



Curso de Front-End

Clase 16



Agenda de la clase



Agenda

- Repaso.
- Ajax.
- Ejercicios.



Repaso



¿Qué es Vue.js?

- Es un **framework JavaScript** para **Front-End**.
- Fue escrito por una sola persona: [Evan You](#) (aunque actualmente existe una comunidad de colaboradores).
- No es tan popular como otros frameworks como Angular y React, pero es más **simple** y por lo tanto más **fácil de aprender**, sobre todo en su **versión 2**. De todas maneras, Vue.js está [creciendo muy rápidamente](#).
- Se dice que Vue.js es “**progresivo**” dado que permite empezar a trabajar con él de una forma muy simple e ir agregándole funcionalidades de a poco, a medida que se necesite.
- Se dice que sigue el paradigma de programación [declarativo](#), en lugar del imperativo (que veníamos usando hasta ahora para manipular el DOM).



Instalación de Vue.js

La instalación de Vue.js es muy sencilla. Similar a como se instaló Bootstrap, se puede copiar y pegar el link del CDN.

```
<body>
  ...
  ...
  ...
  <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
</body>
```

Nota: En caso de necesitarlo, se podría instalar Vue.js y Bootstrap en la misma página.



Utilización de Vue.js

Vue.js permite “atar” la vista (HTML) a los datos (JS). Es lo que se conoce como **Two-way-Binding**.

HTML:

```
<div id="app">
  <p>{{ mensaje }}</p>
  <input type="text" v-model="mensaje">
</div>
```

JS:

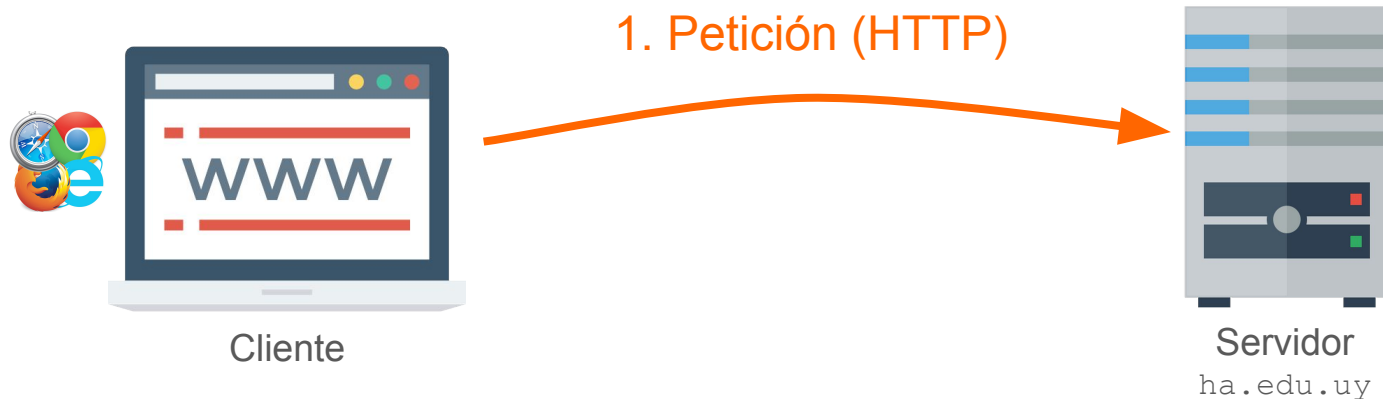
```
var miVueApp = new Vue({
  el: "#app",
  data: {
    mensaje: "Hola"
  }
});
```

En este ejemplo, se creó un elemento `<input>` y se le agregó el atributo `v-model` con el valor “mensaje” (que coincide con el nombre de atributo `mensaje` dentro de `data`).

De este modo, lo que el usuario escriba dentro del `<input>`, automáticamente se guardará dentro de `mensaje`, sobrescribiendo el valor inicial “Hola”. Y no sólo eso, también se actualizará el contenido del `<p>`. **Esto es muy mágico** 🤖🤖🤖🤖🤖🤖.



