¡Les damos la bienvenida!

¿Comenzamos?

CODERHOUSE

Esta clase va a ser

- grabada

CODERHOUSE

Clase 07. REACT JS

Consumiendo API's

CODERHOUSE

Temario

06

Promises, asincronía y MAP

- ✓ Promise
- ✓ MAP

07

Consumiendo API's

- ✓ Paradigmas de intercambio de información
- ✓ REQUESTS VIA HTTP/S
- ✓ REQUESTS EN EL BROWSER

08

Workshop: Hooks, Children y Patrones

- ✓

CODERHOUSE

Objetivos de la clase

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

CODERHOUSE

CLASE N°06

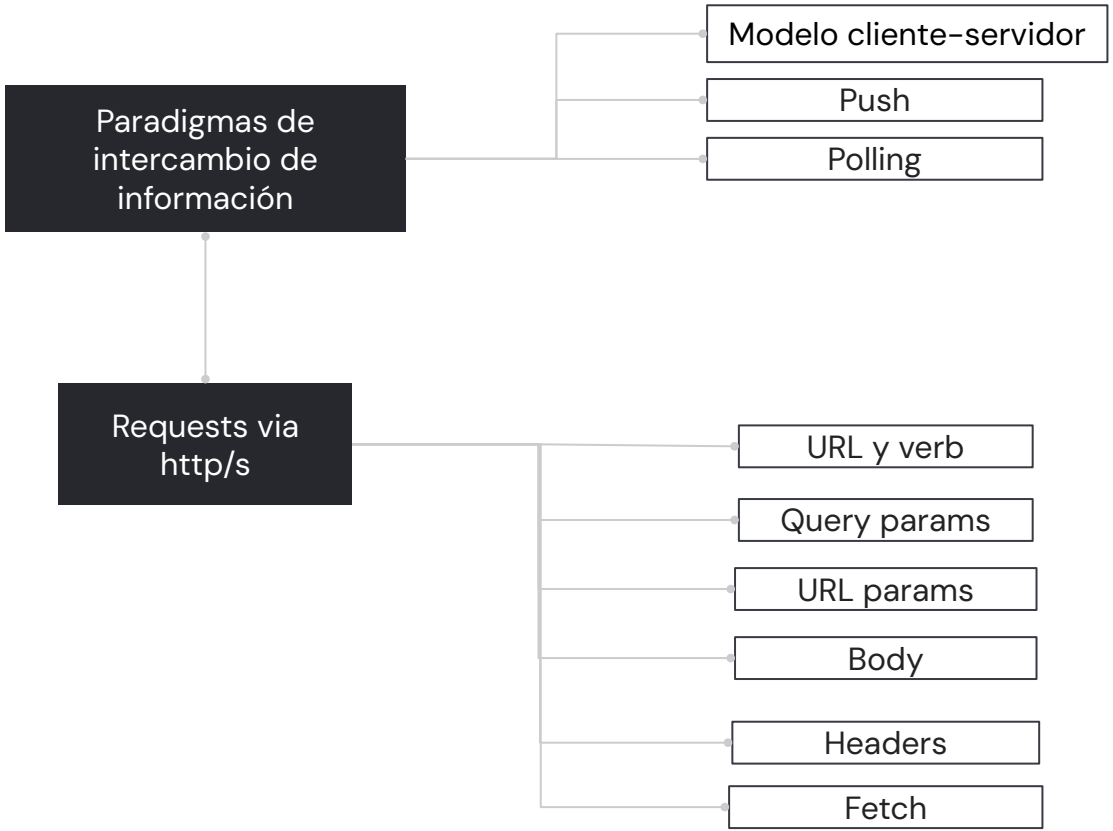
Glosario

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.

