

# AJAX - REST

---

**{ JSON }**

ES7 incorpora la interfaz **fetch()**

```
let promise = fetch(url);  
promise.then(response => ...do something... )
```

O la versión corta

```
fetch(url).then(response => ...do something... )
```

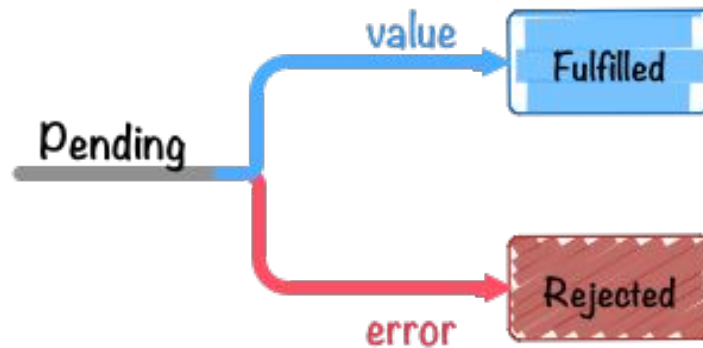
El uso más simple de `fetch()` toma un argumento (la ruta del recurso que se quiera traer) y **el resultado es una promesa** que contiene la respuesta (un objeto Response)

# Terminología

REPASO

Una promesa tiene 4 estados

- Cumplida (*fulfilled*)
- Rechazada (*rejected*)
- Pendiente (*pending*)
- Finalizada (*settled*)



Término **then**

Usando funciones

```
hazAlgo(exitCallback, falloCallback);
```

Usando promesas

```
hazAlgo().then(exitCallback, falloCallback);
```

```
hazAlgo().then(exitCallback).catch(falloCallback);
```

# Cómo funciona fetch



REPASO

Ahora, queremos ver el contenido del archivo

```
fetch('/file.html')
  .then(function(r){
    return r.text()
  })
  .then(function(html) {
    console.log(html); // Contenido del archivo disponible
  })
  .catch(function(e) {
    console.log("Booo");
  })
```

Esperamos a que resuelva la promesa de Fetch pasando una función al método **then()**.

# fetch().then().then()



¿Qué está ocurriendo en cada llamado a la función **then()**?

```
fetch('/file.html')  
  .then(function(r){  
    return r.text()  
  })  
  .then(function(html) {  
    console.log(html);  
  })  
  .catch(function(e) {  
    console.log("Booo");  
  })
```

Respuesta de la solicitud fetch

Procesamiento de la respuesta  
(Nos da otra promesa)

Respuesta procesada

Error de conexión

# Con await/async

REPASO

```
async function load2(event) {
  event.preventDefault();

  let container = document.querySelector("#use-ajax");
  container.innerHTML = "<h1>Loading...</h1>";

  try {
    let response = await fetch(url);
    if (response.ok) {
      let t = await response.text();
      container.innerHTML = t;
    }
    else
      container.innerHTML = "<h1>Error - Failed URL!</h1>";
  }
  catch (response) {
    container.innerHTML = "<h1>Connection error</h1>";
  };
}
```

El valor de retorno de la promesa de Fetch es un **objeto Response** con información del request realizado

Cada respuesta puede tener datos en su cuerpo (HTML, texto, imagenes, JSON, etc)

Podemos especificar el tipo de contenido y cómo debe ser tratado, **todas estas operaciones dan otra promesa**

- `res.text()`
- `res.blob()`      `// Se usa para media: imagenes, audio, video`
- `res.json()`
- otros



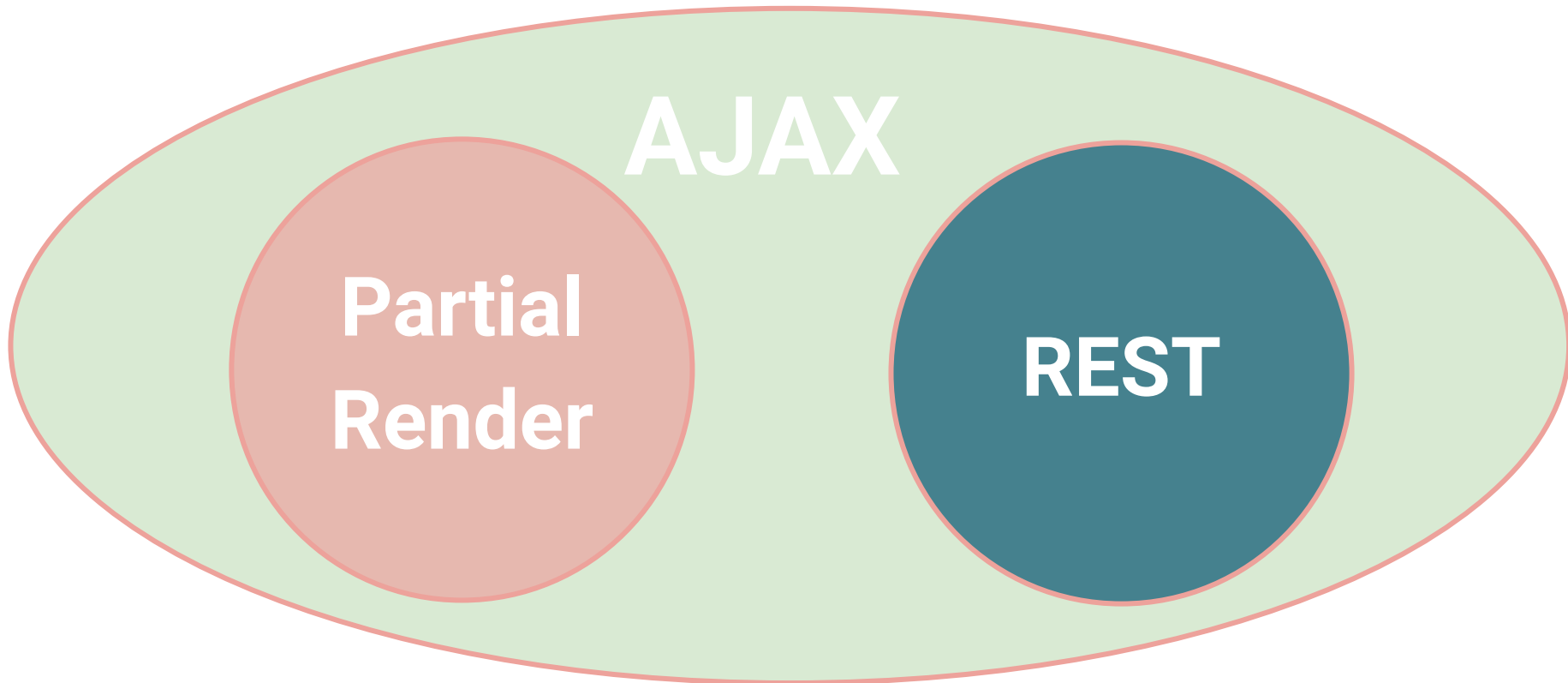
# AJAX REST

Información en formato JSON

# Estilos de AJAX

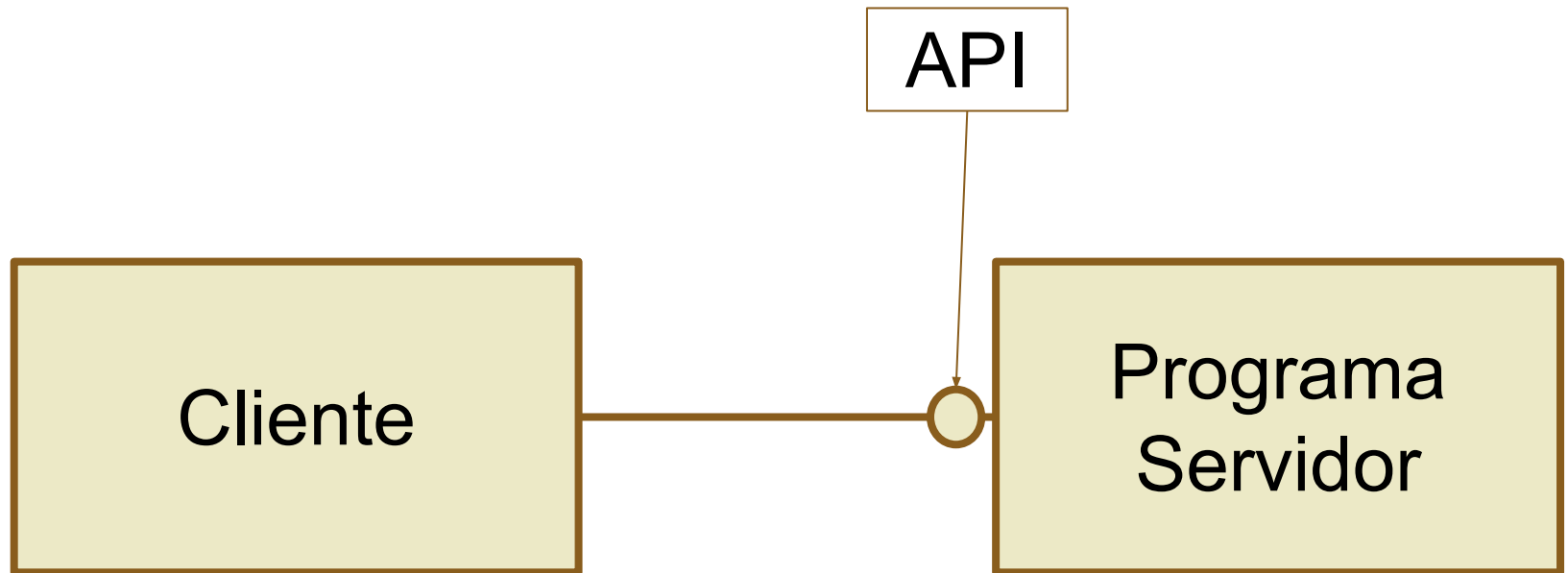
---

- Partial render de páginas
  - Cargar un fragmento de HTML y mostrarlo en un DIV.
- Servicio REST
  - Cargar un objeto JSON y procesarlo del lado del cliente con Javascript



