una API para acceder a aplicaciones backend, de manera que esa comunicación se realice con los estándares definidos por el estilo de arquitectura REST.

- Para realizar el envío y recepción de datos se utilizan las especificaciones más importantes del protocolo HTTP que son GET, POST, PUT, DELETE.

- Todas las solicitudes cuentan con una URL, el tipo y los datos que se requiere enviar.

/* URL (Uniform Resource Locator) */

URL (Uniform Resource Locator)

Localizador uniforme de recursos, es una cadena de caracteres que tiene como funcionalidad permitir localizar cada recurso en internet. Cada recurso que existe en internet posee una URL, sitios, documentos, archivos, carpetas.

/* Tipos de solicitudes */

Tipos de solicitudes

/* Datos */

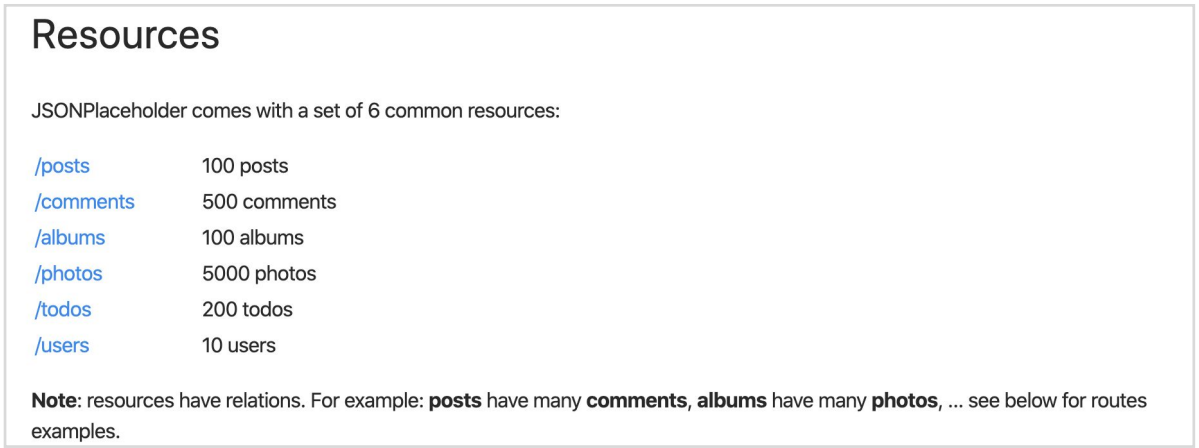
Datos

Los datos pueden ser establecidos en el formato según la herramienta que se esté utilizando para realizar las solicitudes a la API. Las APIs REST no son lenguajes dependientes, por lo que se pueden crear con cualquier lenguaje de servidor como JavaScript (Node), PHP, Python o Java, por nombrar algunos, la respuesta puede ser devuelta en cualquier formato, pero por compatibilidad se utiliza JSON o XML.

/* Consumir APIs */

Consumir APIs

- Las APIs se encargan de manejar nuestras peticiones desde el browser y devolvernos los datos que les solicitamos para que podamos mostrarlos en la web.
- Para consultar y enviar datos utilizaremos la página JSONPlaceholder, que nos permite extraer y enviar datos, además de utilizarlo como API REST para hacer pruebas o prototipos.



Resources

JSONPlaceholder comes with a set of 6 common resources:

/posts	100 posts
/comments	500 comments
/albums	100 albums
/photos	5000 photos
/todos	200 todos
/users	10 users

Note: resources have relations. For example: **posts** have many **comments**, **albums** have many **photos**, ... see below for routes examples.

Fuente: Desafío Latam

Ejercicio guiado: Solicitud a una API

Ejercicio guiado: Solicitud a una API

Conectarse a la sección de usuarios (/users) de la JSONPlaceholder y solicitar los datos que contenga esa dirección, mediante el método fetch y empleando promesas, traer la respuesta y mostrarla en la consola del navegador web.

Para realizar este ejercicio, debemos seguir los siguientes pasos:

