



Clase 07. React JS

Consumiendo APIs REST

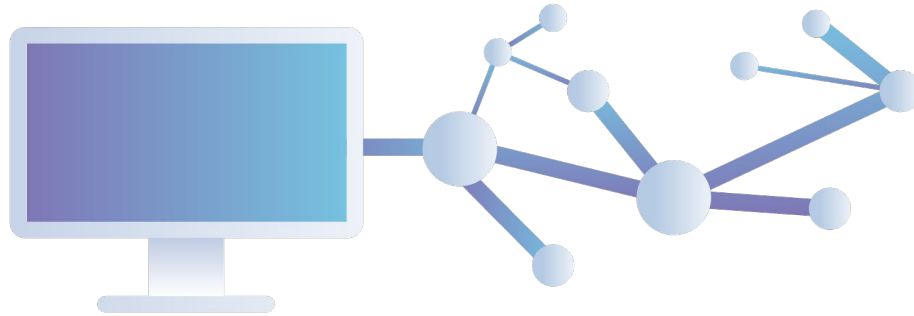


OBJETIVOS DE LA CLASE

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

PARADIGMAS DE INTERCAMBIO DE INFORMACIÓN

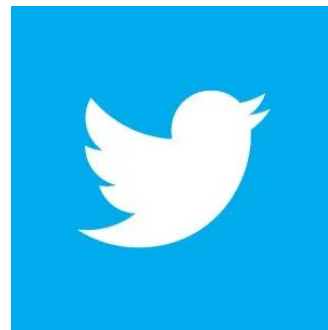
La mayoría de las aplicaciones suelen generar **experiencias de usuario** gracias a que se pueden conectar a un conjunto de servicios de datos



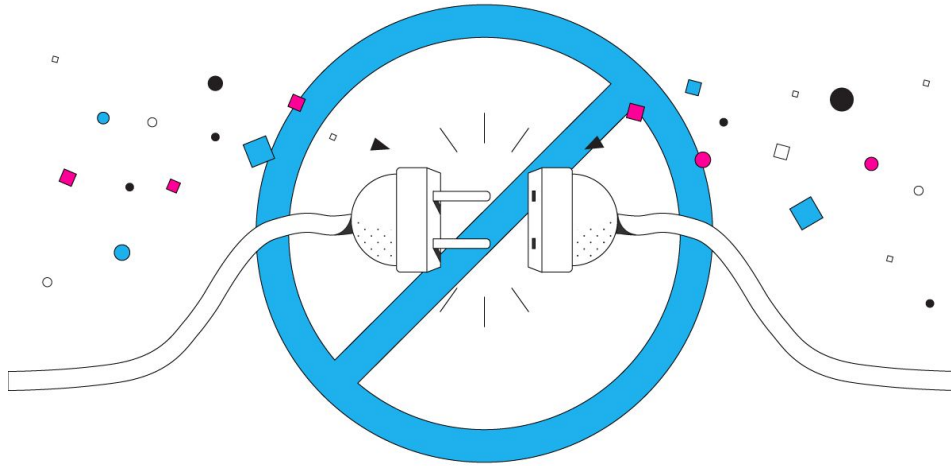
Esta conexión le permite -por ejemplo- a un user de **Instagram** acceder a su perfil y ver sus fotos



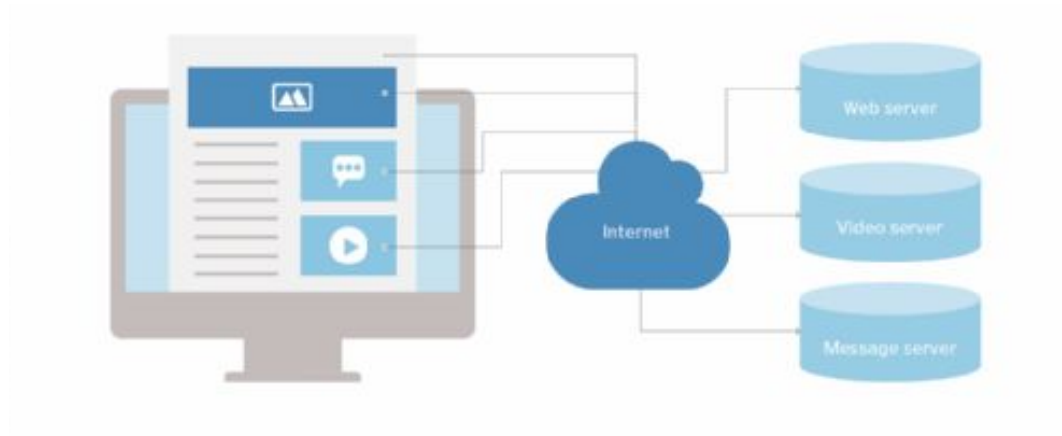
A un user de **Twitter** le permitiría publicar un tweet transmitiendo los 280 caracteres permitidos por cada mensaje



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:

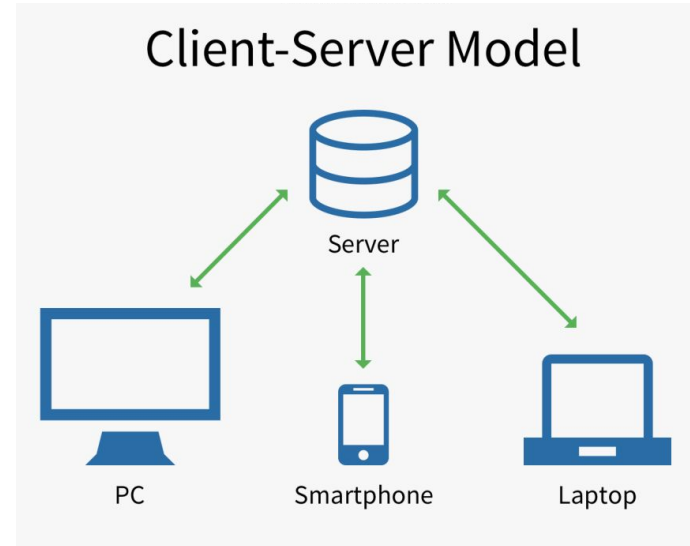
- Velocidad de transferencia
- Tolerancia a fallos
- Seguridad

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

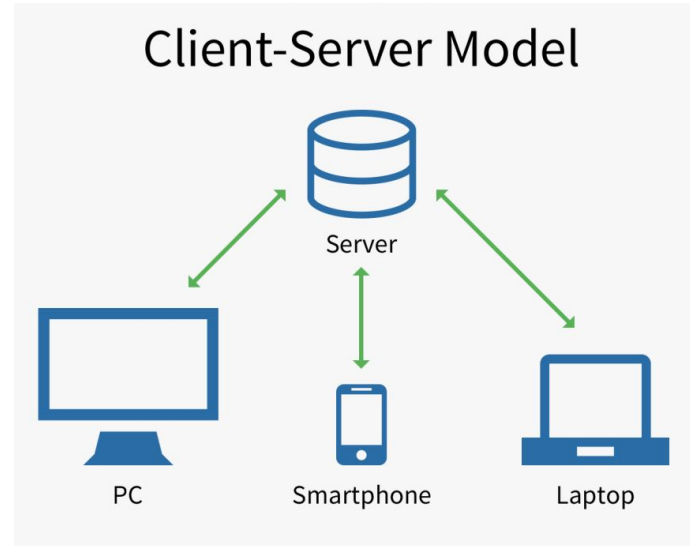
Modelo cliente-servidor

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

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

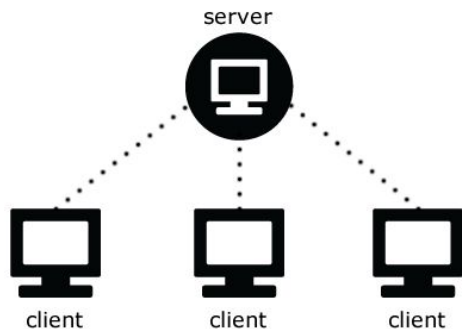


Este modelo establece que los distintos consumidores **se identifican entre ellos** y acuerdan una **manera de transferir** la información.



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?

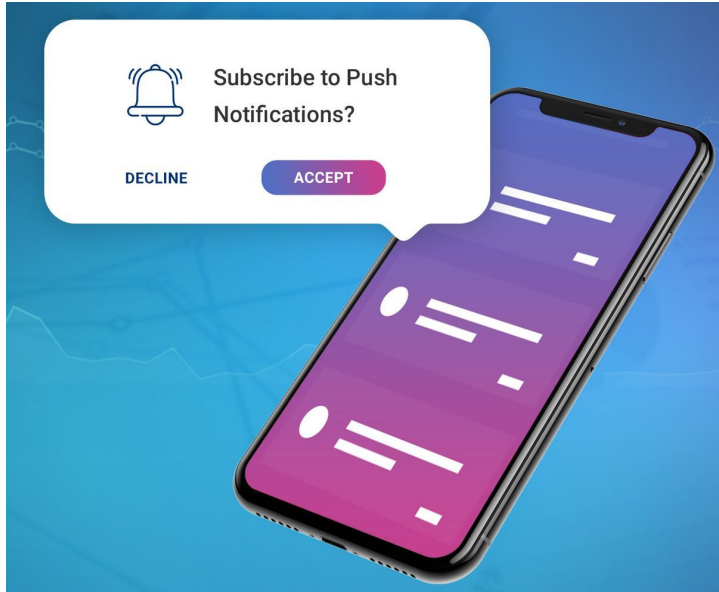


El **cliente** inicia:

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**

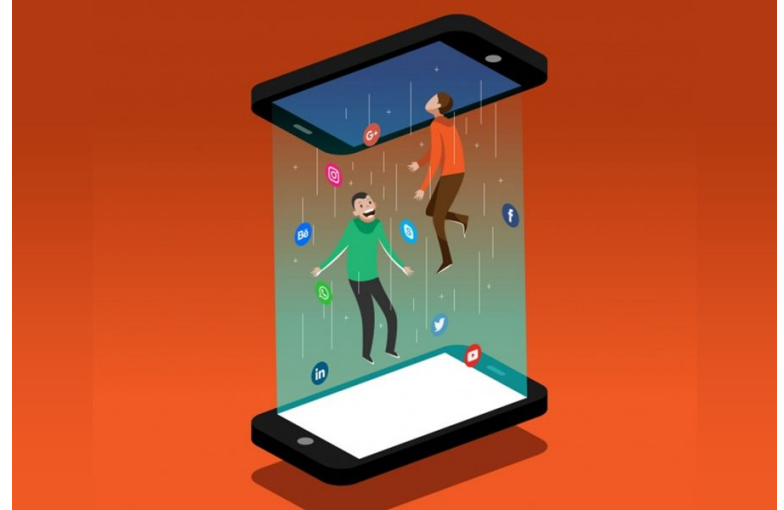


El **servidor** inicia:

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**



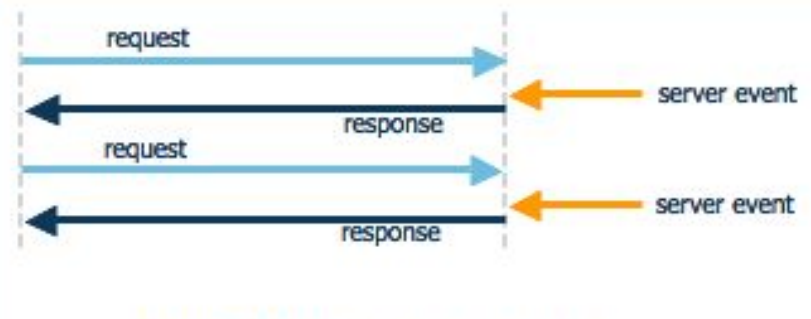
Polling

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

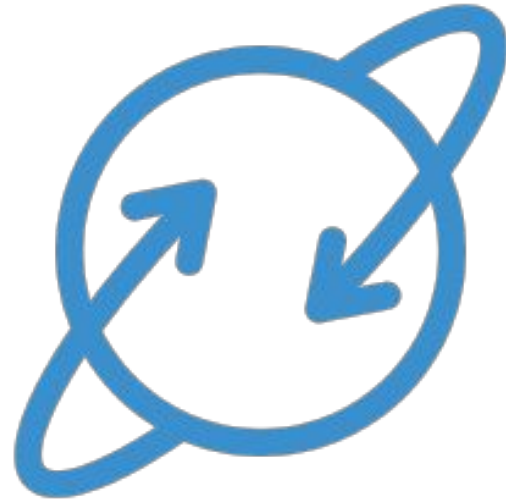


REQUESTS VIA HTTP/S

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 details for this request, with the 'Headers' tab active. The 'Request URL' is circled in red, showing 'https://s7.addthis.com/l10n/client.es...'. The 'Request Method' is 'GET'. The 'Status Code' is '200', also circled in red. Below, the 'Request Headers' are listed, including 'authority: s7.addthis.com', 'method: GET', 'path: /l10n/client.es.min.json', 'scheme: https', and 'accept: */*'. The 'Response Headers' section is also visible but 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**.

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

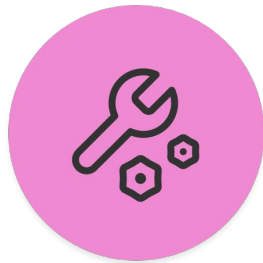
Esta respuesta es una **promise** que nos permite acceder a la respuesta.

```
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](#) 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** y muestre los nombres de los primeros diez (¡si quieres mapear más datos hazlo!)

Tiempo: 20 minutos

¿PREGUNTAS?



BREAK



CODER HOUSE

CORS

CORS

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

```
ed. The new Issues tab displays information about deprecations, breaking changes and other potential problems. Go to Issues
```

```
ges      To edit settings, type this string into the console: omeki6p jose8
```

```
essages  Want to try out the alpha version 5 of jsbin? On https://jsbin.com, run the following code in your console
```

```
document.cookie = 'version=v5; domain=.jsbin.com'
```

```
d[ o_0 ]b
```

```
gs       ▶ 5 Unrecognized feature: 'speaker'.
```

```
se       ▶ 7 A cookie associated with a cross-site resource at <URL> was set without the `SameSite` attribute. It
```

```
only delivers cookies with cross-site requests if they are set with `SameSite=None` and `Secure`. You
```

```
tools under Application>Storage>Cookies and see more details at <URL> and <URL>.
```

```
✖ Access to fetch at 'https://api.mercedes-benz.com/api/v1/vehicles?test=true' from origin 'https://null.
```

```
blocked by CORS policy: The 'Access-Control-Allow-Origin' header has a value 'https://developer.mozilla.org
```

```
supplied origin. Have the server send the header with a valid value, or, if an opaque response serves you
```

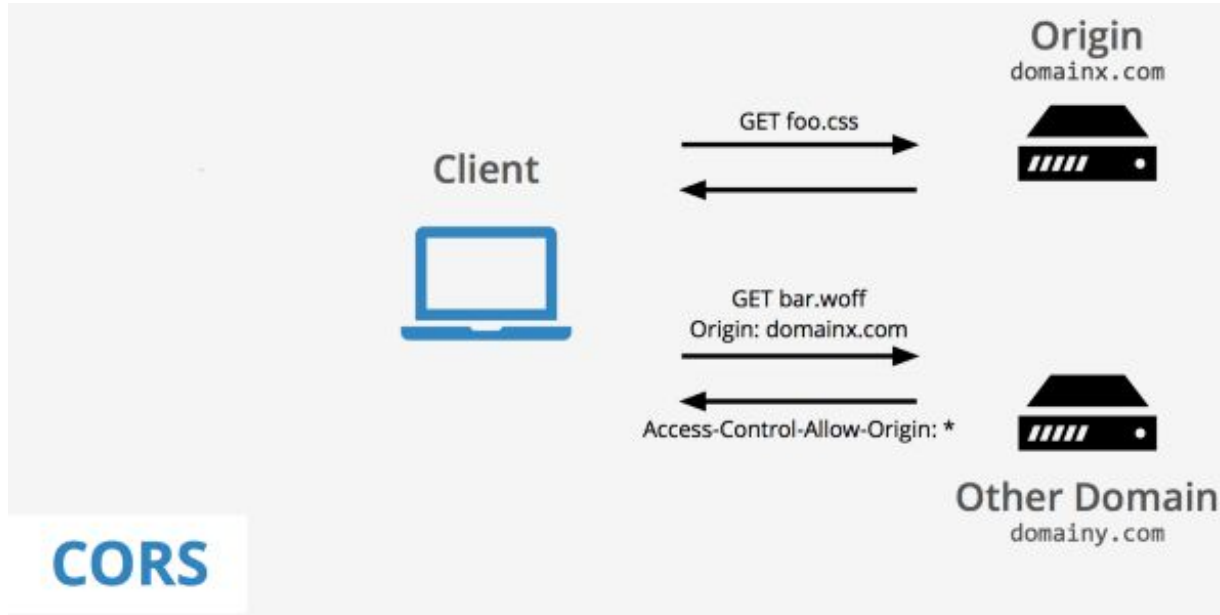
```
to 'no-cors' to fetch the resource with CORS disabled.
```

```
✖ Failed to load resource: net::ERR_FAILED api.mer
```

```
✖ Uncaught (in promise) TypeError: Failed to fetch
```

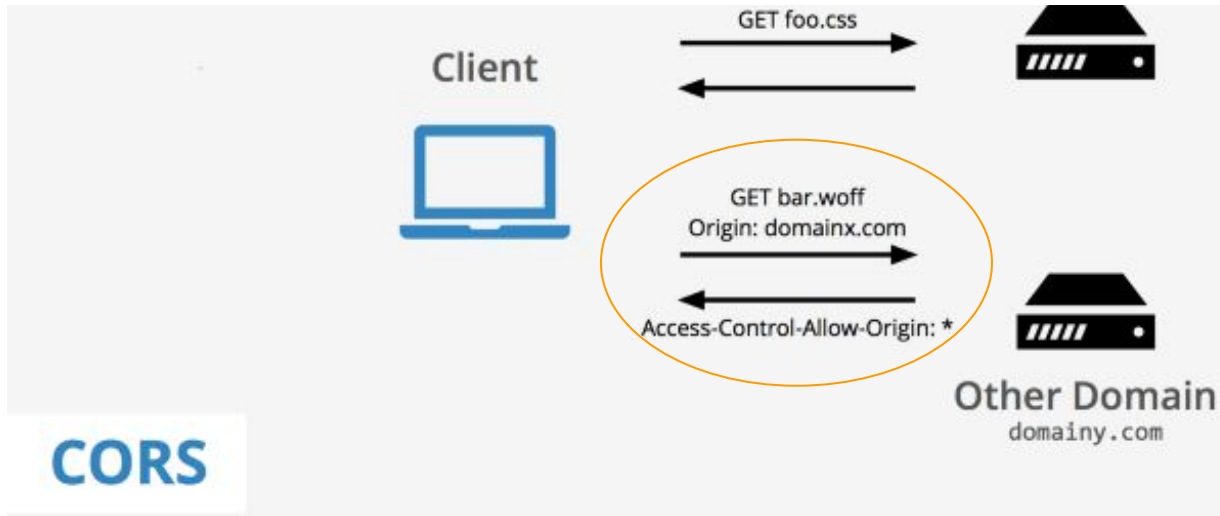
CORS: PREFLIGHT

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



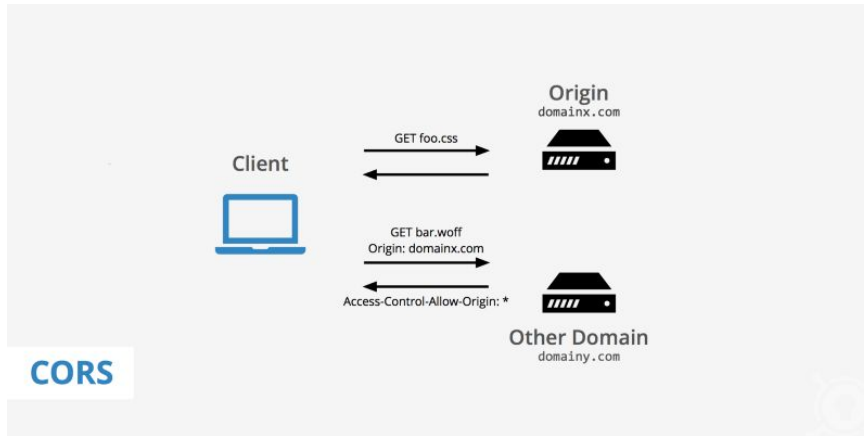
CORS: PREFLIGHT

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



CORS

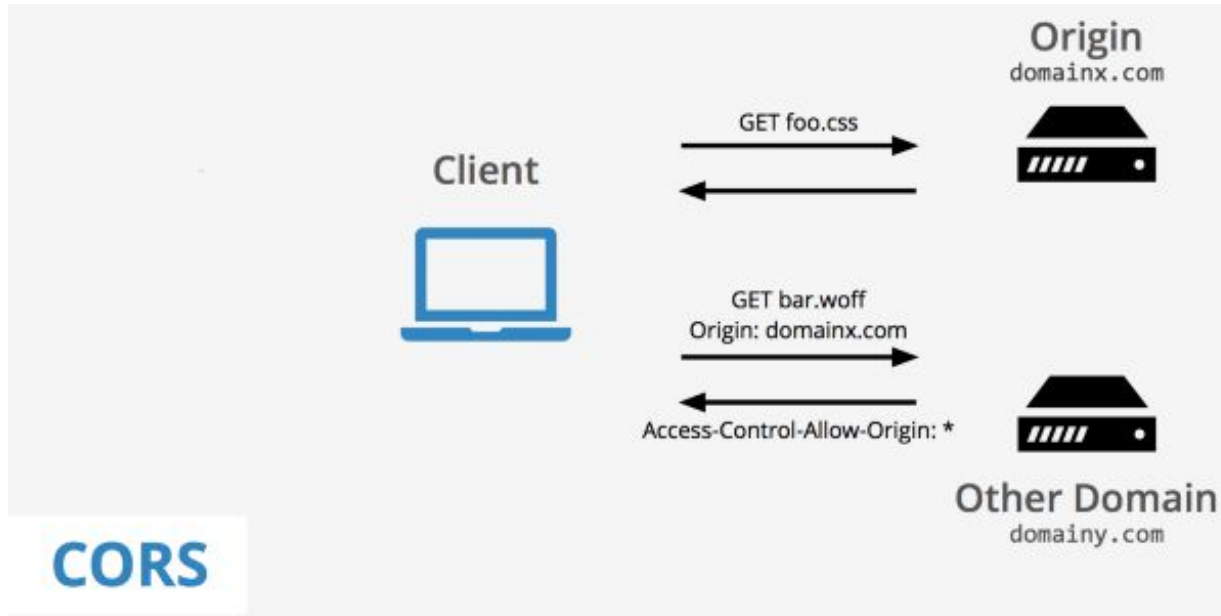
Más que un problema es un bloqueo de seguridad efectuado **por el navegador** (Chrome, Mozilla, etc).



Esto es para que **por default**, el **javascript** alojado en un dominio **misitio.com** sólo pueda ejecutar llamados http hacia **misitio.com**. Esto 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 **otro dominio.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)

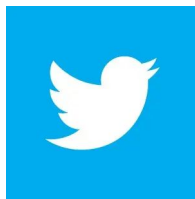


POST

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 de producto - A

1. Crear componente **ItemDetailContainer.js** que al iniciar (usando effects) llame a una **promise** que en 3 segundos le devuelva **un Item** y **lo guarde en un estado** propio, dando feedback visual del estado 'Loading' (como prefieras)



Detalle de producto - B

1. Crear componente ItemDetail.js que debe recibir **por prop** el Ítem obtenido en 6A
2. Conectarlo con el contador del desafío 4 y un nuevo botón
“Comprar”

Formato de entrega (a y b): carpeta comprimida con los archivos del proyecto

Tiempo total: 40 minutos

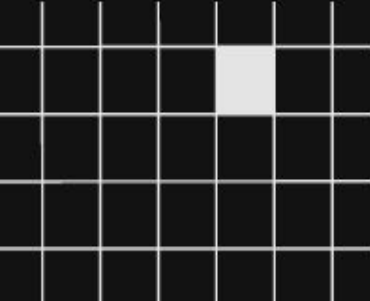
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

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



OPINA Y VALORA ESTA CLASE