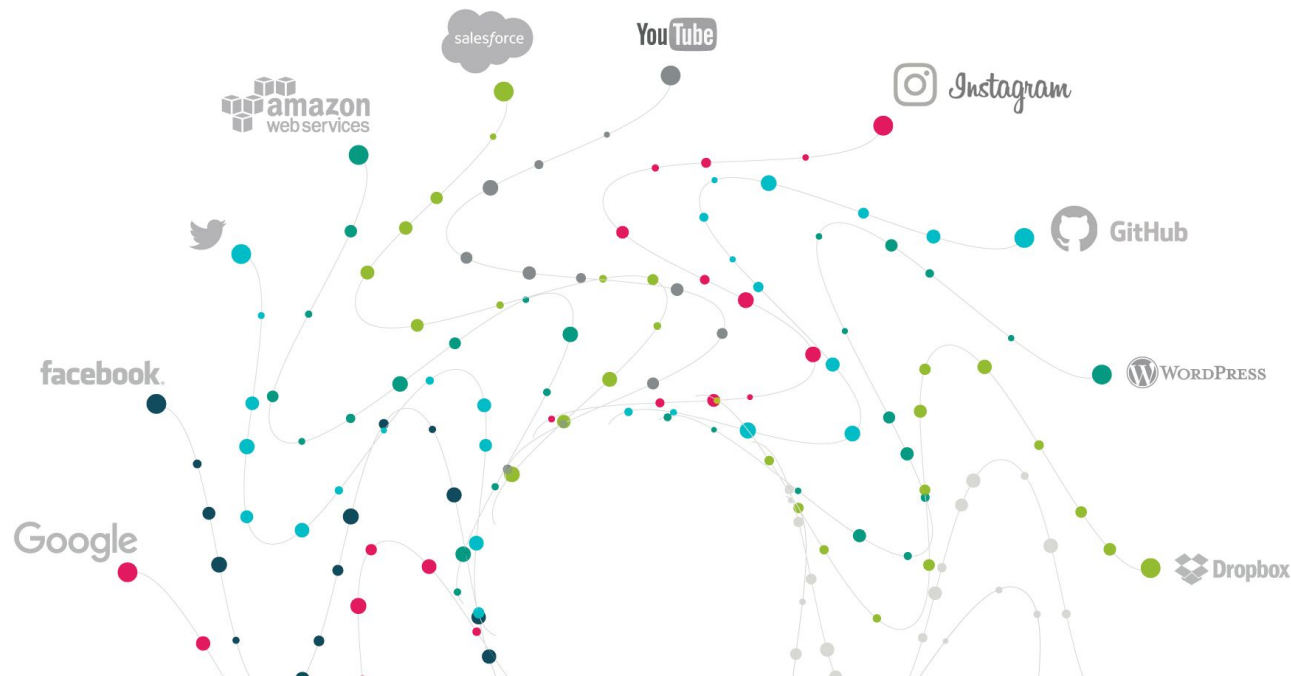
# API

- Una API es una interfaz que nos da una aplicación para comunicarnos con ella



# REST

- REST: REpresentational State Transfer, es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.
- Es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet.
- La mayoría de las APIs REST usan JSON para comunicarse.



# JSON (Repaso)

REPASO

## JavaScript Object Notation

Formato ligero para el intercambio de datos.

Alternativa a XML como representación de objetos.

```
let objeto =  
  {  
    "propiedad": valor,  
    "propiedad2": valor2,  
    "arreglo": [val1, val2]  
  }
```

# REST

---

- Se asocian URLs a recursos.
- Al que se puede acceder o modificar mediante los métodos del protocolo HTTP.
- Se basa en acciones (llamadas verbos) que manipulan los datos.
  - POST: Crear un recurso
  - GET: Obtener uno o muchos recursos
  - PUT: Actualizar uno o muchos recursos
  - DELETE: Borrar un recurso
- Se utilizan los errores del protocolo HTTP.
  - 200 ok, 404 not found, etc.

# API REST - EJEMPLO

---

- **GET** /facturas (en genérico /facturas)
  - Acceder al listado de facturas
- **POST** /facturas (en genérico /facturas)
  - Crear una factura nueva
- **GET** /facturas/123 (en genérico /facturas/:id\_fact)
  - Acceder al detalle de **una** factura
- **PUT** /facturas/123 (en genérico /facturas/:id\_fact)
  - Editar la factura, sustituyendo la **totalidad** de la información anterior por la nueva.
- **DELETE** /facturas/123 (en genérico /facturas/:id\_fact)
  - Eliminar la factura

# Manejo de errores en REST

---

## **Se pueden utilizar los errores del protocolo HTTP:**

- 200 OK Standard response for successful HTTP requests
- 201 Created
- 202 Accepted
- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorised
- 402 Payment Required
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 500 Internal Server Error
- 501 Not Implemented



## Servicio web-unicen

---

- Vamos a usar un servicio que creamos desde la cátedra.
  - URL: `web-unicen.herokuapp.com`
- Este servicio:
  - Guarda información con el siguiente formato:
    - Id: Es autogenerado (no se pasa al crearlo).
    - Thing: **Un objeto JSON.**
  - En la URL decimos nuestro grupo y que vamos a guardar
    - `/catedraweb/notas` para especificar que el grupo *catedraweb* guarda *notas*

## Usar ID de grupos únicos:

- número de grupo,
- inicial/es y apellido/s
- marca del sitio



**/api/groups/1/tpespecial**

**/api/groups/catedraweb/tpespecial**

**/api/groups/grupo01/tpespecial**

**/api/groups/javier/tpespecial**



**KEEP  
CALM  
AND  
LET'S  
CODE**

Post: Envío de datos

¿Cómo subo la información?

- Envio una solicitud
  - Método: POST
  - URL:

`http://web-unicen.herokuapp.com/api/groups/:grupo/:coleccion/`

- La va a grabar en mi grupo y nombre de colección

# Fetch: Opciones

---

Para enviar datos, necesitamos de fetch un segundo parámetro para indicar opciones

```
fetch(url, opciones).then(...)
```

Las opciones permiten definir:


- Verbos HTTP (method)
- Cuerpo del mensaje (body)
- Encabezados (headers)
- Otras opciones

# Opciones - Fetch POST


Usamos las opciones **method**, **headers** y **body**.

```
fetch('http://url...', {  
  'method': 'POST',  
  'headers': {  
    'Content-Type': 'application/json'  
  },  
  'body': '{ "nombre": "Juan" }'  
})  
.then(...
```


Indicamos el verbo  
de la solicitud



Content-Type dice el tipo de  
contenido del body enviado



El cuerpo debe ser una cadena  
(String)



## ¿Cómo guardar la información?

- Método: POST

- URL:

`http://web-unicen.herokuapp.com/api/groups/:GROUP/:COLECCION`

- Data: JSON Object

- Ejemplo

```
{  
  "thing": {"nombre": "Web Unicen!!"}  
}
```



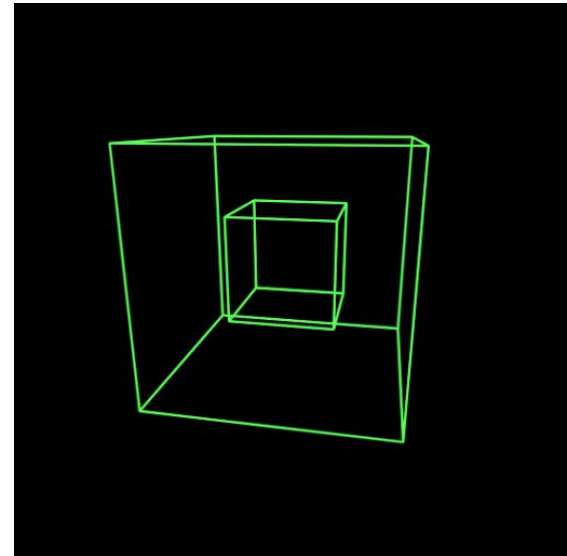
# Guardar datos

El objeto **thing** puede almacenar cualquier información que sea representada mediante JSON.

Ahí guardamos los datos que queremos en nuestro “producto”, “serie”, “noticia” o lo que sea.

Puedo guardar un objeto con subobjetos, un arreglo, o cualquier cosa!

```
let data = {  
  "thing":  
    {  
      "nombre": "Web Unicen!!"  
    }  
};
```

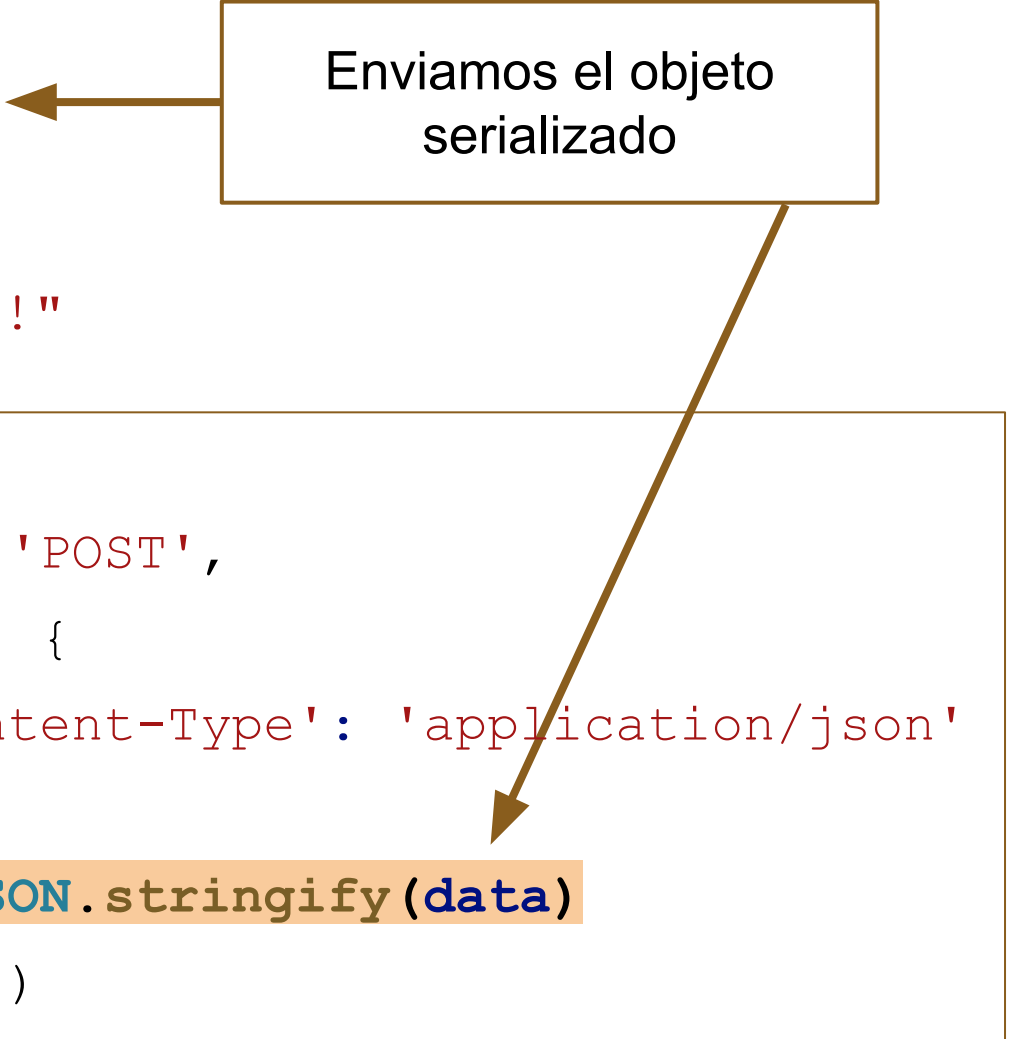


# Enviar JSON

El servicio de la cátedra espera siempre un objeto "thing", lo armamos y convertimos a string

```
let data = {  
  "thing":  
    {  
      "nombre": "Web Unicen!!"  
    }  
};
```

Enviamos el objeto  
serializado



```
graph TD; A[Enviamos el objeto serializado] --> B[body: JSON.stringify(data)];
```

```
fetch(url, {  
  method: 'POST',  
  headers: {  
    'Content-Type': 'application/json'  
  },  
  body: JSON.stringify(data)  
}).then(...)
```

# Ejemplo - Crear Información

```
let data = {
  "thing": {
    "nombre": "Web Unicen!!!"
  }
};

fetch('https://web-unicen.herokuapp.com/api/groups/ejemplos/nombres' , {
  "method": "POST",
  "headers": { "Content-Type": "application/json" },
  "body": JSON.stringify(data)
}).then(function(r) {
  return r.json()
})
.then(function(json) {
  let contenedor = document.querySelector("#result");
  contenedor.innerHTML = JSON.stringify(json);
})
.catch(function(e) {
  console.log(e)
})
```

<https://codepen.io/webUnicen/pen/VdeMed>

# Ejemplo - Crear Información

```
let data = {
  "thing": {
    "nombre": "Web Unicen!!!"
  }
};

fetch('https://web-unicen.herokuapp.com/api/groups/ejemplos/nombres' , {
  "method": "POST",
  "headers": { "Content-Type": "application/json" },
  "body": JSON.stringify(data)
}).then(function(r) {
  return r.json()
})
.then(function(json) {
  let contenedor = document.querySelector("#result");
  contenedor.innerHTML = JSON.stringify(json);
})
.catch(function(e) {
  console.log(e)
})
```

<https://codepen.io/webUnicen/pen/VdeMed>

# Ejemplo - Crear Información

```
let data = {
  "thing": {
    "nombre": "Web Unicen!!!"
  }
};

try{
  let r = await fetch('...',
    {
      "method": "POST",
      "headers": { "Content-Type": "application/json" },
      "body": JSON.stringify(data)
    });
  let json = await r.json();
  let contenedor = document.querySelector("#result");
  contenedor.innerHTML = JSON.stringify(json);
}
catch(e) {
  console.log(e);
}
```

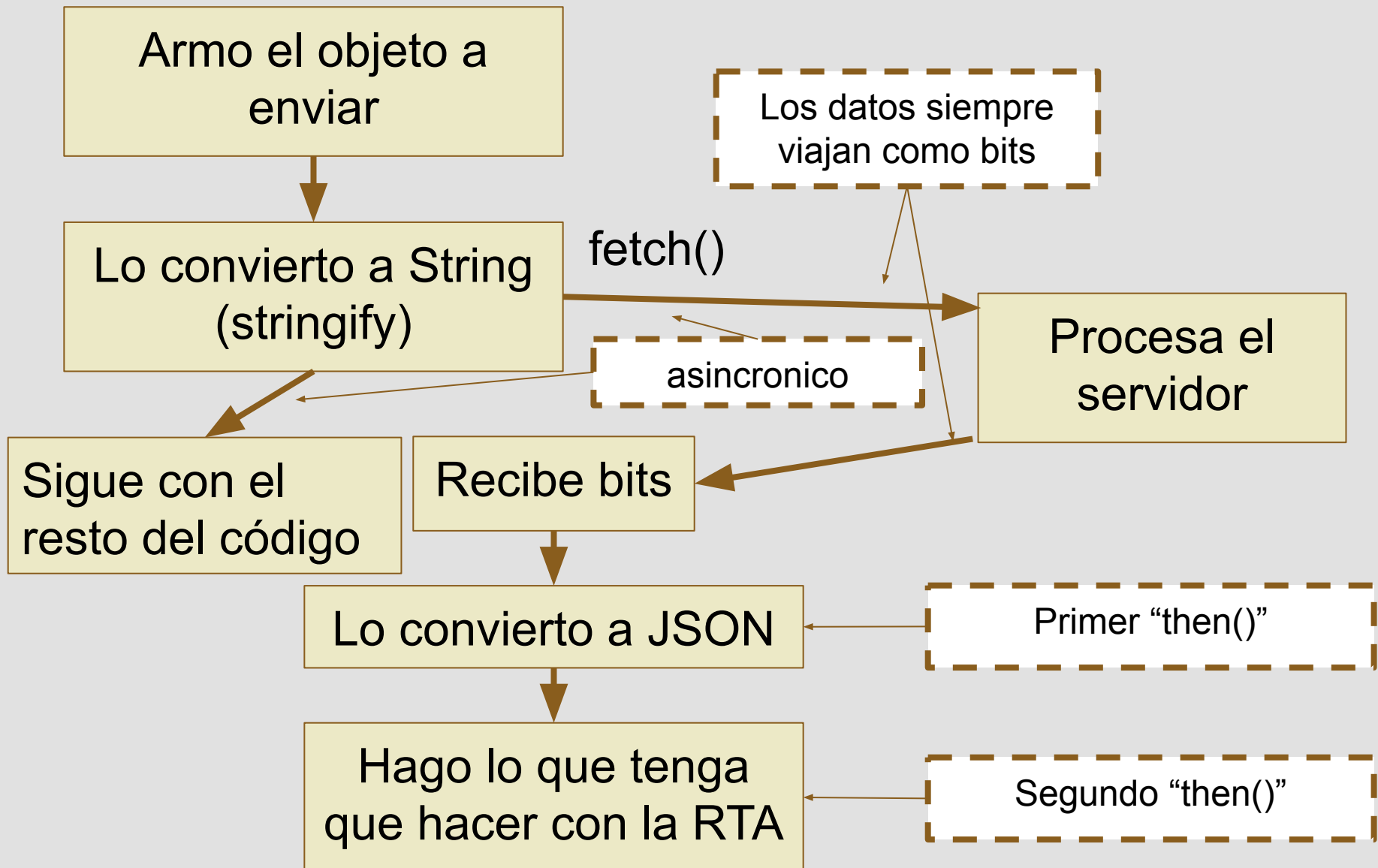
# CORS y Codepen

---

- Los navegadores/servidores tienen políticas de seguridad que en principio evitan que una página acceda a datos de otra
- Las políticas CORS permiten relajar estas restricciones de seguridad
- Por eso nuestros códigos de Codepen tienen opciones adicionales para permitir esta comunicación con otra página

```
fetch(url, {  
    method: 'GET',  
    mode: 'cors',  
}).then(...
```

# Resumen de cómo ejecuta



Get: Obteniendo datos



¿Cómo consulto la información?

- Consulta por ID
  - Método: GET
  - URL:

`http://web-unicen.herokuapp.com/api/groups/:grupo/:coleccion/ + id`

- Consulta por grupo
  - Método: GET
  - URL:

`http://web-unicen.herokuapp.com/api/groups/:grupo/:coleccion`

# Recibir JSON

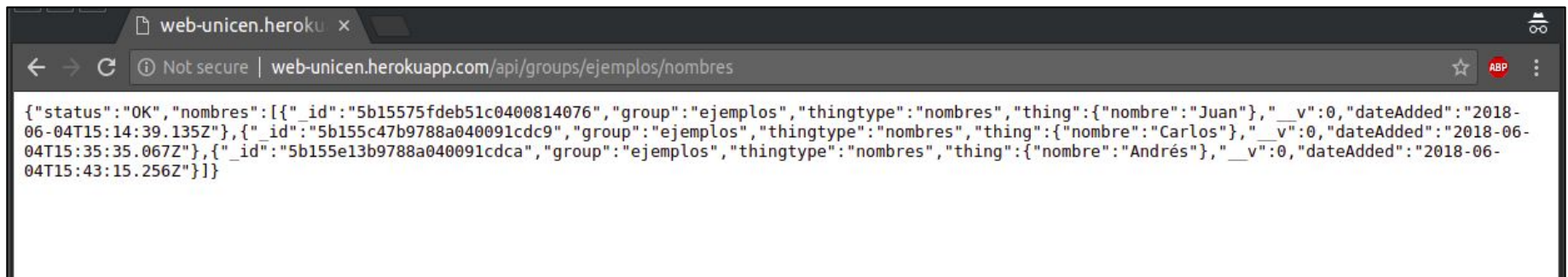
---

```
fetch('http://.../api/groups/' + groupId + '/' + collectionID)
  .then(function(response){
    return response.json()
  })
  .then(function(json) {
    console.log(json);
  })
```

Al ejecutar `res.json()` se parsea (“compila”) a un objeto automáticamente.

# Ver la respuesta JSON

- El navegador por defecto siempre hace GET para bajar las páginas
- Si ponemos la URL en el navegador vemos directamente el JSON (aunque solo nos sirve para GET, no para otros métodos de HTTP)



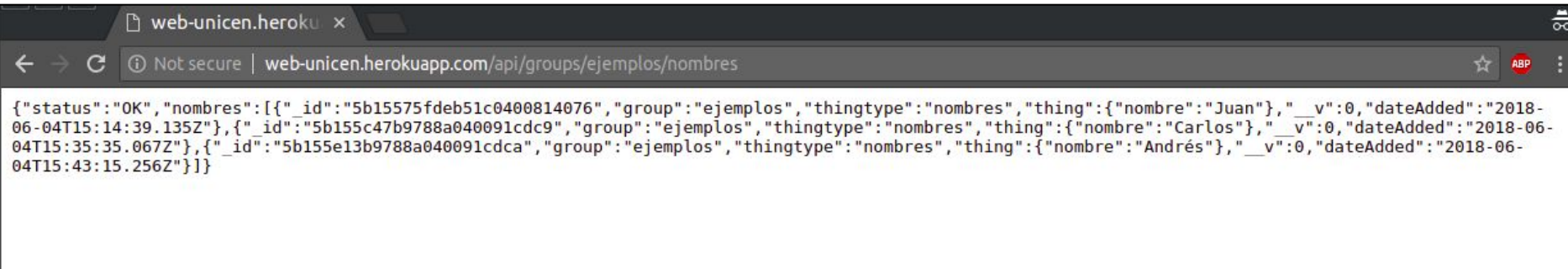
A screenshot of a web browser window. The address bar shows the URL `web-unicen.herokuapp.com/api/groups/ejemplos/nombres`. The page content displays a JSON response from an API. The JSON is formatted with syntax highlighting, showing a status of "OK" and an array of three objects, each representing a name and its creation date.

```
{
  "status": "OK",
  "nombres": [
    {
      "_id": "5b15575fdeb51c0400814076",
      "group": "ejemplos",
      "thingtype": "nombres",
      "thing": {
        "nombre": "Juan"
      },
      "__v": 0,
      "dateAdded": "2018-06-04T15:14:39.135Z"
    },
    {
      "_id": "5b155c47b9788a040091cdc9",
      "group": "ejemplos",
      "thingtype": "nombres",
      "thing": {
        "nombre": "Carlos"
      },
      "__v": 0,
      "dateAdded": "2018-06-04T15:35:35.067Z"
    },
    {
      "_id": "5b155e13b9788a040091cdca",
      "group": "ejemplos",
      "thingtype": "nombres",
      "thing": {
        "nombre": "Andrés"
      },
      "__v": 0,
      "dateAdded": "2018-06-04T15:43:15.256Z"
    }
  ]
}
```

**¿Cómo usamos la respuesta?**  
**¿Qué sabemos hacer y qué**  
**necesitamos?**

# JSON en el navegador

Vemos JSON formateado en el navegador?