Paso 1: Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js. Mientras que en el index.html, debes escribir la estructura básica de un documento HTML como se muestra a continuación:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Conexión API</title>
</head>
<body>
   <h4>Conexión API - Fetch</h4>
   <script src="script.js"></script>
</body>
</html>
```

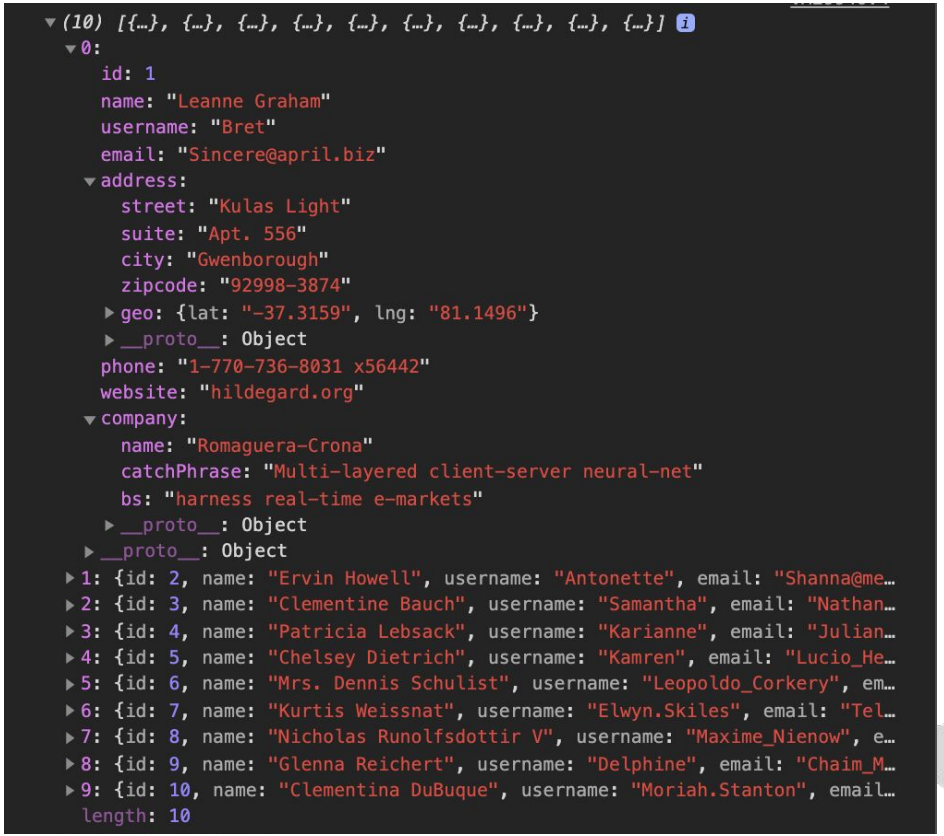
Ejercicio guiado: Solicitud a una API

Paso 2: En el archivo script.js, implementando directamente el método fetch, pasamos la URL como parámetro, luego, como este método retorna una promesa, entonces debemos utilizar el then para recibir la respuesta, pero el método fetch tiene una característica en especial y es que todas las respuestas se deben pasar a JSON, implementando el método ".json()", al finalizar de recibir y transformar la respuesta al formato JSON, se ejecuta nuevamente una promesa con el then para ahora si mostrar el resultado en la consola del navegador web.

```
fetch('https://jsonplaceholder.typicode.com/users')
    .then(response => response.json())
    .then(json => console.log(json))
```

Ejercicio guiado: Solicitud a una API

Paso 3: Ejecutar el index.html en el navegador web y observar el resultado en la consola, nos debe mostrar un arreglo con 10 objetos. Cada objeto debe contener la información (ficticia) de cada usuario.



{desafío} latam_

/* Solicitud de múltiples APIs en secuencia */

Solicitud de múltiples APIs en secuencia

Existen ocasiones que para solicitar datos a una API, dependemos de algunos valores que nos retornan otras APIs y para esto necesitamos solicitar datos a múltiples APIs en forma secuencial. Para ello, implementaremos el método fetch, pero esta vez trabajado de forma asincrónica con Async / Await.

/* Solicitud de múltiples APIs a la vez */

Solicitud de múltiples APIs a la vez

A veces es necesario realizar múltiples llamadas a la vez para obtener lo que necesitamos mostrar, tal cual como cuando explicamos Promise.all en Promesas (Promise) de la clase anterior.

/* Solicitud de múltiples APIs a la vez, utilizando la información */

Solicitud de múltiples APIs a la vez, utilizando la información

Hasta el momento, ya logramos conectarnos a una API y traer la información, mostrando todo el contenido en la consola del navegador web a la vez, pero sin especificar o dividir. Por lo tanto, en esta sección y partiendo del ejemplo anterior, veamos cómo utilizar estos datos que nos retornó la petición para mostrarlos separados y ordenadamente.

¿Te quedó alguna duda en relación a los temas que acabamos de aprender?

Resumiendo

- API: Interfaz de Programación de Aplicaciones (Application Programming Interface).
- Es una especificación formal sobre cómo un módulo de un software se comunica o interactúa con otro.
- Una de las principales funciones de las API es poder facilitar el trabajo a los desarrolladores, ahorrar tiempo y dinero.
- Las APIs se encargan de manejar nuestras peticiones desde el browser y devolvernos los datos que les solicitamos para que podamos mostrarlos en la web.

Próxima sesión...

Presentación - Manejo de errores

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales