APIs (Parte II)

AJAX

Competencias

- Realizar un request a una API utilizando AJAX y procesar el resultado agregando información al DOM.
- Codificar una rutina que consuma información de una API Rest para obtener datos externos.

Introducción

En este punto, comenzaremos a trabajar con la integración de una aplicación desde el cliente hacia el servidor y viceversa. Una de las cosas que han llevado a la web tal y como la conocemos, es la capacidad de comunicar sitios con bases de datos y servidores en todo el mundo, una página estática es el primer nivel para un desarrollador JavaScript. Ahora, hablaremos de dinamismo en la web y cómo podemos transformar una página web estática en un cliente dinámico, en comunicación con algún servidor, en donde, en base a la información que capturemos de dicho servidor, podemos renderizar y generar contenido en nuestro sitio. Generando una aplicación funcional, tal y como se trabaja en el mercado.

Para llevar a cabo este nuevo nivel de dinamismo es donde hablaremos de un nuevo concepto que está muy bien integrado con jQuery, AJAX (JavaScript Asíncrono y XML), una técnica que permite crear aplicaciones interactivas, mejorando los tiempos de carga, velocidad y usabilidad.

Comprender estos conceptos nos permitirá trabajar con estos recursos en nuestros proyectos, de acuerdo a los estándares y buenas prácticas, entendiendo los alcances y recursos con los que contamos para trabajar con APIs.

¿Qué es AJAX?

AJAX (Asynchronous JavaScript and XML) se refiere a un grupo de tecnologías que se utilizan para desarrollar aplicaciones web. Al combinar estas tecnologías, las páginas web son más receptivas, puesto que los paquetes pequeños de datos se intercambian con el servidor y las páginas web no se vuelven a cargar cada vez que un usuario realiza un cambio de entrada. AJAX, permite que un usuario de la aplicación web interactúe con una página web sin la interrupción que implica volver a cargar la página web. La interacción del sitio web ocurre rápidamente sólo con partes de la página de recarga y renovación.

AJAX se compone de las siguientes tecnologías para crear un nuevo enfoque al desarrollo de aplicaciones web:

- XHTML y CSS para presentar información.
- DOM (Document Object Model - modelo de objetos de documento) para visualizar e interactuar de forma dinámica la información presentada.
- El objeto XMLHttpRequest para manipular los datos de forma asíncrona con el servidor web.
- XML, HTML y XSLT para el intercambio y la manipulación de datos.
- Se visualiza JavaScript para enlazar solicitudes e información de datos.

Introducción al Concepto de API

Una interfaz de programación de aplicaciones (API) es un conjunto de herramientas, definiciones y protocolos que se usa para diseñar e integrar software de aplicaciones. Permite que un producto o servicio se comunique con otros productos y servicios, sin la necesidad de saber cómo se implementan. Las APIs simplifican el desarrollo de las aplicaciones, permitiendo ahorrar tiempo a los desarrolladores y generar un estándar o protocolo legible para futuros programadores que aborden dicho proyecto. Cuando se diseñan herramientas y productos nuevos, las APIs otorgan flexibilidad; simplifican el diseño, la administración y el uso; y proporcionan oportunidades para la innovación.

Las APIs son un medio simplificado para conectar tu proyecto a través del desarrollo de aplicaciones nativas de la nube, pero también le permiten compartir sus datos con clientes y otros usuarios externos. Las APIs públicas representan un valor comercial único porque simplifican y amplían la forma en que se conecta con sus partners y, además, pueden rentabilizar sus datos. Un ejemplo conocido es la API de Google Maps o de servicios de Share de facebook, incluso integraciones con servicios más pequeños que nos permiten obtener valores como el clima, provincias, ciudades, estados de un país o información de carácter global. Las APIs públicas son esenciales para el desarrollo de código abierto.

En resumen, las APIs le permiten habilitar el acceso a sus recursos y, al mismo tiempo, mantener la seguridad y el control, además de cómo habilitar el acceso y a quienes dependen de ti. La seguridad de las APIs tiene que ver con una buena gestión de ellas. Para conectarse a las APIs y crear aplicaciones que utilicen los datos o las funciones que estas ofrecen, se puede utilizar una plataforma de integración distribuida que conecte todos los elementos, incluidos los sistemas heredados y el Internet de las cosas (IoT).