A screenshot of a web browser window. The address bar shows the URL `web-unicen.herokuapp.com/api/groups/ejemplos/nombres`. The browser console displays the raw JSON response from the API, which is a single line of text containing the JSON structure.

```
{ "status": "OK", "nombres": [ { "_id": "5b15575fdeb51c0400814076", "group": "ejemplos", "thingtype": "nombres", "thing": { "nombre": "Juan", "__v": 0, "dateAdded": "2018-06-04T15:14:39.135Z" }, { "_id": "5b155c47b9788a040091cdc9", "group": "ejemplos", "thingtype": "nombres", "thing": { "nombre": "Carlos", "__v": 0, "dateAdded": "2018-06-04T15:35:35.067Z" }, { "_id": "5b155e13b9788a040091cdca", "group": "ejemplos", "thingtype": "nombres", "thing": { "nombre": "Andr s", "__v": 0, "dateAdded": "2018-06-04T15:43:15.256Z" } ] }
```

Extensi n para Chrome:  
[JSON Formatter](#)



A screenshot of a web browser window showing the same URL as the previous image. The JSON data is now displayed in a formatted, collapsible tree view. The browser's developer tools are open, and the JSON is color-coded and indented for readability.

```
{
  "status": "OK",
  "nombres": [
    {
      "_id": "5b15575fdeb51c0400814076",
      "group": "ejemplos",
      "thingtype": "nombres",
      "thing": {
        "nombre": "Juan"
      },
      "__v": 0,
      "dateAdded": "2018-06-04T15:14:39.135Z"
    },
    {
      "_id": "5b155c47b9788a040091cdc9",
      "group": "ejemplos",
      "thingtype": "nombres",
      "thing": {
        "nombre": "Carlos"
      },
      "__v": 0,
      "dateAdded": "2018-06-04T15:35:35.067Z"
    },
    {
      "_id": "5b155e13b9788a040091cdca",
      "group": "ejemplos",
      "thingtype": "nombres",
      "thing": {
        "nombre": "Andr s"
      },
      "__v": 0,
      "dateAdded": "2018-06-04T15:43:15.256Z"
    }
  ]
}
```

# Ejemplo de respuesta

Analizamos la estructura de la respuesta para poder leerla

**GET** /api/groups/ejemplos/nombres

```
{
  "status": "OK",
  "nombres": [
    {
      "_id": "5b14b90fad4f7604001f1d53",
      "group": "ejemplos",
      "thingType": "nombres"
      "thing": {
        "nombre": "Web Unicen!!!"
      },
      "__v": 0,
      "dateAdded": "2018-06-04T03:59:11.684Z"
    }
  ]
}
```

Array de todos los registros



# Ejemplo - Consulta por grupo

```
fetch('https://web-unicen.herokuapp.com/api/groups/ejemplos/nombres' )
  .then(function(r) {
    return r.json()
  })
  .then(function(json) {
    let contenedor = document.querySelector("#result");
    contenedor.innerHTML = ''
    for (let data of json.nombres) {
      contenedor.innerHTML += data.thing.nombre + "<br />";
    }
  })
  .catch(function(e) {
    console.log(e)
  })
```

← Llamada asincrónica mediante GET

Recibir info y editar el DOM

← Loguear Error

<https://codepen.io/webUnicen/pen/rKxzZx>

# Solución

---

Solución ¿completa?

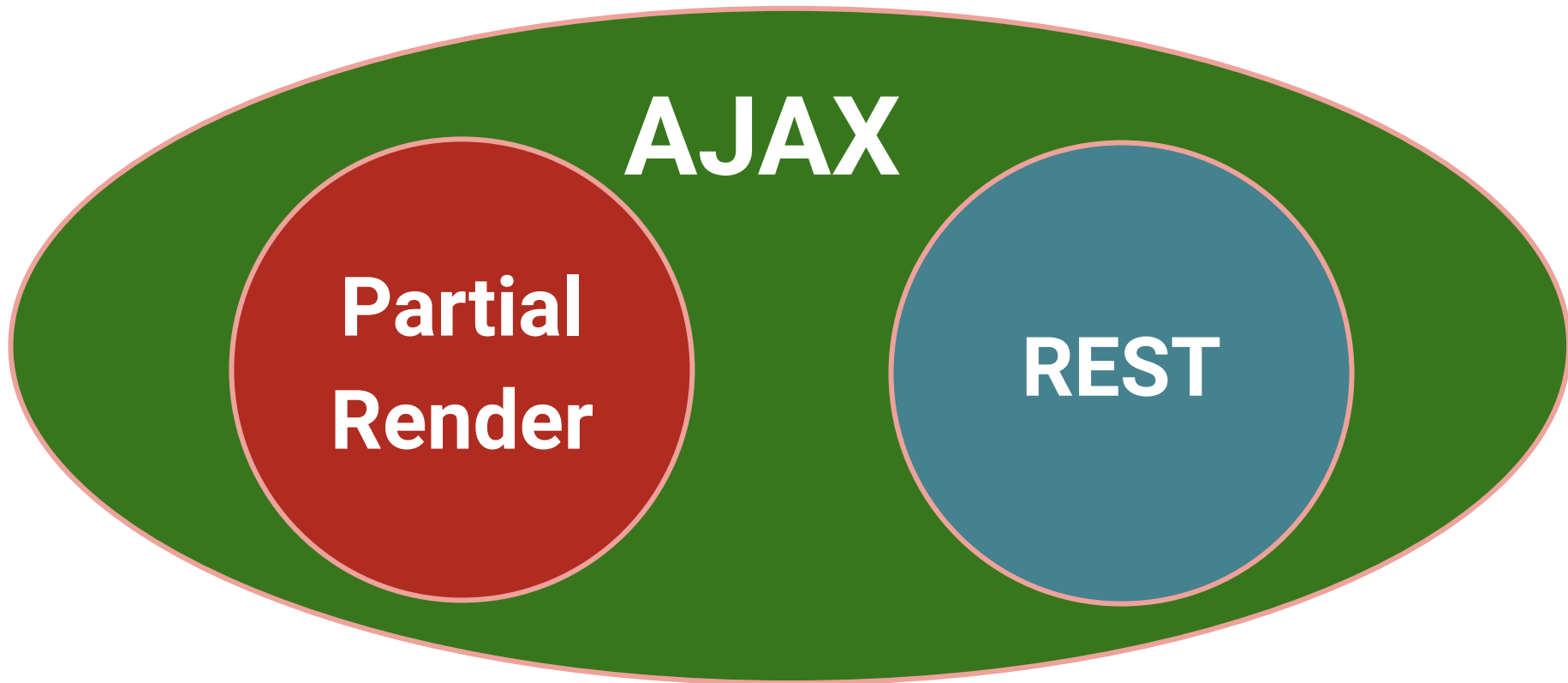
- <https://codepen.io/webUnicen/pen/Nzxavq>