Arq. Cliente-Servidor (1/2)

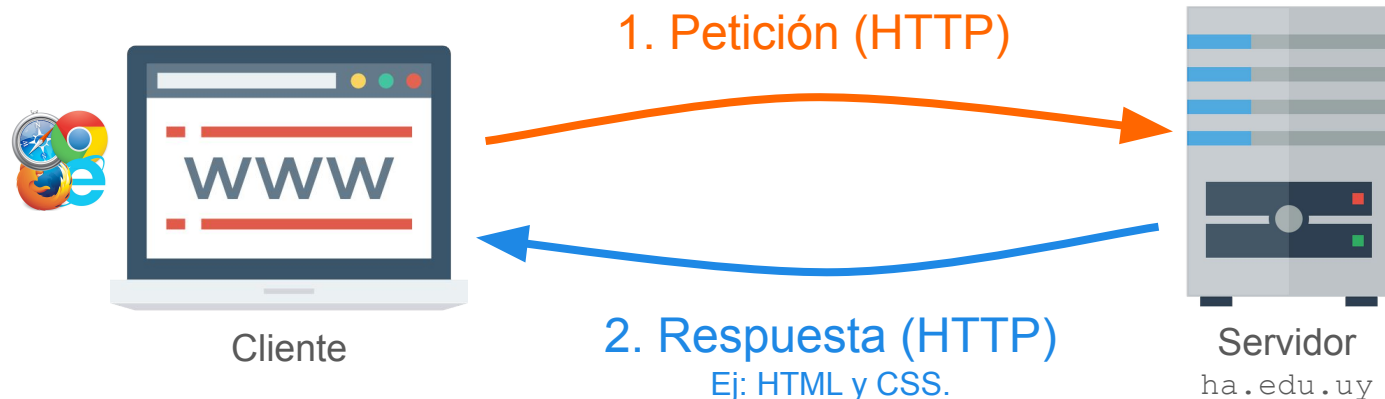


Cuando un usuario escribe una URL en su navegador y presiona `Enter`, lo que hace es realizarle una **petición** (HTTP) a un servidor web.

Básicamente el usuario le dice al servidor: *"Dame lo que haya en esta URL: `https://ha.edu.uy`".*



Arq. Cliente-Servidor (2/2)



Al recibir la petición, una aplicación en el servidor la analiza, la procesa y responde al cliente con una **respuesta** que suele ser en formato HTML (pero puede ser CSS, JavaScript, JSON, una imagen o texto plano). Los posibles formatos se pueden consultar [aquí](#).

El servidor jamás retorna código PHP, Java, Ruby, Python, C# u otro código de Back-End. Recordar que el navegador sólo "entiende" cosas como HTML, CSS, JavaScript e imágenes.



Clientes, Navegadores y Servidores

Los **clientes** son desktops, móviles, *tablets*, etc., que acceden a Internet a través de navegadores.

Los **navegadores** son programas que despliegan contenido (principalmente HTML) de una forma amigable para el usuario.

Existe una gran variedad de navegadores y cada uno tiene sus particularidades.

Dos navegadores diferentes pueden mostrar la misma página web de manera distinta.

Los **servidores** son computadoras que contienen la información (archivos HTML, archivos CSS, imágenes, etc) de un sitio web.

Suelen ser equipos poderosos que están prendidos y conectados a Internet las 24 horas.



AJAX



¿Qué es AJAX?

Se dice "Asíncrono" porque no se sabe cuándo se obtendrá una respuesta y no se frena al programa mientras se espera.

- AJAX = Asynchronous JavaScript and XML.
- Es una **técnica** que permite **interactuar** (enviar y recibir información) **con un servidor sin necesidad de recargar la página**.
- Su surgimiento a mediados de la década del 2000 significó un cambio enorme para JavaScript y el desarrollo de aplicaciones web. Fue una revolución.
- Permitió la creación de aplicaciones como Gmail, Google Maps, Facebook, Twitter, etc. ⇨ **Single Page Applications** (SPA) que se parecen a aplicaciones de escritorio.



¿Qué se puede recibir del servidor vía AJAX?

Cualquier cosa, pero en general:

- Texto plano (texto sin un formato específico).
- HTML.
- **JSON** (que sustituyó al XML; igual se le sigue llamando AJAX).



JSON



¿Qué es JSON?

Como puede ser CSV o XML.

- JSON = JavaScript Object Notation.
- Es un **formato de texto** muy simple para **intercambiar información**.
- Es ligero y fácil de leer por humanos.
- Deriva de JavaScript, pero es independiente del lenguaje de programación.
- La información se guarda en pares de clave-valor.

```
{  
  "nombre"      : "María",  
  "apellido"    : "Rodríguez",  
  "edad"        : 36,  
  "nacionalidad" : ["Uruguay", "España"]  
}
```

A diferencia de un objeto de JavaScript, las propiedades deben ir en comillas (dobles).

¿Qué "cosas" existen que pueden ir dentro de un objeto JavaScript pero no pueden ir dentro de un JSON?



Ejemplos de JSON

Prueben ingresar a las siguientes URLs:

- Open Movie Database: <https://private.omdbapi.com/?apikey=bef9c583&t=cinema+paradiso>
- Star Wars: <https://swapi.dev/api/people/1/?format=json>
- Google Books: <https://www.googleapis.com/books/v1/volumes?q=0596554877>


Estos son servicios que **en lugar de retornar un código HTML retornan código JSON**.

Si quieren hacer pruebas con datos ficticios en formato JSON, entren aquí:

<http://jsonplaceholder.typicode.com>. Sirve para probar y prototipar.



Sugerencia: Instalar esta extensión de Chrome.



JSON Formatter

offered by callumlocke.co.uk

★★★★★ (1168) [Developer Tools](#) 596,760 users

ADDED TO CHROME

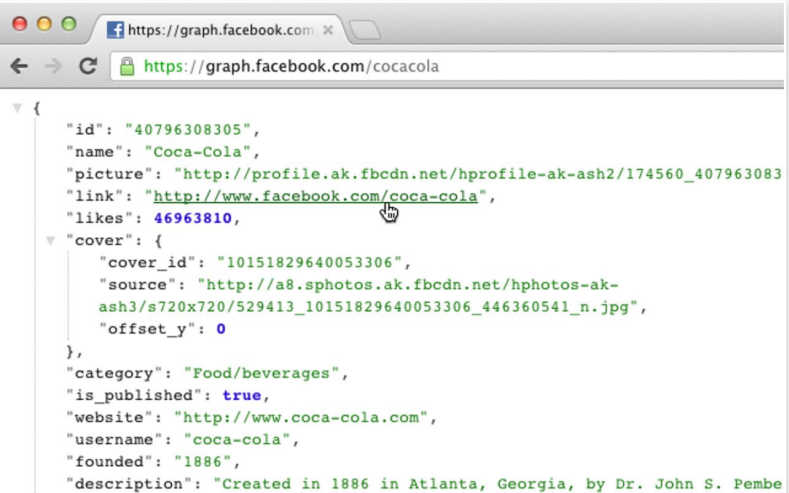
OVERVIEW

REVIEWS

SUPPORT

RELATED

G+



```
{
  "id": "40796308305",
  "name": "Coca-Cola",
  "picture": "http://profile.ak.fbcdn.net/hprofile-ak-ash2/174560_407963083",
  "link": "http://www.facebook.com/coca-cola",
  "likes": 46963810,
  "cover": {
    "cover_id": "10151829640053306",
    "source": "http://a8.sphotos.ak.fbcdn.net/hphotos-ak-ash3/s720x720/529413_10151829640053306_446360541_n.jpg",
    "offset_y": 0
  },
  "category": "Food/beverages",
  "is_published": true,
  "website": "http://www.coca-cola.com",
  "username": "coca-cola",
  "founded": "1886",
  "description": "Created in 1886 in Atlanta, Georgia, by Dr. John S. Pembe"
```

Compatible with your device

Makes JSON easy to read. Open source.

