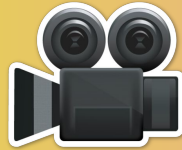


***RECUERDA PONER A GRABAR LA
CLASE***





¿DUDAS DEL ON-BOARDING?

MIRALO AQUI



Clase 07. REACT JS

CONSUMIENDO API'S



OBJETIVOS DE LA CLASE

- Identificar distintos paradigmas de intercambio de datos.
- Consumir recursos vía llamados a API's.

GLOSARIO:

Clase 6

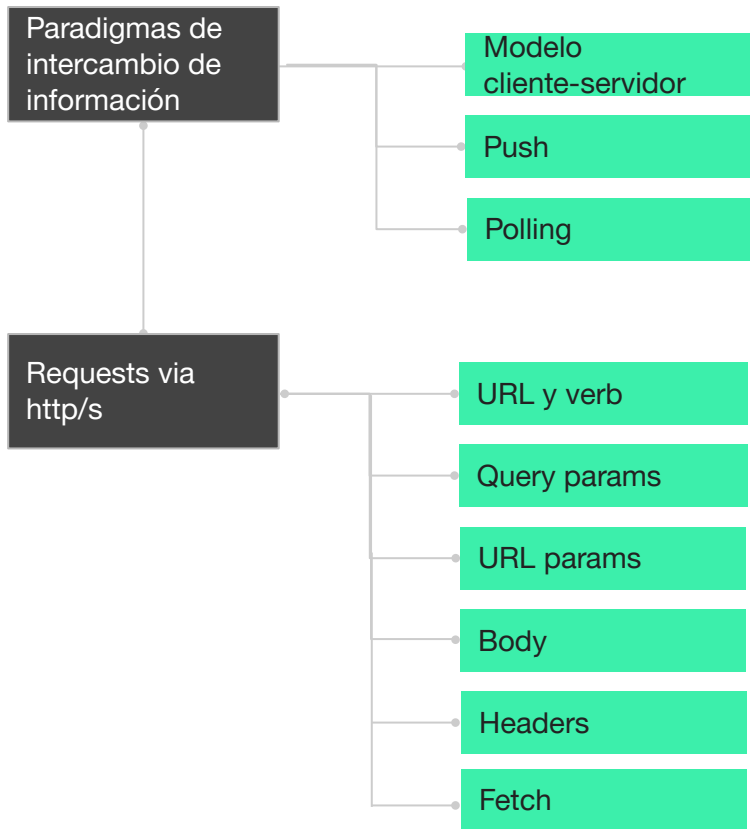
Promise: es un objeto que permite representar y seguir el ciclo de vida de una tarea/operación (función).

Map: es un método que nos permite generar un nuevo array, tomando de base otro, y utilizando una función transformadora.

MAPA DE CONCEPTOS

MAPA DE CONCEPTOS CLASE 7

¡Para
recordar!



CRONOGRAMA DEL CURSO

Clase 6



Promises, asincronía y MAP



EJEMPLOS EN VIVO



MOCK ASYNC SERVICE



CATÁLOGO CON MAPS Y PROMISES

Clase 7



Consumiendo API's



EJEMPLOS EN VIVO



FETCH API-CALL



DETALLE DE PRODUCTO A Y B

Clase 8



Routing y Navegación



EJEMPLOS EN VIVO



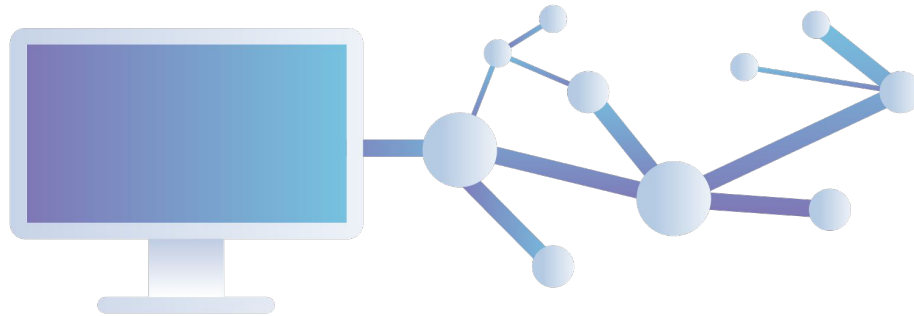
AGREGAR UN ROUTER A TU APP



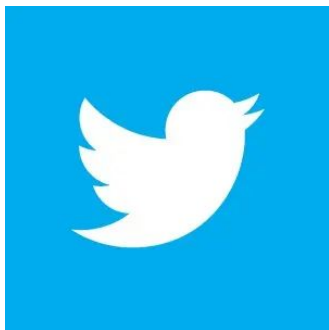
PRIMERA ENTREGA DEL PROYECTO FINAL

PARADIGMAS DE INTERCAMBIO DE INFORMACIÓN

La mayoría de las aplicaciones suelen generar **experiencias de usuario** gracias a que se pueden conectar a un conjunto de servicios de datos

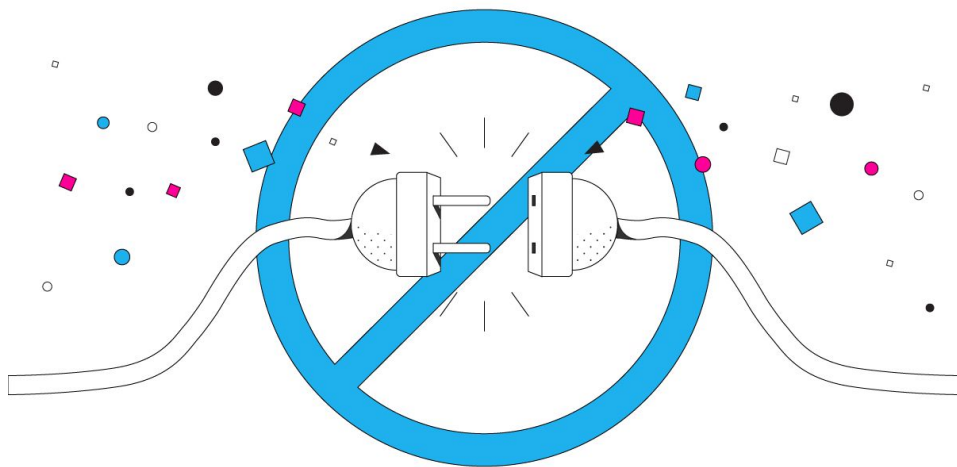


Esta conexión le permite, por ejemplo, a un user de **Instagram**, acceder a su perfil y ver sus fotos.

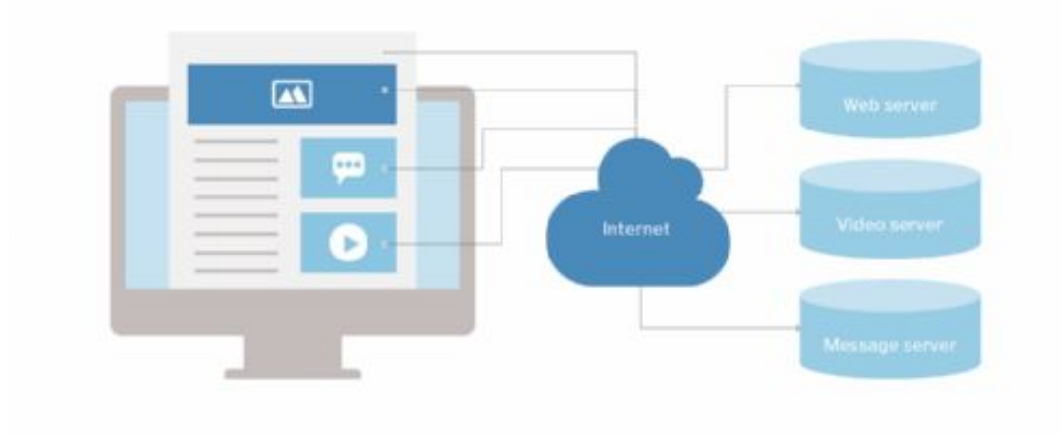


A un user de **Twitter** le permitiría publicar un tweet, transmitiendo los 280 caracteres permitidos por cada mensaje.

Carecer de una conexión a un servicio de datos es un gran limitante para prácticamente cualquier **app** que busque vender, o conectar personas.



El consumo y la transferencia han evolucionado mucho desde su creación.



Las **mejoras tecnológicas** permitieron saltos agigantados en el terreno de las **aplicaciones web y mobile**:

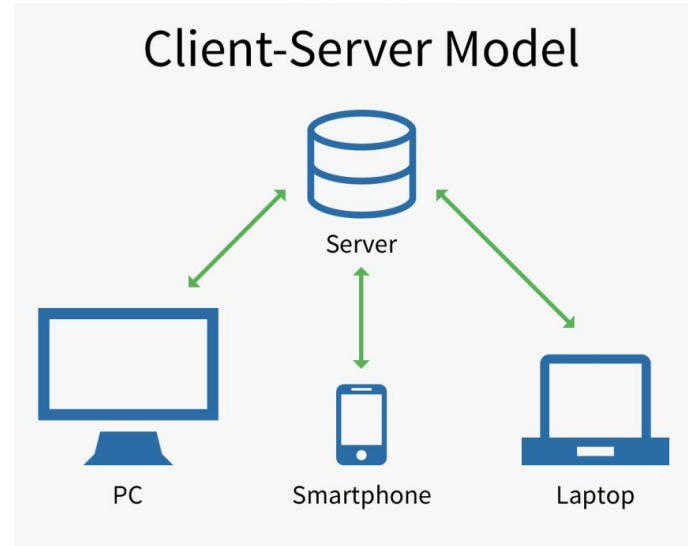
- Velocidad de transferencia.
- Tolerancia a fallos.
- Seguridad.

2G - 1993	< 14.4 Kbps
3G - 2001	< 3.1 Mbps
4G - 2009	< 100 Mbps
5G - Jul-2020	< 400 Mbps

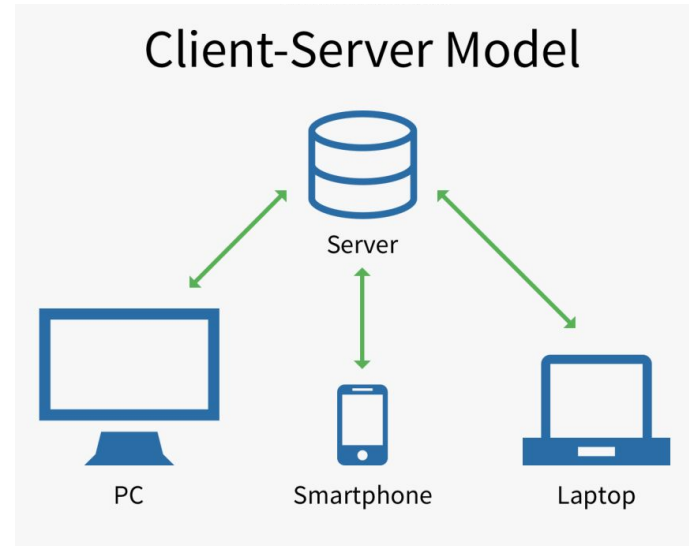
MODELO CLIENTE-SERVIDOR

Independientemente de esto, hay algo que parece no cambiar hasta el momento, y es que hay dos protagonistas:

- **Cliente (consumidor)**
- **Servidor (proveedor)**

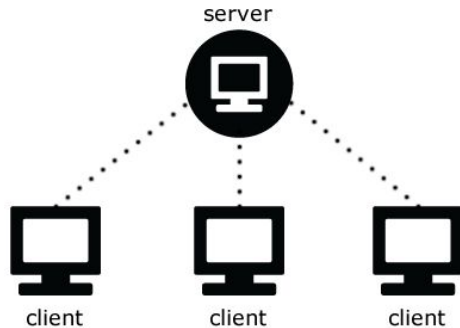


Este modelo establece que los distintos consumidores **se identifican entre ellos**, y acuerdan una **manera de transferir** la información.



Lo más importante a recordar es que la variación más notable que podemos identificar queda definida por:

¿Quién es el que inicia la operación y cómo sincronizan?

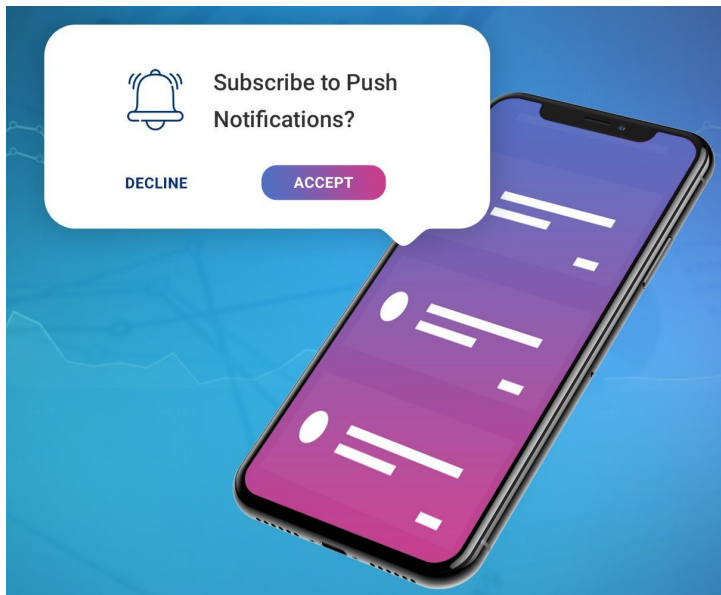


El **cliente** inicia:

1. El cliente solicita info.
2. El servidor envía la respuesta.
3. Fin de la comunicación.

PUSH

Si invertimos la lógica, se la conoce como **PUSH**.

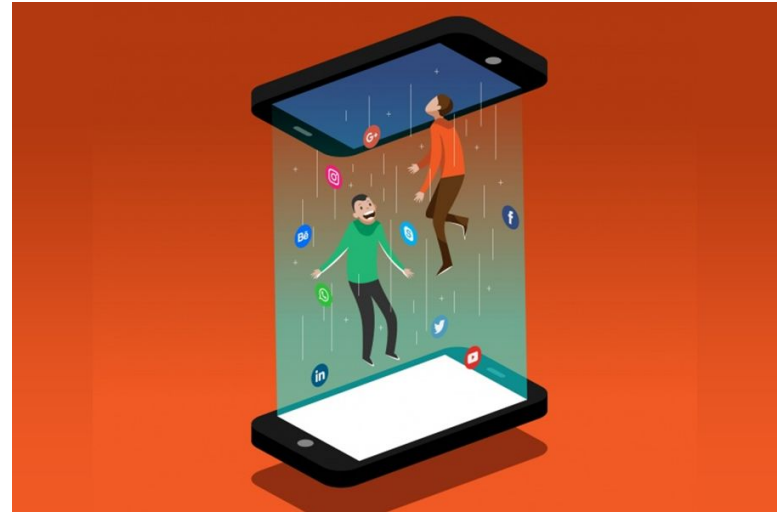


El **servidor** inicia:

1. Cliente se suscribe.
2. El servidor elige el momento del inicio de la transferencia, y la envía a un servicio.
3. El servicio se la provee al cliente.

PUSH

Push nace para poder generar **engagement**, y lograr que los usuarios recuerden que nuestra app existe, y que puede proveerles con algo que les pueda **interesar**, en el **momento** en el que el servidor considere **oportuno**.



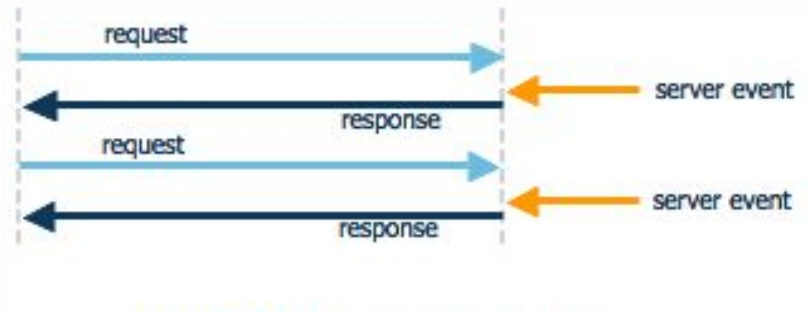
POLLING

POLLING

De no utilizar **PUSH**, deberíamos configurar nuestros **client** para que estén constantemente preguntando:

- ¿Tienes algo nuevo para mi?
- ¿Tienes algo nuevo para mi?
- ¿Tienes algo nuevo para mi?

De manera indefinida, sin optimizar los recursos/datos del usuario y nuestro servidor.

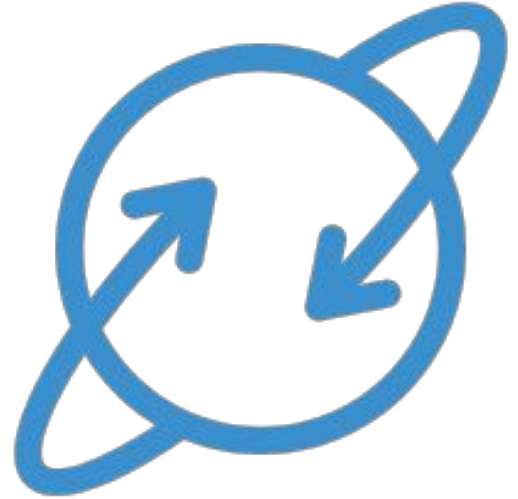


REQUESTS VIA HTTP/S

REQUESTS VIA HTTP/S

Vienen para ayudarnos a realizar una solicitud a un servidor, y nos permiten **establecer un protocolo de transferencia** definido por:

- Dirección/URL.
- Verbo (**GET, POST, PUT, DELETE +**).
- Parámetros: vía query o url.
- Headers.
- Body (contenido en un **POST**).



URL Y VERB

URL Y VERB

Nos permiten definir una manera de explicarle al servidor la dirección y nuestras intenciones:

- **GET**: quiero obtener.
- **POST**: quiero crear.
- **PUT**: quiero crear o actualizar.
- **PATCH**: quiero alterar parcialmente.
- **DELETE**: quiero eliminar.

GET	/pet/{petId}	Find pet by ID
PUT	/pet	Update an existing pet
DELETE	/pet/{petId}	Deletes a pet
POST	/pet/{petId}/uploadImage	uploads an image

URL Y VERB

Ningún verbo representa una seguridad y/u obligación.
Pero si el servidor y el consumidor los respetan, se pueden lograr algunas mejoras como por ejemplo:

El navegador sabe que un **POST** no debería ser cacheado, si hacemos un **GET** y fuera cacheable el navegador podrá cachearlo, pero nunca lo hará con un recurso con verbo **POST**.



QUERY PARAMS

QUERY PARAMS

Nos permiten incluir **en la dirección** información que se usa para especificarle al receptor parámetros para efectuar una búsqueda, son más comunes para buscar recursos que no tengo la seguridad de que existan:

<https://www.google.com.ar/search?q=coderhouse>

Se puede leer como:

- busca en **google.com.ar**
- utilizando **https...**
- el recurso **search** (resultados de búsqueda) ...
- que contengan la palabra (**q = query**) 'coderhouse'

URL QUERY PARAMS

- Se separa la URL de los parámetros utilizando un signo de pregunta ?
- Cada parámetro tendrá **key=value & key2=value2**
- Cada parámetro se puede separar por &
- <http://url.com/find?type=order&id=1234>

URL PARAMS

URL PARAMS/SEGMENT

Son una convención para incluir el identificador del recurso dentro de la misma url, son más comunes cuando ya se conoce el recurso específico que se buscará.

<https://myapp.coder/student/1234>

Se puede leer como:

- busca en **myapp.coder**
- utilizando **https...**
- el recurso **student**
- con id 1234

<https://myapp.coder/student/1234/courses>

Se puede leer como:

- busca en **myapp.coder**
- utilizando **https...**
- el recurso **courses**
- Únicamente para **student** 1234



RECURSOS/RESTFUL

- Cuando se crea y provee un servicio basado y pensado en términos de **recursos**, y se respetan las convenciones de verbo/método y código de respuesta, estamos frente a un diseño arquitectural de tipo **REST**.
- Si además transferimos **javascript o xml**, es conocido como **AJAX**.

BODY

BODY

Se utiliza para **transferir piezas de información entre el cliente y el servidor.**

```
POST /create-user HTTP/1.1
```

```
Host: localhost:3000
```

```
Connection: keep-alive
```

```
Content-type: application/json
```

} header

```
{ "name": "John", "age: 35 }
```

} body

HEADERS

HEADERS

Se usan para:

- Definir las respuestas soportadas, requeridas o preferidas.
- Agregar información extra:
 - Auth tokens, cookies.
 - Lenguaje preferido.
 - Si acepta contenido cacheado.
- Lo que quieras en forma de texto.

```
Request URL: https://s7.addthis.com/l10n/client.es.min.json
Request Method: GET
Status Code: 200
Remote Address: 23.192.128.32:443
Referrer Policy: no-referrer-when-downgrade
```

Response Headers (14)

Request Headers

```
:authority: s7.addthis.com
:method: GET
:path: /l10n/client.es.min.json
:scheme: https
accept: */*
accept-encoding: gzip, deflate, br
accept-language: es-419,es;q=0.9,pt-BR;q=0.8,pt;q=0.7,en;q=0.6
cache-control: no-cache
origin: https://www.stanfordchildrens.org
pragma: no-cache
referer: https://www.stanfordchildrens.org/es/service/autism/au
```

HEADERS

The screenshot shows the Chrome DevTools Network tab. The left pane lists network requests, with 'client.es.min.json' selected and circled in red. The right pane shows the 'Headers' tab for this request. The 'Request URL' is 'https://s7.addthis.com/l10n/client.es...' and is circled in red. The 'Request Method' is 'GET'. The 'Status Code' is '200' and is circled in red. The 'Remote Address' is '23.192.128.32:443'. The 'Referrer Policy' is 'no-referrer-when-downgrade'. Below these, there are sections for 'Response Headers (14)' and 'Request Headers'. The 'Request Headers' section is expanded, showing: ':authority: s7.addthis.com', ':method: GET', ':path: /l10n/client.es.min.json', ':scheme: https', and 'accept: */*'. The 'Response Headers' section is collapsed.

Los puedes leer desde la consola de Chrome e identificar todas sus partes. Habrá headers del request y del response (los envía el servidor).

REQUESTS EN EL BROWSER

FETCH

UTILIZANDO FETCH

Podemos hacer un request de manera simple, utilizando **Fetch API**.

Esta nos provee con una promesa, que se resuelve al terminar el request.

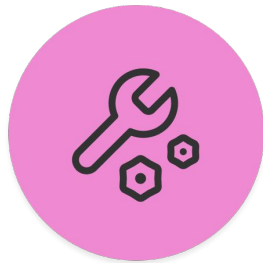
Esta respuesta es una **promise**, que nos permite acceder a la respuesta.

```
fetch('https://api.coder.com.ar/user/1234')  
  .then(function(response) {  
    return response.json();  
  })  
  .then(function(user) {  
    console.log(user);  
  });
```

VAMOS AL CÓDIGO



CODER HOUSE



FETCH API-CALL

Crea en [Stackblitz](https://stackblitz.com) una app de React que al iniciar utilice Fetch API.



¡A PRACTICAR!

Crea en [Stackblitz](#) una app de **React** que al iniciar (utilizando un mount effect) utilice **Fetch API** para mostrar un listado de productos consumidos de la API de [Mercadolibre](#) o Pokémons de la [Poké API](#) y muestre los nombres de los primeros diez (¡si quieres mapear más datos hazlo!)

Cuentas con 20 minutos para realizarlo



BREAK

**Si adquiriste un servicio de talento,
recordá chequear tu correo de
spam, no deseado, publicidad y/o
social.**

En caso de no haberlo recibido, escribinos a talento@coderhouse.com

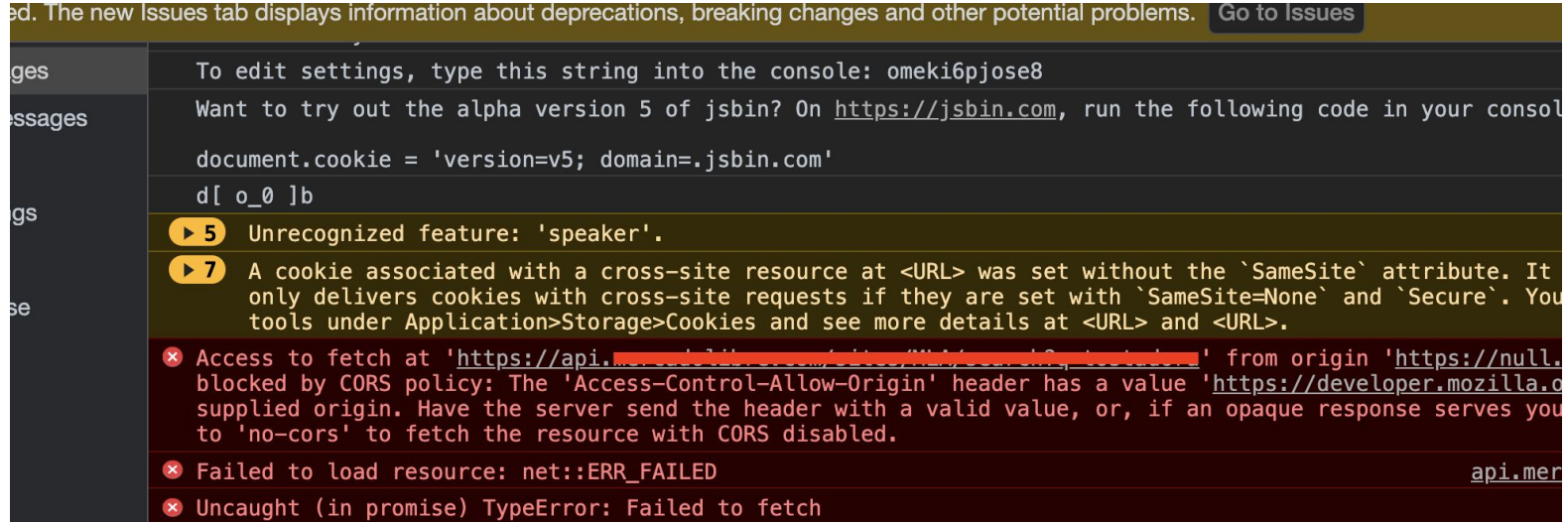
¡5/10 MINUTOS Y VOLVEMOS!

CODER HOUSE

CORS

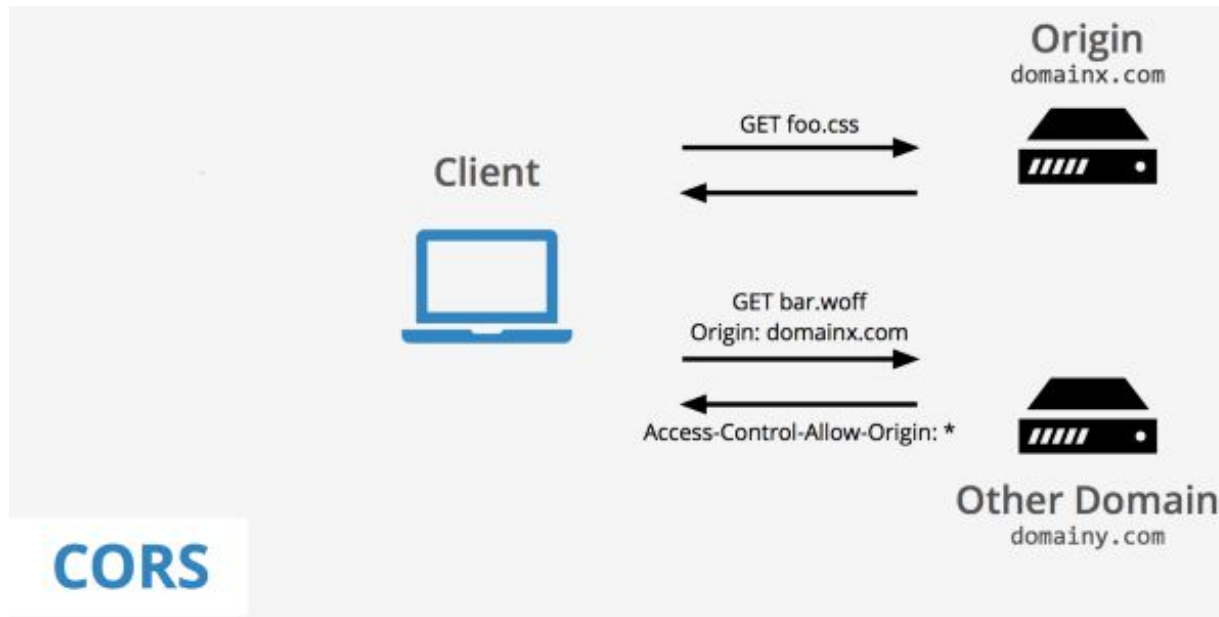
CORS

Al realizar un request, nos podemos encontrar con este error/problema:



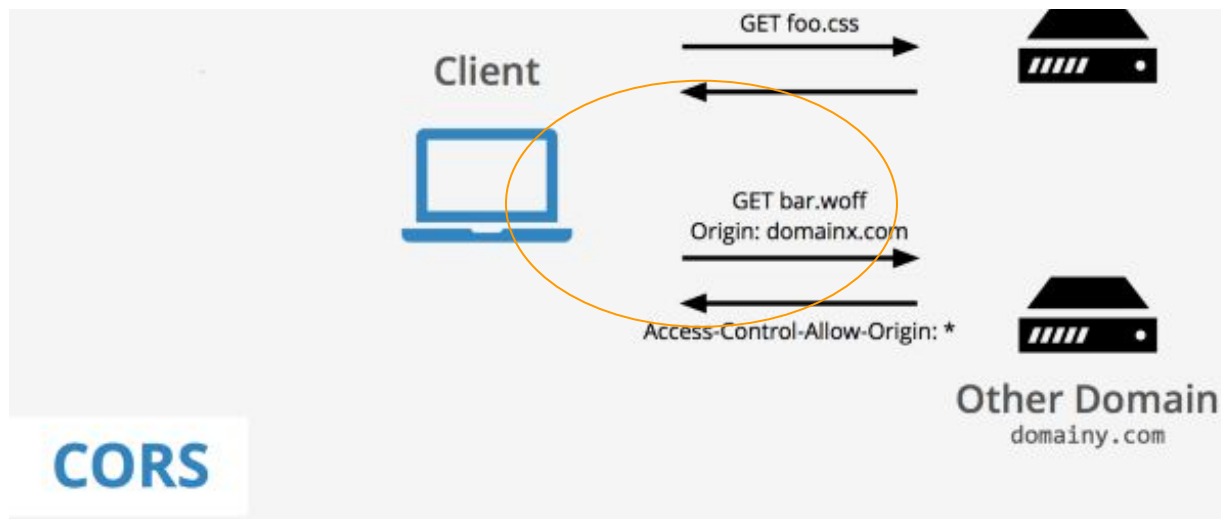
CORS: PREFLIGHT

Antes de enviar un request entre dominios como en el siguiente ejemplo, el browser envía un request options, llamado **pre-flight**:



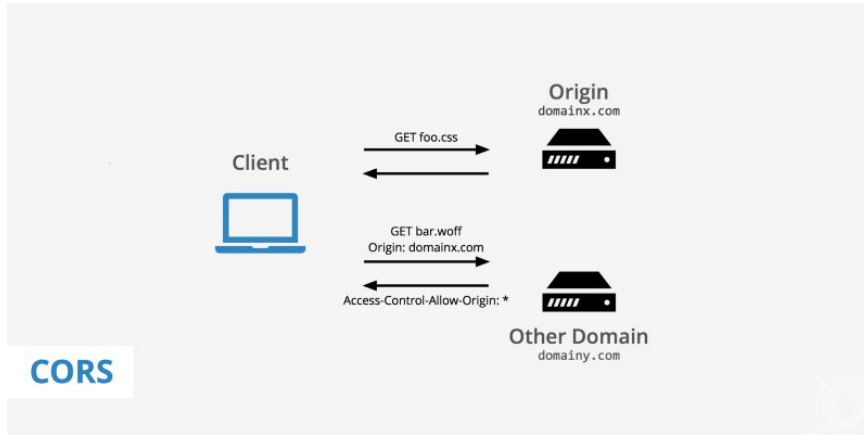
CORS: PREFLIGHT

En este request, **se le pregunta al otro dominio** si acepta requests provenientes de un dominio distinto (cross):



CORS

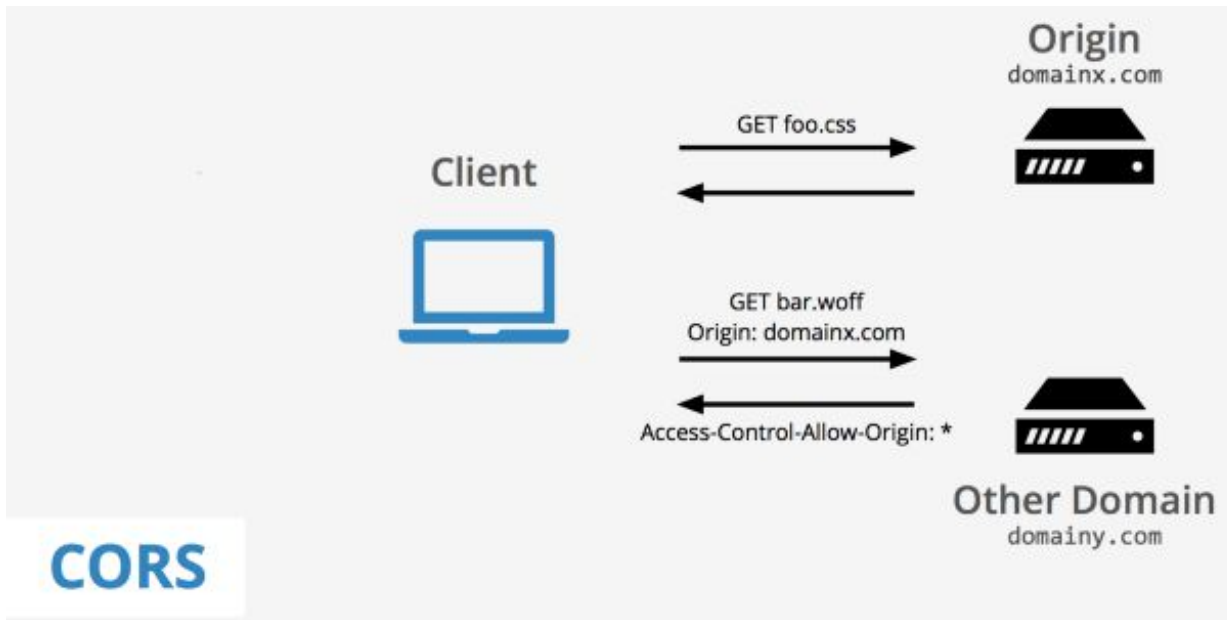
Más que un problema, es un bloqueo de seguridad efectuado **por el navegador** (Chrome, Mozilla, etcétera).



Esto es para que **por default**, el **javascript** alojado en un dominio **misitio.com** sólo pueda ejecutar llamados http hacia **misitio.com**. Previene algunos problemas de seguridad.

CORS

Usualmente ocurre cuando tengo un servidor para mi React App alojado en **`https://localhost:3000`**, tratando de hacer un request contra una api levantada en **`https://localhost:3001`**, u **`otrodominio.com`**



CORS

El modo de solucionarlo es configurar al otro servidor para que admita **CORS** respondiendo el siguiente header ante un **OPTIONS** preflight

```
Access-Control-Allow-Origin: *
```

ó

```
Access-Control-Allow-Origin: https://localhost:3000
```



CORS

- Estos headers se deben activar ante un verbo **OPTIONS**, aunque por comodidad podemos también activarlos para otros verbos.
- Podemos activar uno o todos (*) los dominios.
- Configurarlos bien nos puede ayudar a resolver algunos inconvenientes durante el desarrollo.

¿POR QUÉ TANTA VUELTA CON EL TEMA?

Los mejores servicios y/o integraciones proveen integraciones mediante API's Rest usando **http/s**.



Son el canal transaccional **más importante del mundo**.

Nos conectan a soluciones que puedan complementar nuestro **modelo de negocio** y suman **adoptabilidad**.



EJEMPLOS REALES



GET

https://graph.instagram.com/{user-id}/media?access_token=123434

(Nos permite leer información de users de instagram vía API)

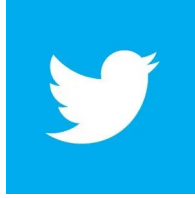


POST

https://api.mercadopago.com/v1/payments?access_token=123434

(Nos permite usar http para habilitar el pago a nuestros usuarios)

OTROS EJEMPLOS



CODER HOUSE



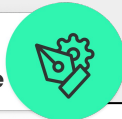
DETALLE DEL PRODUCTO

1. Crea tu componente **ItemDetailContainer**, con la misma premisa que ItemListContainer.
2. Luego, crea tu componente **ItemDetail.js**.

DETALLE DEL PRODUCTO - PUNTO 1

Formato: link al último commit de tu repositorio en GitHub. Debe tener el nombre "Idea+Apellido".

Desafío
entregable



>> Consigna: crea tu componente `ItemDetailContainer`, con la misma premisa que `ItemListContainer`.

>>Aspectos a incluir en el entregable:

Al iniciar utilizando un efecto de montaje, debe llamar a un async mock, utilizando lo visto en la clase anterior con `Promise`, que en 2 segundos le devuelva un 1 ítem, y lo guarde en un estado propio.

>>Ejemplo inicial:

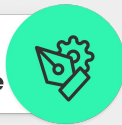
```
const getItem = () => { /* Esta función debe retornar la promesa que resuelva con delay */ }  
  
function ItemDetailContainer() {  
  // Implementar mock invocando a getItem() y utilizando el resolver then  
  return /* JSX que devuelva un ItemDetail (punto 2) */  
}
```

CODER HOUSE

DETALLE DEL PRODUCTO - PUNTO 2

Formato: link al último commit de tu repositorio en GitHub. Debe tener el nombre “Idea+Apellido”.

Desafío
entregable



>> Consigna: crea tu componente ItemDetail.js.

>>Aspectos a incluir en el entregable:

ItemDetail.js, que debe mostrar la vista de detalle de un ítem incluyendo su descripción, una foto y el precio.

>>Ejemplo inicial:

```
function ItemDetail({ item }) {  
  
  return <>  
    ...  
    // Desarrolla la vista de detalle expandida del producto con su imagen título,  
    descripción y precio  
    ...  
  </>;  
  
}
```

CODER HOUSE

DETALLE DEL PRODUCTO - PUNTO 2

Ejemplo:



Nuevo | 14484 vendidos



Samsung Galaxy S20 FE 128 GB
cloud navy 6 GB RAM

★★★★★ 2172 opiniones

RECOMENDADO en Celulares

~~\$ 86.999~~

\$ 76.999 11% OFF

en 9x \$ 8.555** sin interés

[Ver los medios de pago](#)

Memoria interna: **128 GB**

128 GB

Es Dual SIM: No

No

Memoria RAM: **6 GB**

6 GB

Color: **Cloud navy**



Lo que tenés que saber de este producto

- Dispositivo liberado para que elijas la compañía telefónica que prefieras.

Llega gratis mañana **FULL** ✓

Comprando dentro de las próximas
9 h 21 min

[Enviar a V. Gerde 114](#)

Retíralo gratis a partir de mañana
en correos y otros puntos

Comprando antes de las 23hs.

[Ver en el mapa](#)

Tienda oficial **Samsung**
139.045 ventas

Stock disponible

Cantidad: **1 unidad** ▼ (837 disponibles)

Comprar ahora

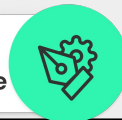
Devolución gratis. Tenés 30 días desde
que lo recibís.

Compra Protegida, recibí el producto
que esperabas o te devolvemos tu
dinero.

Mercado Puntos. Sumás 769 puntos.

12 meses de garantía de fábrica.

**Desafío
entregable**



CODER HOUSE

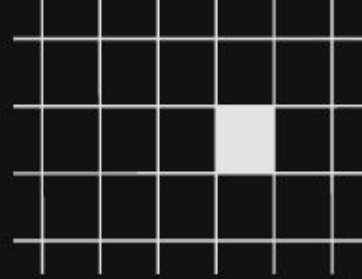
¿PREGUNTAS?



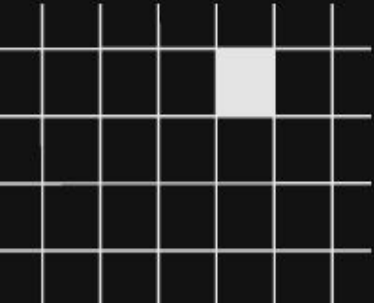
¡DESCUENTO EXCLUSIVO!



[Quiero saber más](#)



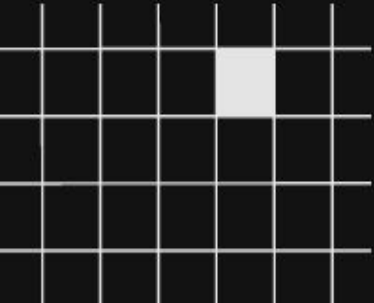
¡Completa tu carrera y potencia tu desarrollo profesional!
*Ingresando el cupón **CONTINUATUCARRERA** tendrás un*
descuento para inscribirte en el próximo nivel. Puedes
acceder directamente desde la plataforma, entrando en la
sección ["Cursos y Carreras"](#).





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Paradigmas de comunicación
 - HTTP / REST / Fetch
 - CORS
- 



OPINA Y VALORA ESTA CLASE