# Estilos de AJAX

---

- Partial render de páginas
  - Cargar un fragmento de HTML y mostrarlo en un DIV.
- Servicio REST
  - Cargar un objeto JSON y procesarlo del lado del cliente con Javascript



# Comparativa

## Partial Render

```
let urlHTML = ...;  
  
let r = await fetch(urlHTML);  
  
let html = await r.text();  
  
let div = document.querySe...  
  
contenedor.innerHTML = html;
```

## REST

```
let url = ".../api/...";  
  
let r = await fetch(url);  
  
let json = await r.json();  
  
let div = document.querySe...  
  
contenedor.innerHTML = ''  
  
for (let data of json.nombres) {  
  
    contenedor.innerHTML +=  
  
    "<p>"...  
  
}
```

# Comparativa

## Accede a un HTML

```
let urlHTML = ...;  
  
let r = await fetch(urlHTML);  
  
let html = await r.text();  
  
let div = document.querySe...  
  
contenedor.innerHTML = html;
```

## Accede a una API

```
let url = ".../api/...";  
  
let r = await fetch(url);  
  
let json = await r.json();  
  
let div = document.querySe...  
  
contenedor.innerHTML = ''  
  
for (let data of json.nombres) {  
  
    contenedor.innerHTML +=  
  
    "<p>" ...  
  
}
```

# Comparativa

## Partial Render

Lee un texto (que sabe que es un HTML)

```
let html = await r.text();  
  
let div = document.querySelector(...)  
  
contenedor.innerHTML = html;
```

## REST

Lee un JSON

```
let json = await r.json();  
  
let div = document.querySelector(...)  
  
contenedor.innerHTML = ''  
  
for (let data of json.nombres) {  
    contenedor.innerHTML +=  
        "<p>" ...  
}
```

# Comparativa

## Partial Render

```
let urlHTML = ...;
```

Inserta HTML en el DOM

```
let div = document.querySelector...  
  
contenedor.innerHTML = html;
```

## REST

```
let url = ".../api/...";
```

Arma un HTML con el JSON  
y lo inserta en el DOM

```
let div = document.querySelector...  
  
contenedor.innerHTML = ''  
  
for (let data of json.nombres) {  
  
    contenedor.innerHTML +=  
  
    "<p>"...  
  
}
```

Put: Modificación de datos

---

**¿Qué estrategia tendríamos que adoptar para actualizar un único valor de un thing?**

Ej: cambiar solo el nombre del señor  
Perez

```
"thing": {  
  "nombre": "Juan",  
  "apellido": "Pérez"  
},
```

## Respuesta

---

Este servicio sobrescribe todo el objeto “thing” entero, debemos enviar un objeto thing con el apellido también sin cambiar su valor.



# Fetch - Put

---

TBC

<https://codepen.io/webUnicen/pen/zarEeW>

Delete: Borrado de datos

---

# Fetch - Delete

---

TBC

# Reflexionemos

---

**TPE: ¿Cómo harían para que esos resultados se muestren en una tabla?**

**[TBC en el TPE]**

# Repaso AJAX: Ensalada de tecnologías

---

Una mezcla donde cada tecnología aporta algo:

- Presentación en estándares HTML y CSS.
- Display e interacciones dinámicas via DOM.
- Intercambio de datos mediante XML (o JSON)
- Lectura de datos asincrónica mediante fetch
- Y JavaScript para unir todo.

# Consecuencias de asincrónico

---

Creo handlers para objetos que aún no existen.  
etc...

EXTRAS



# Ejemplo de carga asincrónica de una imagen



Podemos descargar una imagen de forma asincrónica y mostrarla en un tag `<img>`.

```
let miImagen = document.querySelector('.mi-imagen');  
fetch('https://picsum.photos/200/300.jpg')  
  .then(res => res.blob())  
  .then(res => {  
    var objectURL = URL.createObjectURL(res);  
    miImagen.src = objectURL;  
  });
```



se ejecuta el método **blob()** para que procese el tipo de archivo que esperamos (imagen).

Codepen: <https://codepen.io/webUnicen/pen/MGoYEz>

# Referencias

---

- <http://api.jquery.com/jquery.ajax>
- <https://eamodeorubio.wordpress.com/category/webservices/rest/>
- <https://developer.mozilla.org/es/docs/AJAX>
- <http://www.restapitutorial.com/lessons/whatisrest.html>
- [https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Usar\\_promesas](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Usar_promesas)
- <https://developers.google.com/web/fundamentals/primers/promises?hl=es>
- “BulletProof AJAX” Jeremy Keith

**AHORA LES TOCA PRACTICAR :D**