CODERHOUSE

MAPA DE CONCEPTOS

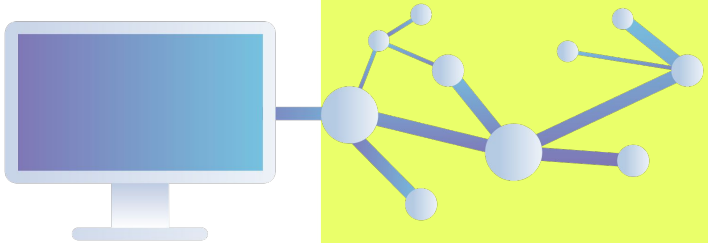


CODERHOUSE

Paradigmas de intercambio de información

CODERHOUSE

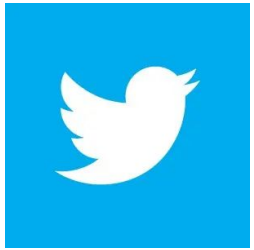
La mayoría de las aplicaciones suelen generar experiencias de usuario gracias a que se pueden conectar a un conjunto de servicios de datos



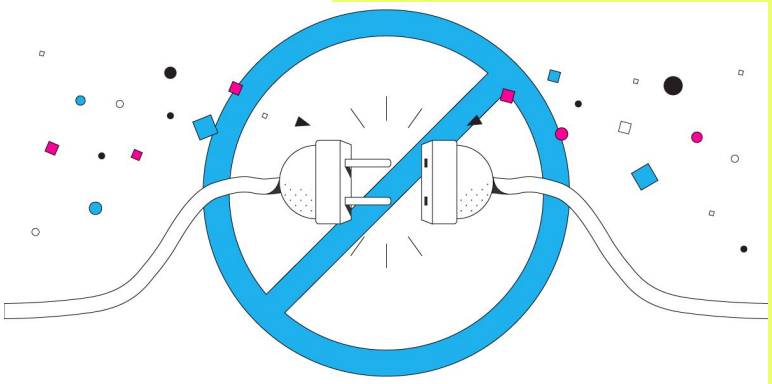
CODERHOUSE

- ✓ 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.

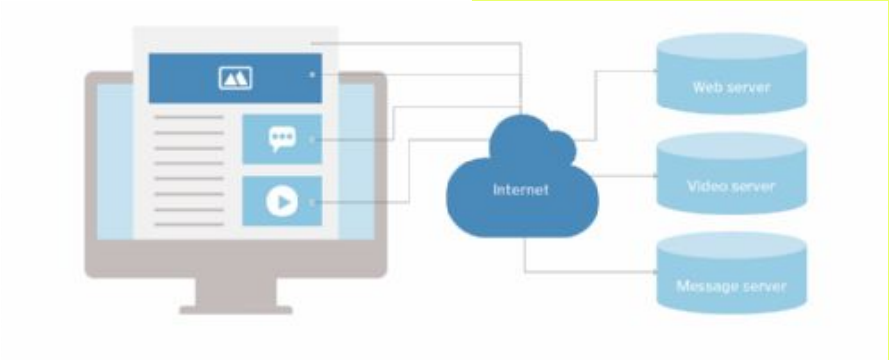


Carecer de una conexión a un servicio de datos es un gran limitante para prácticamente cualquier app que busque vender, o conectar personas.



CODERHOUSE

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



CODERHOUSE

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

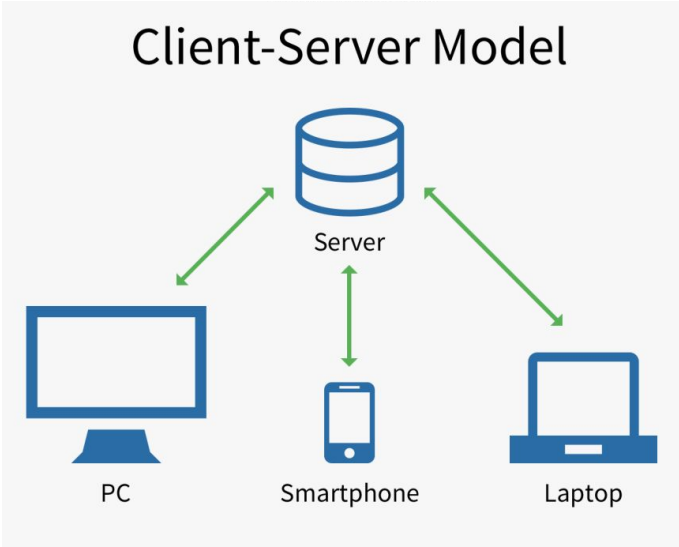
CODERHOUSE

Modelo cliente-servidor

CODERHOUSE

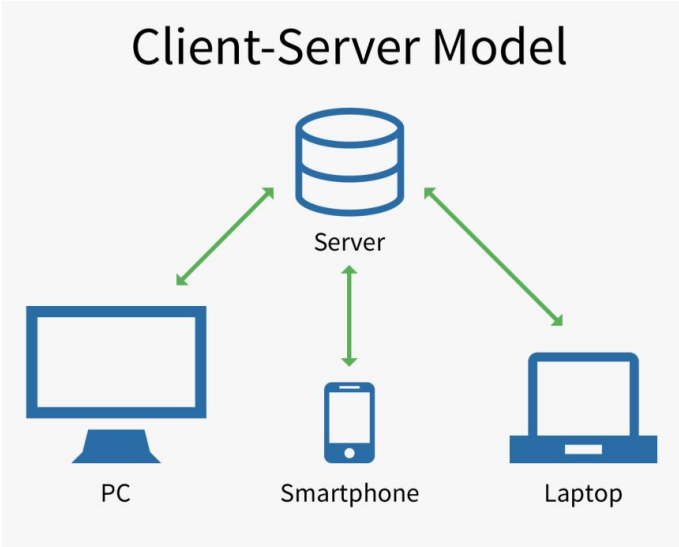
Independientemente de esto, hay algo que parece no cambiar hasta el momento, y es que hay dos protagonistas:

- ✓ Cliente (consumidor)
- ✓ Servidor (proveedor)



CODERHOUSE

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

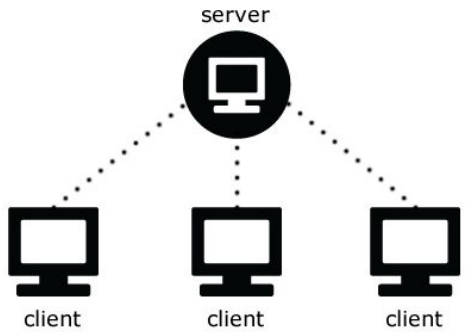


CODERHOUSE

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:

- ✓ El cliente solicita info.
- ✓ El servidor envía la respuesta.
- ✓ Fin de la comunicación.



CODERHOUSE

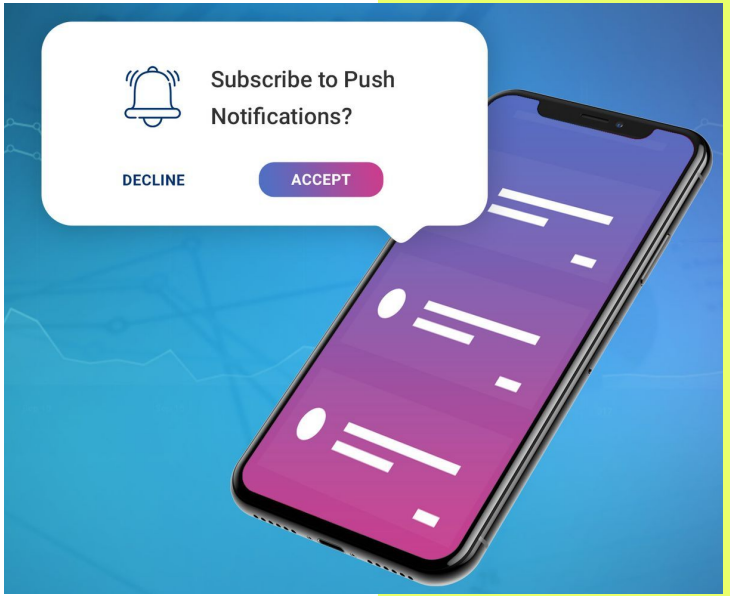
Push

CODERHOUSE

Si invertimos la lógica, se la conoce como PUSH.

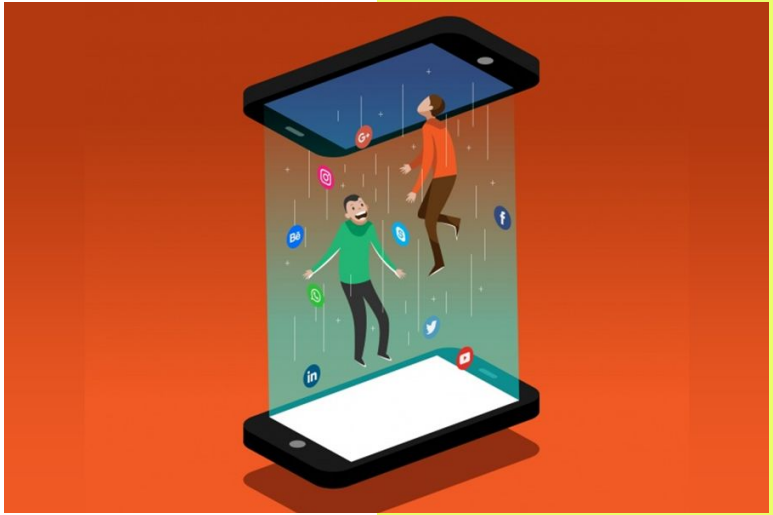
El servidor inicia:

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



CODERHOUSE

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.



CODERHOUSE

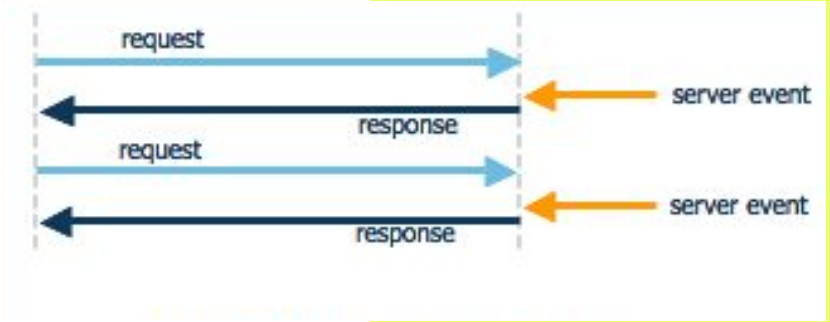
Polling

CODERHOUSE

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

- ✓ ¿Tienes algo nuevo para mí?
- ✓ ¿Tienes algo nuevo para mí?
- ✓ ¿Tienes algo nuevo para mí?

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



CODERHOUSE

REQUESTS VIA HTTP/S

CODERHOUSE

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).

CODERHOUSE

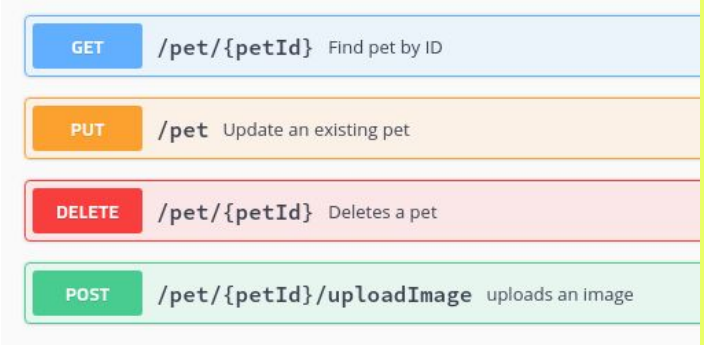
URL Y VERB

CODERHOUSE

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.



CODERHOUSE

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.

CODERHOUSE

Query params

CODERHOUSE

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 tendra key=value & key2=value2
- ✓ Cada parámetro se puede separar por &
- ✓ http://url.com/find?type=order&id=1234

CODERHOUSE

URL params

CODERHOUSE

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	https://myapp.coder/student/1234/courses
Se puede leer como: ✓ *busca en **myapp.coder*** ✓ *utilizando **https...*** ✓ *el recurso **student*** ✓ *con id 1234*	Se puede leer como: ✓ *busca en **myapp.coder*** ✓ *utilizando **https...*** ✓ *el recurso **courses*** ✓ *Únicamente para **student** 1234*

CODERHOUSE

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.

CODERHOUSE

Body

CODERHOUSE

Body

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

CODERHOUSE

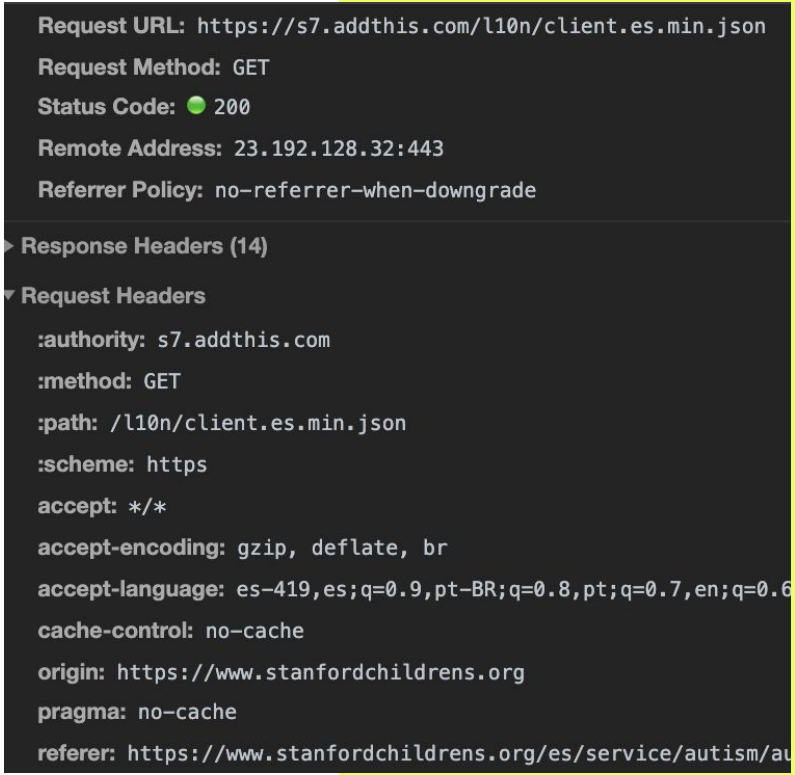
Headers

CODERHOUSE

HEADERS

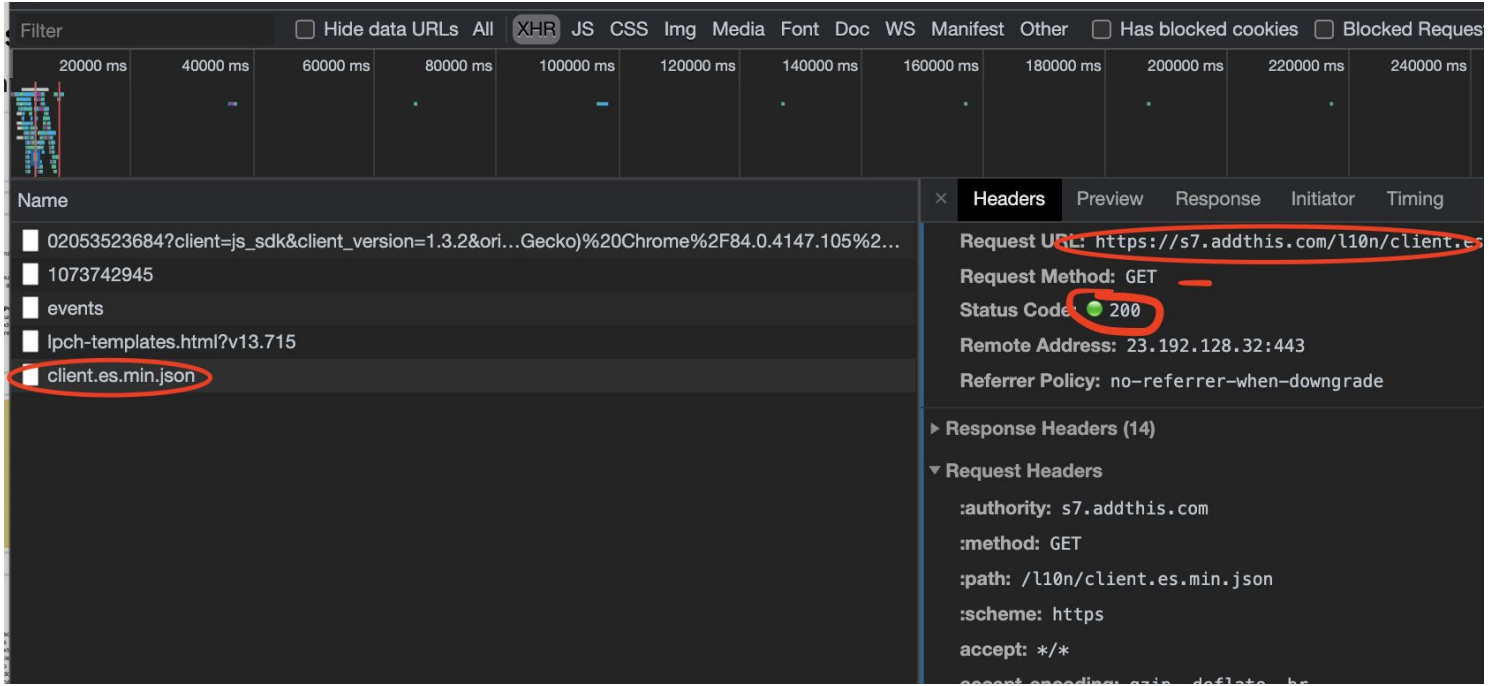
Se usan para:

- ✓ 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.



CODERHOUSE

HEADERS



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

CODERHOUSE

REQUESTS EN EL BROWSER

CODERHOUSE

Fetch

CODERHOUSE

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);
  });
```

CODERHOUSE

Ejemplo en vivo

¡Vamos al código!

CODERHOUSE

FETCH API-CALL

Crea en Stackblitz una app de React que al iniciar utilice Fetch API.

Duración: **20 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

FETCH API-CALL

Descripción de la actividad.

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!)

Break

¡10 minutos y volvemos!

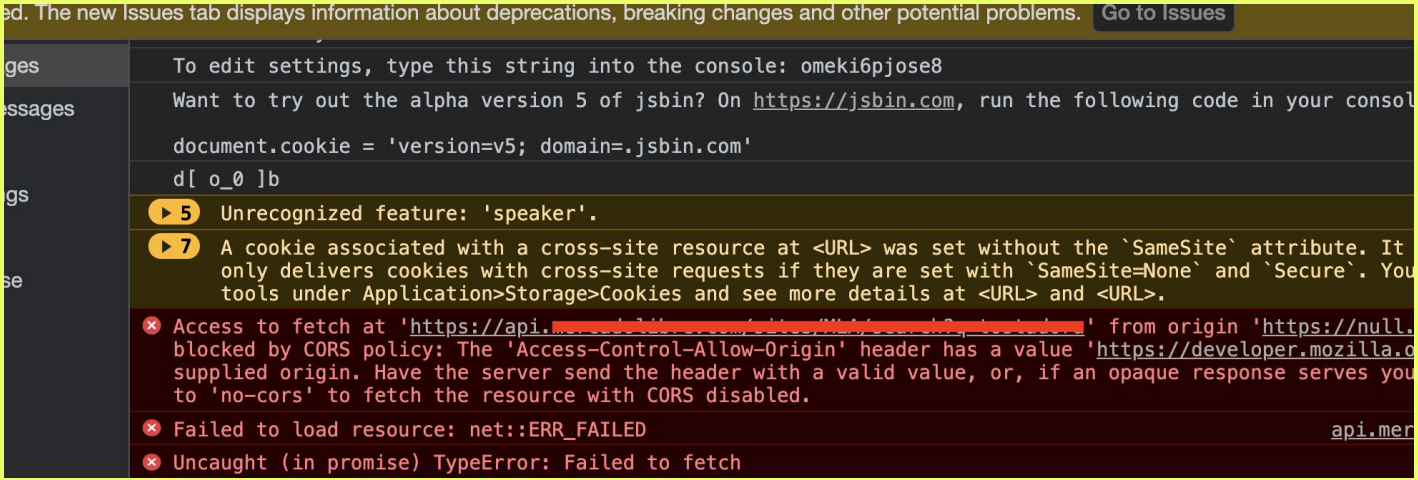
CODERHOUSE

Cors

CODERHOUSE

CORS

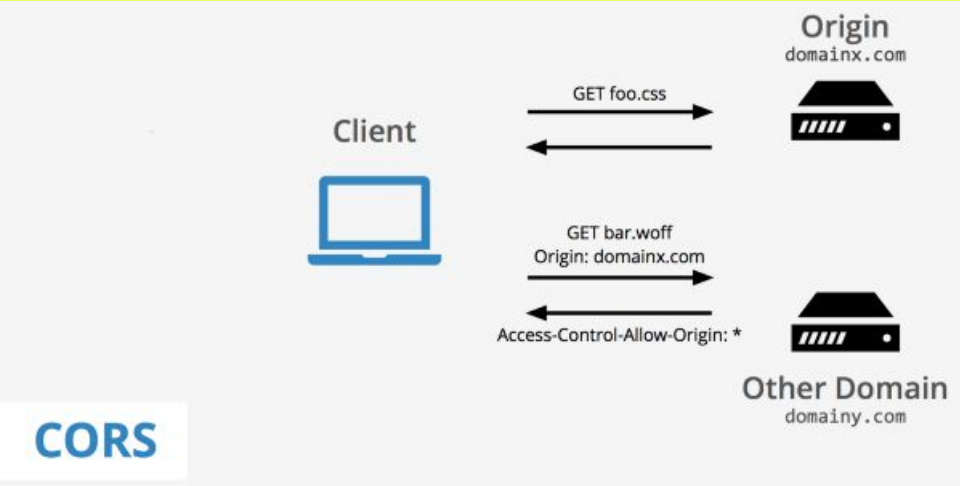
- ✓ Al realizar un request, nos podemos encontrar con este error/problema:



CODERHOUSE

CORS: PREFLIGHT

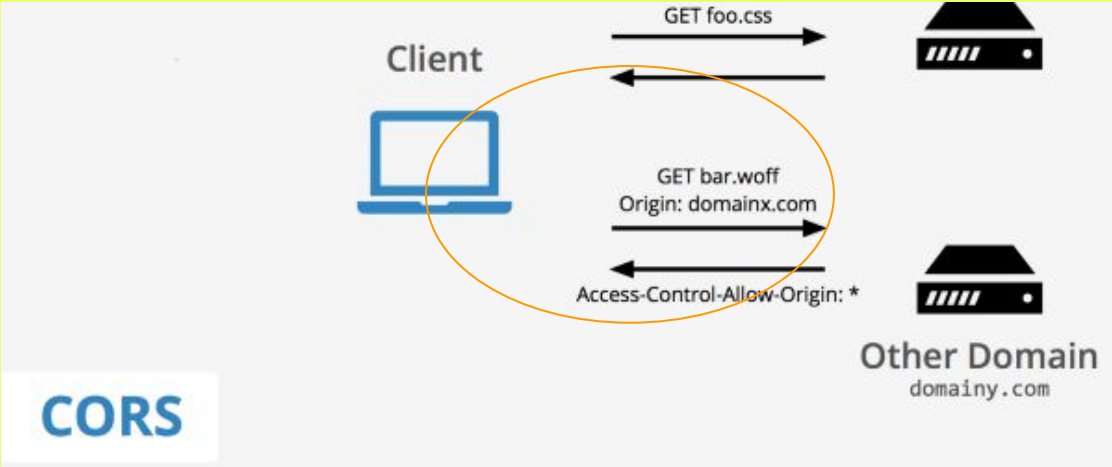
- ✓ Antes de enviar un request entre dominios como en el siguiente ejemplo, el browser envía un request options, llamado pre-flight:



CODERHOUSE