Transferencia de Estado Representacional (REST)

Luego de una introducción teórica al concepto de API, cabe preguntarnos realmente cómo podemos comunicarnos con alguna de éstas de una forma estándar en donde, da igual la aplicación que consulta la información a un servidor, el resultado mediante un protocolo establecido siempre será el mismo. Así es como nace REST.

REST, es un estilo arquitectónico diseñado para proporcionar estándares entre los sistemas informáticos en la web, lo que facilita la comunicación entre éstos. Los sistemas compatibles con REST, a menudo llamados sistemas RESTful, se caracterizan por ser stateless, es decir, separan la lógica del cliente y el servidor.

Un servidor que opera bajo el paradigma REST, no importa qué tecnología utilice, solo importa definir de forma correcta los puntos de consulta (endpoints) que quedarán expuestos o abiertos, para que otro sistema a través de una url pueda hacer una petición HTTP para obtener información.

Peticiones (Request)

REST permite que un cliente pueda hacer peticiones HTTP a un servidor, este puede ser remoto, público o privado. Normalmente, una petición es una función que gatilla una acción, la cual puede recolectar datos o modificar datos de una base de datos remota alojada en dicho servidor.

Una petición normalmente consta con:

- Una acción HTTP. (Más adelante se detallarán acciones existentes)
- Una cabecera (header) que indica por lo general información que pueda necesitar el servidor para decodificar o interpretar la petición. Los headers, almacenan información que pueden otorgar seguridad al momento de hacer peticiones a servidores privados.
- Una ruta (PATH) en donde se hace la consulta
- De forma opcional y dependiendo del tipo de petición, puede contener opciones adicionales como un cuerpo (BODY) que puede contener cierto tipo de información.

Métodos HTTP

Para indicar el tipo de operación que se llevará a cabo, existen métodos HTTP para cada caso y de esta forma no tener verbos en las rutas.

Existen 4 acciones básicas HTTP que son utilizadas para interactuar con servidores:

- **GET:** Este método se utiliza para obtener un recurso específico o una colección de datos.
- **POST:** Este método se usa para crear recursos nuevos.
- **PUT:** Este método se usa para actualizar un recurso.
- **DELETE:** Este método se usa para eliminar recursos.

Estos conceptos se basan en la operación básica de orientación a objetos llamado CRUD. Te recomendamos investigar sobre estos términos a modo de complemento con esta lectura.

Cabeceras (headers)

Como se explicó anteriormente, las cabeceras permiten entregarle al servidor información adicional sobre cómo se espera recibir o enviar cierta información.

Por ejemplo:

```
GET /articles/23
Accept: text/html, application/json
```

Genera una acción de tipo get y en su cabecera se determina que dicha petición acepta como respuesta texto en formato de HTML o bien un objeto JSON.

La cantidad de cabeceras o condiciones a la petición son extensas, recomendamos leer la documentación oficial de Mozilla que indica la gama de posibilidades y condiciones que se pueden exponer en una cabecera.

Rutas (PATH)

La ruta es uno de los elementos más importantes a la hora de hacer una petición HTTP. Es la dirección física en donde se indica que consultar y a dónde hacerlo.

Entendamos el concepto con un ejemplo:
https://pokeapi.co/api/v2/pokemon/pikachu

Si analizamos en detalle esta ruta, podemos observar varios elementos:

- Un protocolo HTTP por el cual se hace la consulta.
- Una ruta base (BASE_URL) en este caso sería pokepi.co/api/v2/. Esta es la ruta base, la raíz a donde las peticiones irán en todo momento.
- Una ruta relativa (RELATIVE PATH) pokemon/pikachu que indica la acción, busca a un pokémon en específico, en este caso, pikachu.

La ruta del ejemplo se puede construir de forma genérica de la siguiente forma:

https://pokeapi.co/api/v2/pokemon/:pokemon_name

En donde las variables :pokemon_name pueden ser variables según lo que requiera la consulta.

Códigos de respuestas (Response Codes)

Cuando uno realiza una petición HTTP, además de traer información en su metadata, podemos observar diversos parámetros, uno de los más importantes son los códigos de respuestas a la petición realizada.

Dichos códigos nos permiten entender si la consulta fue realizada con éxito, si hay un problema con la consulta realizada por parte del cliente o bien, el servidor presenta un problema al momento de realizar la consulta.

Algunos códigos de respuestas conocidos:

- 200 (OK)
- 201 (CREATED)
- 204 (NO CONTENT)
- 400 (BAD REQUEST)
- 403 (FORBIDDEN)
- 404 (NOT FOUND)
- 500 (INTERNAL SERVER ERROR)

Para internalizar mejor estos conocimientos, invitamos a consultar la documentación oficial de Mozilla sobre los contextos de dichas respuestas y cómo utilizar esta información a beneficio de las personas que desarrollan aplicaciones.

Notación de Objetos de JavaScript (JSON)

Ya sabemos cómo realizar peticiones, en base a un formato estándar y conocido. Lo que nos queda por resolver es cómo recibimos y decodificamos la información. Es acá donde nace el concepto de JSON.
Por ejemplo, en las siguientes líneas podemos observar la estructura básica de un JSON.

```
{
  "customer" : {
    "name": "Scylla Buss"
    "email": "scyllabuss1@some.com"
  }
}
```

JSON es el acrónimo para JavaScript Object Notation y aunque su nombre lo diga, no es necesariamente parte de JavaScript, de hecho es un estándar basado en texto plano para el intercambio de información, por lo que se usa en muchos sistemas que requieren mostrar o enviar información para ser interpretada por otros sistemas, la ventaja de JSON al ser un formato que es independiente de cualquier lenguaje de programación, es que los servicios que comparten información por éste método, no necesitan hablar el mismo idioma, es decir, el emisor puede ser Java y el receptor PHP, cada lenguaje tiene su propia librería para codificar y decodificar cadenas de JSON.

JSON puede representar cuatro tipos primitivos (cadenas, números, booleanos, valores nulos) y dos tipos estructurados (objetos y arreglos).

En JSON:

- Una Cadena es una secuencia de ceros o más caracteres Unicode.
- Un Objeto es una colección desordenada de cero o más pares nombre:valor, donde un nombre es una cadena y un valor es una cadena, número, booleano, nulo, objeto o arreglo.
- Un Arreglo es una secuencia desordenada de ceros o más valores.

POSTMAN

Existe una herramienta muy potente para probar APIs sin necesidad de programar, nos ayuda a conocer la respuesta que tienen las APIs.

Instalando Postman

Podemos descargar Postman desde la página oficial, dentro del sitio busca el botón de descarga e instálalo en tu sistema operativo.

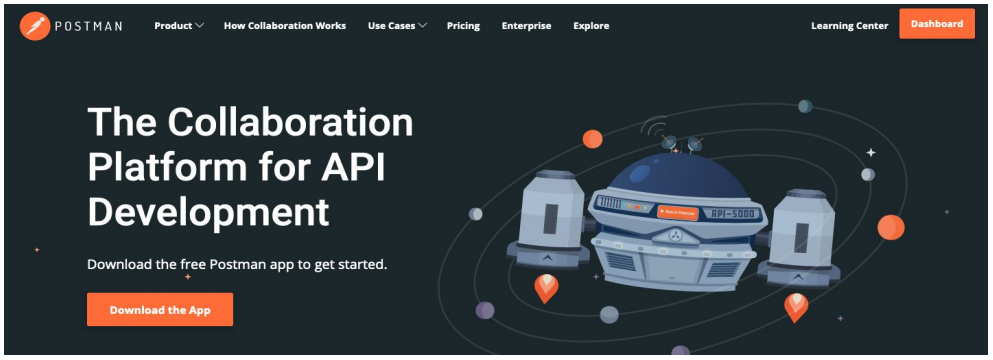

Imagen 1. Sitio Postman.
Fuente: Postman

Instalada la aplicación, ábrela y verás lo siguiente:

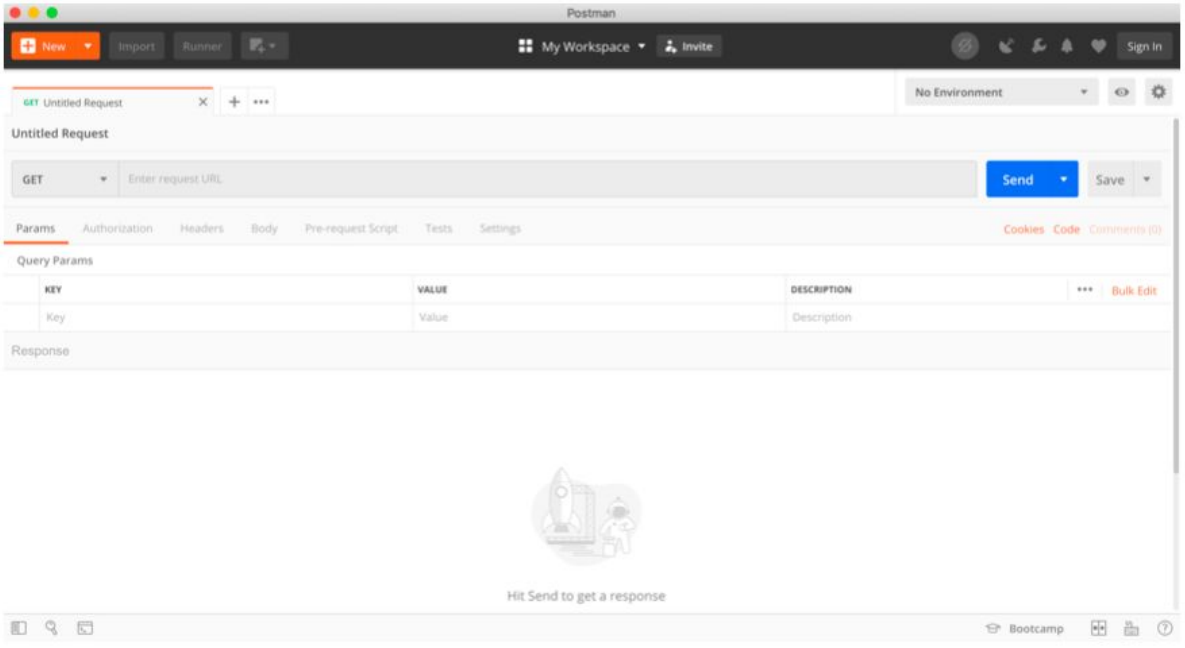

Imagen 2. Interfaz Postman.
Fuente: Postman

Ejercicio guiado: Nuestra primera consulta a una API usando Postman

Nuestra primera consulta será a jsonplaceholder. Es una API falsa en el sentido que no tiene información real, pero es muy completa y nos servirá para aprender a utilizar postman.

En este caso utilizaremos la información de "posts", que contiene un id de usuario, id del mensaje, título y cuerpo del mensaje en formato JSON. Puedes ver la información en la siguiente URL: https://jsonplaceholder.typicode.com/posts

- ***Paso 1:*** Donde dice Request URL pondremos el siguiente enlace: https://jsonplaceholder.typicode.com/posts

- ***Paso 2:*** Lo que haremos será realizar una petición GET para obtener un recurso, por lo tanto, se debe definir que es una petición de tipo GET para luego hacer click donde dice send y obtendremos nuestra respuesta.

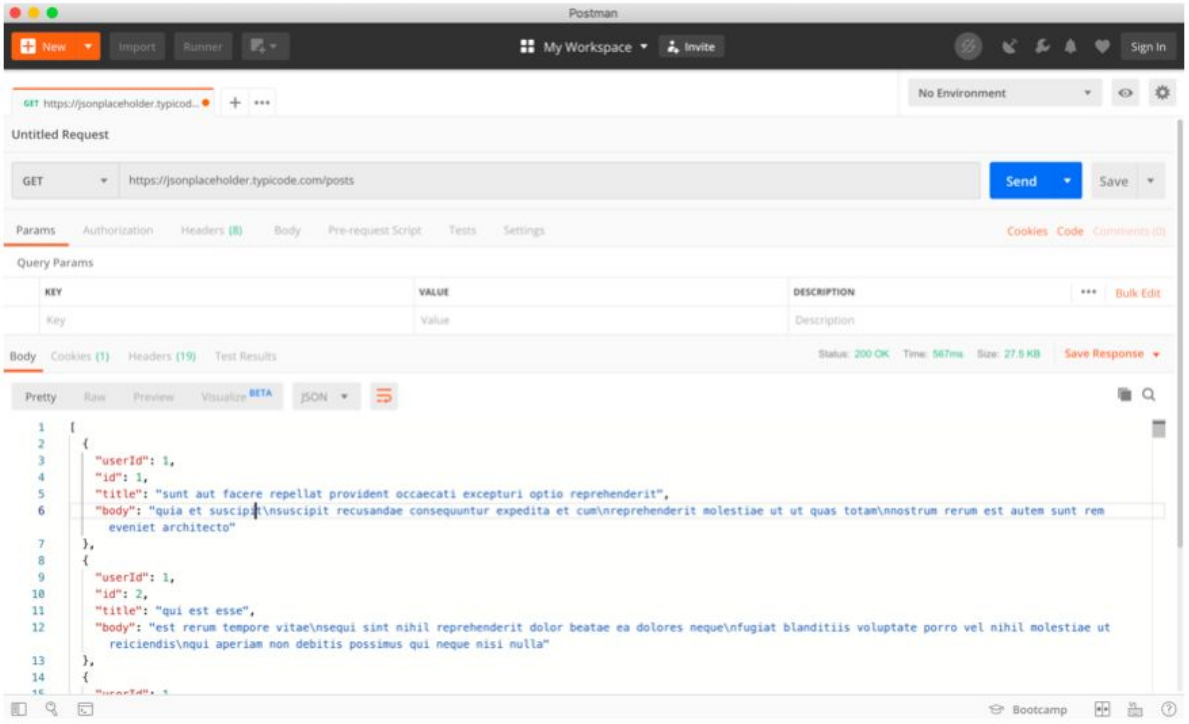

Imagen 3. Mi primer request.
Fuente: Postman

- ***Paso 3:*** Luego de algunos milisegundos el servidor nos enviará una respuesta. Ese es el texto que aparece en la parte inferior de la imagen 3. La respuesta está en formato JSON.

Ejercicio guiado: Obtener datos financieros

Consultar la siguiente API que nos muestra la información del dolar, UF, UTM y otros datos financieros en formato JSON. Si la consultas a través del navegador, verás su estructura; a través de POSTMAN podremos trabajar con ella, para obtener más adelante sólo aquellos datos que nos aporten a la información que queremos mostrar.

Por ahora consultar la URL y validar que nos responda con un código 200 (OK):

- ***Paso 1:*** Para esto copiaremos el siguiente endpoint https://mindicador.cl/api en Postman.

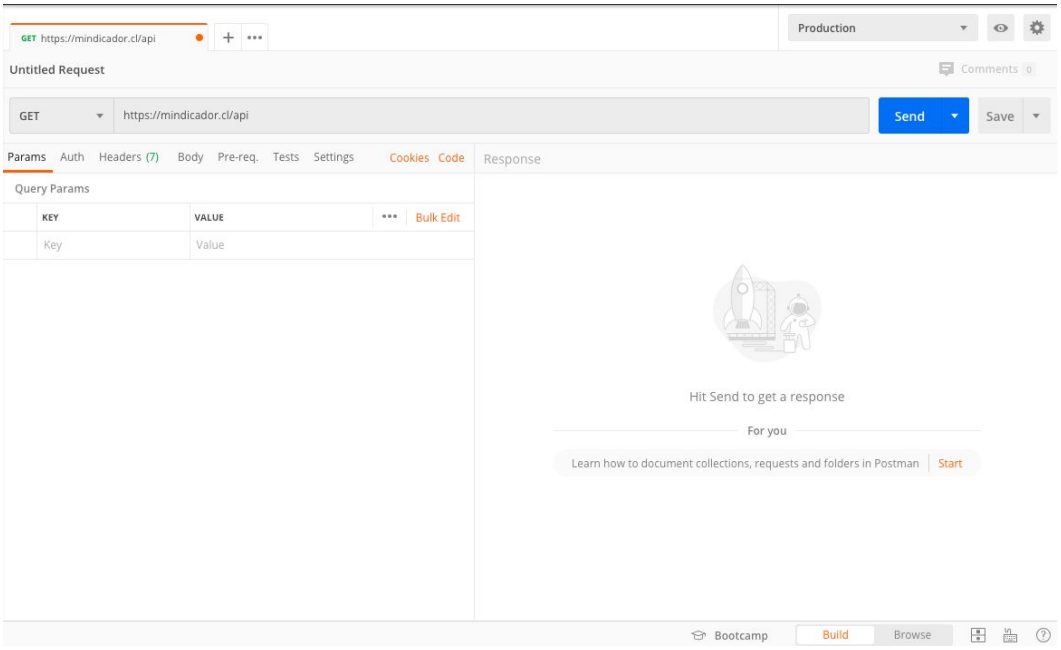

Imagen 4. Ingresar endpoint en Postman.
Fuente: Postman

- ***Paso 2:*** Ahora si apretamos el botón de "Send", ejecutaremos el request y obtendremos el resultado de la API.