FEATURES

- JSON & JSONP support
- Syntax highlighting
- Collapsible trees, with indent guides
- Clickable URLs
- Toggle between raw and parsed JSON
- Works on any valid JSON page – URL doesn't matter
- Works on local files too (if you enable this in chrome://extensions)
- You can inspect the JSON by typing "json" in the console

[Website](#)

[Report Abuse](#)

Additional Information

Version: 0.6.0

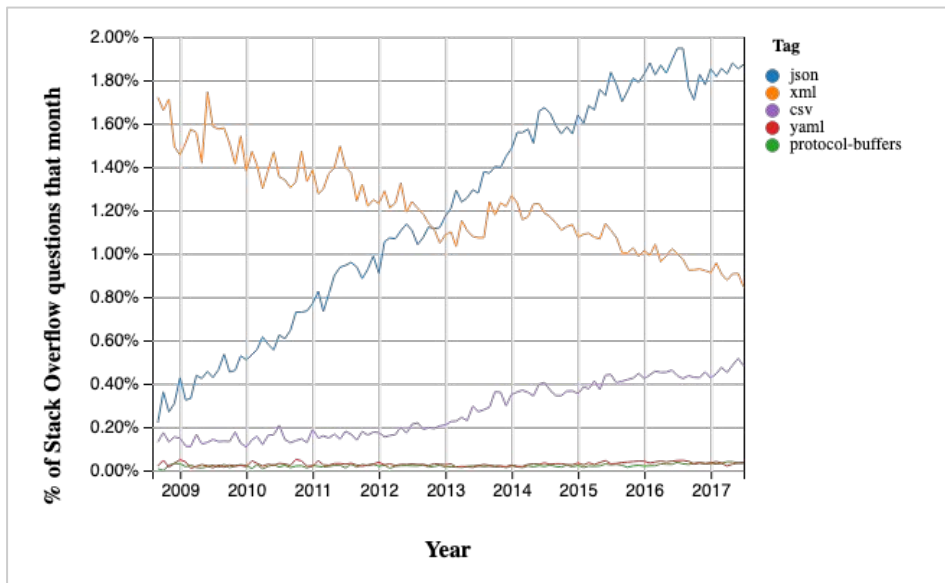
Updated: October 5, 2014

Size: 27.62KIB

Language: English



JSON es cada vez más el formato por defecto a la hora de compartir información entre sistemas web.



Fuente: <https://twobithistory.org/2017/09/21/the-rise-and-rise-of-json.html>



Realizar llamadas HTTP con JavaScript



Realizar llamadas HTTP (1/4)

Antiguamente, la forma de realizar **llamadas HTTP** con JavaScript era utilizando la funcionalidad [XMLHttpRequest](#).

Sin embargo, su dificultad de uso motivó a que en 2015 se implementase una nueva forma de realizar este tipo de llamadas usando la función [fetch](#) (nativa de JavaScript).

👉 También existen librerías externas que cumplen la misma funcionalidad como, por ejemplo, [Axios](#) (que es muy popular e incluye algunas ventajas).

Incluso la antigua librería jQuery tiene un método llamado [ajax](#) que permite hacer este tipo de llamados.



Realizar llamadas HTTP (2/4)

Ejemplo de uso de la función [fetch](#):

```
fetch("https://swapi.dev/api/people/1/?format=json")
  .then(function (datosDelServidor) {
    return datosDelServidor.json();
  })
  .then(function (personaje) {
    return console.log("Nombre del personaje: " + personaje.name);
  });
```

La sintaxis de la función `fetch` suele ser complicada de entender para los alumnos principiantes dado que está basada en el concepto de [promesas](#) de JavaScript. A continuación se realizará una explicación básica y lo más sencilla posible, contemplando los conocimientos teóricos que el alumno dispone al día de hoy.



Realizar llamadas HTTP (3/4)

Explicación de la función [fetch](#):

```
fetch("https://...")  
  
  .then(function(data) {return data.json()})  
  
  .then(function (data) { /* Hacer algo con data */ })  
  
  .catch(function (error) { alert(error) });
```

1. La función `fetch` recibe como parámetro un *string* con la **URL** a donde se desea realizar el llamado HTTP. A modo de simplificación, este código está diciendo: *“quiero obtener los datos que se encuentran en esta dirección”*.
2. El primer `then` es una función que se ejecuta cuando la llamada HTTP haya **finalizado** y se hayan recibido los datos (en este caso datos en formato JSON). La función `json` convierte el JSON recibido (que es un *string*) a un objeto de JavaScript.
3. El segundo `then` se utiliza al final del proceso (si es que el proceso fue exitoso). Aquí se indica qué se debe realizar con los datos recibidos. Aquí el parámetro `data` es un objeto JavaScript.
4. Opcionalmente, se puede definir una función `catch` que se ejecuta en caso de que haya ocurrido un error en el proceso.



Realizar llamadas HTTP (4/4) – Dev Tools

Probar de realizar una llamada HTTP (puede ser desde la consola del navegador) y observar la pestaña “*Network*” en las Developer Tools.

The screenshot shows the Chrome DevTools Network tab. The top bar includes tabs for Elements, Console, Sources, Network (selected), Performance, Memory, Application, Security, and Audits. Below the tabs, there are icons for a red dot, a circle with a slash, a video camera, and a funnel. The 'View' section shows a list icon, a tree icon, and checkboxes for 'Preserve log', 'Disable cache', and 'Offline'. The 'No throttling' option is selected. A filter input field is present. Below the filter, there are checkboxes for 'Regex' (checked), 'Hide data URLs', and 'All' (selected). The 'All' filter is active, showing a list of network requests. The first request is a 200 status XMLHttpRequest (xhr) from jquery.js:9664, with a size of 486B and a time of 346ms. The waterfall view shows a green bar for this request. At the bottom, it says '1 requests | 486B transferred'.

Name	Status	Type	Initiator	Size	Time	Waterfall
<input type="checkbox"/> ?format=json	200	xhr	jquery.js:9664	486B	346ms	

1 requests | 486B transferred




Un par de notas...



Loaders y mensajes de error

Cuando se hace un pedido al servidor (*request*) es posible que el **servidor demore** en contestar (y no se puede saber cuánto demorará). Incluso podría ocurrir que el **servidor nunca conteste** por estar caído. Por este motivo, es necesario tomar medidas que generen una mejor experiencia de usuario.

Por ejemplo:

- Usar “loaders”.
 - Ejemplo 1: <http://loading.io>. 
 - Ejemplo 2: <https://getbootstrap.com/docs/5.2/components/progress>.
 - Ejemplo 3: <https://fontawesome.com/how-to-use/on-the-web/styling/animating-icons>.
- Mostrar mensajes de error si la respuesta no es la deseada.
- Dar la opción de reintentar el *request* (la petición).



AJAX y SEO (1/2)

🔍 Existen sitios web donde el **Search Engine Optimization** (SEO) es fundamental. Por ejemplo: un sitio institucional, un blog, MercadoLibre, PedidosYa o cualquier sitio de e-commerce. Son sitios web cuyo contenido se quiere que aparezca en los primeros lugares de los resultados de búsqueda de Google (u otro *search engine*).

⚠ En estos casos, hay que tener mucho cuidado si nuestro sitio web consiste en un único HTML, el cual levanta todo el contenido a través de llamadas **AJAX**. A los “ojos” de Google, nuestro sitio tendría sólo una página *indexable*: `index.html`.

👉 Esto no siempre es un problema. En el caso de un sitio web privado, de uso interno en una empresa, el SEO no es relevante.



AJAX y SEO (2/2)

Un sitio web para el cual el SEO es importante, intentará tener la mayor cantidad de páginas *indexadas* en los motores de búsqueda.

Por ejemplo, **PedidosYa** se preocupó de que cada restaurante tenga su propia página, con su propia URL y que dicho contenido pueda ser *indexado* por Google:

- <https://www.pedidosya.com.uy/cadenas/la-pasiva>
- <https://www.pedidosya.com.uy/cadenas/chiviteria-marcos>
- <https://www.pedidosya.com.uy/restaurantes/montevideo/pastas-blanes-menu>

De esta forma, Google *indexa* cada una de estas páginas por separado, generando más contenido en el buscador, y aumentando las probabilidades de que alguien encuentre a PedidosYa en Google.