CORS: PREFLIGHT

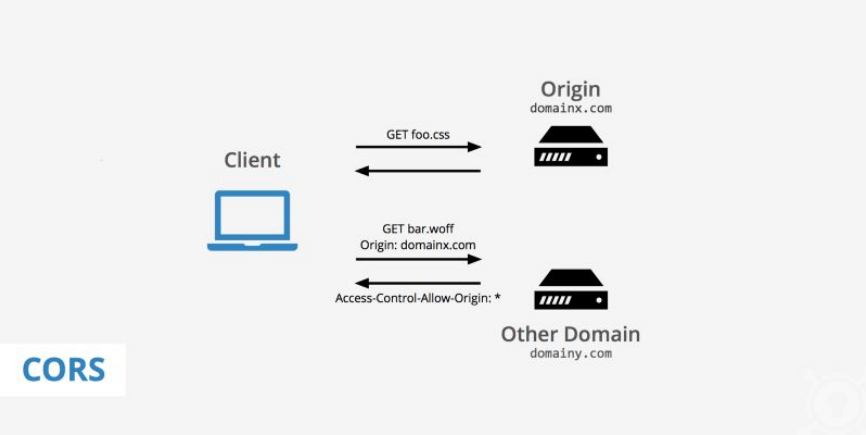
- ✓ En este request, se le pregunta al otro dominio si acepta requests provenientes de un dominio distinto (cross):



CODERHOUSE

CORS

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

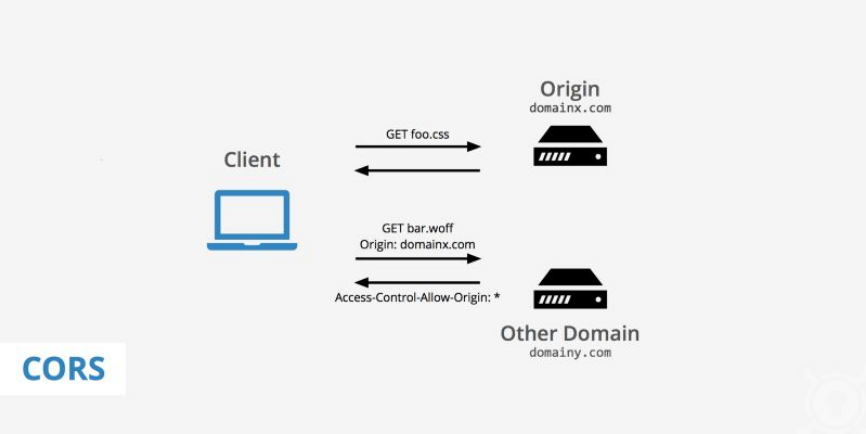


Esto es para que por default, el javascript alojado en un dominio misitio.com sólo pueda ejecutar llamados http hacia misitio.com. Previene algunos problemas de seguridad.

CODERHOUSE

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



CODERHOUSE

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

CODERHOUSE

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.
- ✓ Configurarlo bien nos puede ayudar a resolver algunos inconvenientes durante el desarrollo.

CODERHOUSE

¿Por qué tanta vuelta con el tema?

CODERHOUSE

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

Son el canal transaccional más importante del mundo.

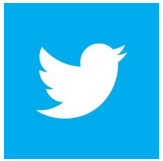
Nos conectan a soluciones que puedan complementar nuestro modelo de negocio y suman adoptabilidad.

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



CODERHOUSE

#Coderalert

Ingresa al manual de prácticas y realiza la tercera actividad “Detalle del producto”. Ten en cuenta que el desarrollo de la misma será importante para la resolución del Proyecto Final.

Actividad Nº 3

Segunda pre-entrega PF

Detalle del producto - Parte I

Descripción de la actividad.

- ✓ Crea tu componente ItemDetailContainer con la misma premisa que ItemListContainer.

Recomendaciones

- ✓ 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 prop

Actividad Nº 3

Segunda pre-entrega PF

Detalle del producto - Parte II

Descripción de la actividad.

- ✓ Crea tu componente ItemDetail.js

Recomendaciones.

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

CODERHOUSE

¿Preguntas?

CODERHOUSE

Resumen
de la clase hoy

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

CODERHOUSE

Opina y valora esta clase

CODERHOUSE

Muchas gracias.

CODERHOUSE

#DemocratizandoLaEducación

CODERHOUSE