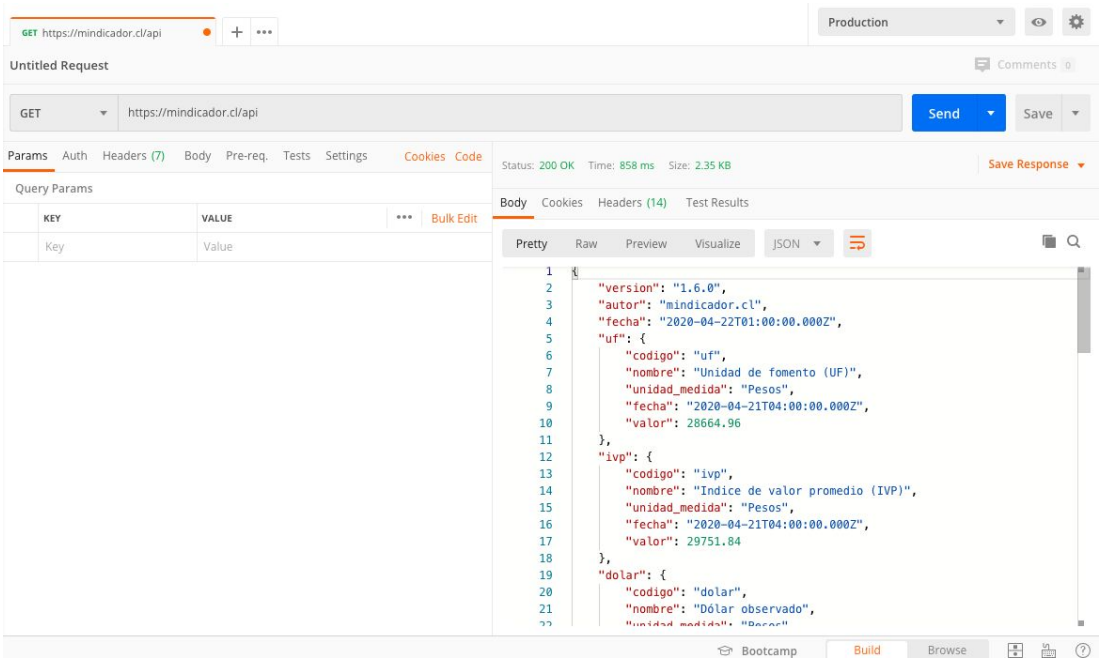

Imagen 5. Resultado API.
Fuente: Postman

Si todo está correcto deberías ver la estructura de la respuesta y código http 200.

Para conocer más APIs que pronto estarás utilizando, visita: Public APIs

Ejercicio propuesto (1)

Utiliza Postman para realizar una conexión, traer y mostrar la información de la siguiente API: https://reqres.in/api/users.

Utilizando AJAX

Competencia

- Crear un script que realice peticiones asíncronas utilizando la librería jQuery y AJAX para resolver un problema planteado.

Introducción

Ya hemos visto algunos elementos clave de la teoría que nos permitirán desarrollar APIs y obtener el mayor beneficio de los recursos tanto externos como propios, para incluir información de manera asíncrona en nuestros proyectos.

En este capítulo, profundizaremos el concepto de AJAX y utilizaremos esta herramienta para obtener información e incluirla en un sitio web.

Actualmente, en la web existen diversos recursos que nos pueden ayudar a resolver requisitos específicos en nuestros proyectos, como el valor del dólar o la UF que veíamos en los ejemplos anteriores. El poder consultar estos datos y manipularlos, son competencias altamente valoradas en el mundo laboral.

Utilizando AJAX

Anteriormente se pudo leer sobre el concepto y finalidad de AJAX, pero hay algo importante que se debe mencionar y tener en cuenta cuando trabajamos con AJAX. Por defecto y su acrónimo así lo indica, todas las solicitudes que se realicen implementando jQuery-AJAX se envían de forma asincrónica, ya que es su valor por defecto. Esto, se puede observar en la estructura básica de una petición AJAX:

```
$.ajax({
   type:"tipo de petición, puede ser GET, POST, PUT...",
   url:"aquí va la url a consultar",
   dataType:"puede ser xml, json, script, or html",
   success: function(data) {
     //si todo sale bien, se agrega la funcionalidad aquí
   },
   error: function(data) {
     //esta función se activa si ocurre algún error durante el proceso
   },
   async: true,
});
```

Vamos a ir dejando la teoría un poco de lado para aplicar los conceptos aprendidos utilizando AJAX. Para que la explicación sea más sencilla, consideraremos un ejemplo en donde un usuario puede hacer click sobre un botón y mostrar primeramente abajo de ese botón el mensaje: `'loading . . .'`. Seguidamente se debe conectar a una API para traer la información mediante el método GET y mostrarla al usuario eliminando el mensaje anterior. La API a la cual se debe conectar el código es: Meetup. Por ende, el extracto del código HTML a utilizar que gatilla dicha acción sería:

```
<button type="button" class="btn">Click me!</button>
<p class="text">Replace me!!</p>
<script src="https://code.jQuery.com/jQuery-2.1.1.min.js"></script>
```

Mientras que el código en JavaScript implementando jQuery es:

```
    $('.btn').click(function() {
      $('.text').text('loading . . .');

      $.ajax({
        type:"GET",
        url:"https://api.meetup.com/2/cities",
```

```
        success: function(data) {
          $('.text').text(JSON.stringify(data));
        },
        dataType: 'json',
      });
    });
```

Vamos analizando paso por paso la solicitud.

Tenemos una función que se gatilla al hacer click en el botón de clase btn. Al realizar la acción de click, reemplazamos el texto presente en la etiqueta p con otro contenido y de forma asíncrona realizamos una consulta HTTP utilizando una función AJAX.

Dicha función recibe varios parámetros:

- type: indica el tipo de la petición
- url: el PATH al cual se hará la consulta
- success: una función que gatilla una acción cuando la petición fue realizada con éxito.

Al gatillar la función de éxito, se reemplaza el contenido en p con la respuesta recibida por la API. En este caso se aplica una función que transforma la respuesta en formato JSON en una cadena de caracteres (string) de esa forma entregamos una respuesta comprimida del resultado. Como pueden observar, una vez interiorizados los conceptos de API y JSON, es sencillo armar peticiones AJAX considerando los elementos necesarios para realizar dicha petición.

Considera además, que todas las peticiones son asíncronas, es decir, si esperas la respuesta de una petición para ejecutar otras acciones, siempre declara estas funciones dentro de la opción de tipo success. De esta forma nos aseguramos que una vez tengamos la respuesta, ejecutaremos la acción esperada.

Ejercicio guiado: Obtener datos financieros con AJAX

¿Recuerdan el endpoint de información financiera que revisamos en Postman? ¿Te parece si llevamos esa API a nuestro sitio web? Lo que haremos será utilizar la API de indicadores económicos (https://mindicador.cl/api) y, mediante un llamado AJAX, mostraremos la información de la UF en un proyecto web. Para esto, utilizaremos el siguiente extracto HTML:

```
<div>
  <h2>El valor de la UF hoy es de <span id="valorUf"></span></h2>
</div>
```

Tómate unos minutos e intenta hacerlo tu... ¿Lo lograste?. Si la respuesta es no, no te preocupes y vamos en conjunto realizando el código.

- ***Paso 1:*** Lo primero que haremos será escribir un código HTML en donde mostraremos el valor de la UF:

```
<div>
  <h2>El valor de la UF hoy es de <span id="valorUf"></span></h2>
</div>
```

- ***Paso 2:*** Tenemos nuestra estructura básica, ahora escribiremos nuestro código con jQuery para hacer el llamado, esta acción se debe ejecutar cargada la página así que no debe llamarse al endpoint con algún click o algo por el estilo, por lo que usaremos $.ajax y completamos la estructura con los datos necesarios.

```
    $(document).ready(function(){
        $.ajax({
            type:"GET",
            url:"https://mindicador.cl/api",
            dataType:"json",
            success: function(data) {
                        console.log(data);
                  }
        });
    });
```

Antes de continuar es importante aclarar que el usar el nombre data como argumento de la función, es un nombre que se usa por convención entre los desarrolladores, pero podrías usar el nombre que quieras, muchos ejemplos en internet nombran este valor como response, res, etc. Aclarado este punto veamos que nos entrega data al hacer un console.log

```
Object { version: "1.6.0", autor: "mindicador.cl", fecha:
"2020-04-22T02:00:00.000Z", uf: {...}, ivp: {...}, dolar: {...},
dolar_intercambio: {...}, euro: {...}, ipc: {...}, utm: {...}, ... }
```