🤔 ¿Qué hacemos si queremos tener una *Single Page Application* y además nos preocupa el SEO? Afortunadamente se puede tener lo mejor de los dos mundos, pero requiere [trabajo extra](#).



Ejercicio 1



Ejercicio 1

Crear una página web que tenga un campo texto (`<input>`) y un botón “Buscar” (`<button>`) que **obtenga información sobre la película ingresada** en el `<input>` y **la muestre** en la parte derecha de la página. Utilizar la API de OMDb y consumir la información vía AJAX.

Película

Buscar película

Buscador de películas

Escribí el título de una película (en inglés) y obtené información sobre la misma.

Este servicio funciona gracias a la API de [OMDB](#) que retonra información sobre películas en formato JSON.

EXTRA

Agregar un *loader* a la página mientras se cargan los datos de la película. El *loader* se debe activar cuando comience la búsqueda (click en el botón) y desactivarse cuando termine la búsqueda.

Buscar película

[Buscar película](#)

Cinema Paradiso

Director: Giuseppe Tornatore

Argumento

A filmmaker recalls his childhood, when he fell in love with the movies at his village's theater and formed a deep friendship with the theater's projectionist.

Actores

Antonella Attili, Enzo Cannavale, Isa Danieli, Leo Gullotta

País

Italy, France

Premios

Won 1 Oscar. Another 21 wins & 25 nominations.

Ratings:

- Internet Movie Database - 8.5/10
- Rotten Tomatoes - 90%
- Metacritic - 80/100

[Ir al sitio web](#)



Ejercicio 1 (cont)

En caso de no encontrar una película, se debe mostrar un mensaje de error.

Película

Lo sentimos. No fue posible encontrar una película con el título buscado.

Tips:

1. Centrarse primero en obtener información de la API usando AJAX.
2. Luego intentar mostrar dicha información en la consola (para verificar que la información se está obteniendo correctamente).
3. Luego intentar mostrar la información en la página HTML. 🖱️ Sugérimos usar **Vue.js**.
4. Finalmente dedicarle tiempo a la estética de la página, mensajes de error, etc.



Ejercicio 2



Ejercicio 2

El servicio <http://jsonplaceholder.typicode.com/comments> provee 500 comentarios en formato JSON (que simulan ser los comentarios de un blog).

Se pide:

- Crear una página web que al cargarse muestre **sólo los primeros 10 comentarios** proveídos por el servicio JSON Placeholder.
- Cuando el usuario haga **scroll** y llegue al final de la página, **cargar los 10 comentarios siguientes**, y así sucesivamente.
- Como alternativa, en lugar de detectar el evento “*scroll*” se pueden cargar comentarios vía click en botón: “*Mostrar más comentarios*”.
- La idea es que en cada AJAX *request* se traigan sólo 10 comentarios.

La idea del ejercicio es simular el comportamiento del *feed* de Facebook, Twitter o Instagram.



Comentarios

id labore ex et quam laborum

laudantium enim quasi est quidem magnam voluptate ipsam eos tempora quo necessitatibus dolor quam autem quasi reiciendis et nam sapiente accusantium

ID del comentario: 1

quo vero reiciendis velit similique earum

est natus enim nihil est dolore omnis voluptatem numquam et omnis occaecati quod ullam at voluptatem error expedita pariatur nihil sint nostrum voluptatem reiciendis et

ID del comentario: 2

odio adipisci rerum aut animi

quia molestiae reprehenderit quasi aspernatur aut expedita occaecati aliquam eveniet laudantium omnis quibusdam delectus saepe quia accusamus maiores nam est cum et ducimus et vero voluptates excepturi deleniti ratione

ID del comentario: 3

alias odio sit

non et atque occaecati deserunt quas accusantium unde odit nobis qui voluptatem quia voluptas consequuntur itaque dolor et qui rerum deleniti ut occaecati

ID del comentario: 4

La URL que se debe llamar tiene el siguiente formato:

http://jsonplaceholder.typicode.com/comments?_start=10&_limit=10

El parámetro `_start` indica desde qué número de comentario se realiza la carga. En este ejemplo, a partir del comentario con id = 11. Probar dicha URL con distintos valores de `_start` para entender bien su funcionamiento.