- ***Paso 3:*** Como podemos apreciar es la misma respuesta que ya conocíamos por Postman, ahora que tenemos nuestro llamado realizado, lo único que debemos hacer es llegar al valor de la UF, esto lo hacemos recorriendo el JSON de respuesta con notación de puntos (data.uf.valor).

```
$(document).ready(function(){
    $.ajax({
        type:"GET",
        url:"https://mindicador.cl/api",
        dataType:"json",
        success: function(data) {
                    console.log(data.uf.valor);
                }
    });
});
```

- ***Paso 4:*** Si revisamos ahora deberíamos ver 28664.96 ya tenemos el valor ahora con jQuery debemos agregarlo en la etiqueta span de nuestro html.

```
$(document).ready(function(){
    $.ajax({
        type:"GET",
        url:"https://mindicador.cl/api",
        dataType:"json",
        success: function(data) {
                    $('#valorUf').text(data.uf.valor);
                }
    });
});
```

Lo que hicimos fue usar el selector de ID y con el método text agregar el valor que recibimos desde la API, si todo salio bien deberías ver lo siguiente en tu navegador:

```
El valor de la UF hoy es de 28664.96
```

El valor de la UF puede variar dependiendo cuando realices el ejercicio.

Ejercicio guiado: Utilizando AJAX

Realizar la conexión a siguiente API que cuenta con datos de usuario, como id, email, nombre, apellido y una imagen. En base al siguiente HTML, una vez que el usuario haga click en el botón, se obtendrá la información de la API y será mostrada en el documento web mediante una función de AJAX:

```
<h4>Información de la API</h4>
<div>
   <button type="button">Mostrar Información</button>
   <div class="resultado"></div>
</div>
```

- ***Paso 1*:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js.

- ***Paso 2*:** En el index.html, escribir la estructura básica de un documento HTML, incorporar el enlace o CDN de jQuery, que nos permita incluir esta librería en nuestro ejemplo, agregar el extracto del código HTML entregado en el enunciado y enlaza el archivo externo denominado “script.js”, como se muestra a continuación:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"><
/script>
   <title>Document</title>
</head>
<body>
   <h4>Información de la API</h4>
   <div>
      <button type="button">Mostrar Información</button>
      <div class="resultado"></div>
   </div>
   <script src="script.js"></script>
</body>
</html>
```

- ***Paso 3*:** Ahora en el archivo script.js, se debe agregar la función de AJAX para realizar una petición del tipo GET.

```
$.ajax({
   type:"GET",
   url:"aquí va la url a consultar",
   dataType:"json",
   success: function(data) {
     //si todo sale bien, se agrega la funcionalidad aquí.
   },
   error: function(error) {
     //si todo sale bien, se agrega la funcionalidad aquí.
   },
});
```

- ***Paso 4*:** Ya dispuesta la estructura de AJAX para una petición del tipo GET, se debe agregar la función principal de jQuery, además el escucha y manejador de evento para el botón, porque la idea es que cuando el usuario haga un click sobre el botón, se realice el llamado a la API, se traiga la información y se muestre.

```
$(document).ready(function(){
   $('button').on('click',function(){
      $.ajax({
         type:"GET",
         url:"https://reqres.in/api/users",
         dataType:"json",
         success: function(data) {
           //si todo sale bien, se agrega la funcionalidad aquí.
         },
         error: function(error) {
            //si todo sale bien, se agrega la funcionalidad aquí.
         },
      });
   });
});
```

- ***Paso 5*:** Con la funcionalidad lista de AJAX más el evento “click” activo en el botón. Agrega dentro de la función “success” de AJAX, lo que debe ocurrir cuando la información que retorne la API esté completa y sea correcta la conexión. Mientras que en la función “error”, se desplegará la información para indicar que hubo un error con la conexión a la API. En este caso, vamos a mostrar la información primeramente en un console.log, para ver que trae la API (es recomendable utilizar

postman para ver la estructura de la información). En este ejercicio, la información la estamos recibiendo en una variable denominada "datosApi" y dentro de ella se extrae la data original que viene con la API para esa petición en específico.

```
$(document).ready(function(){
   $('button').on('click',function(){
      $.ajax({
         type:"GET",
         url:"https://reqres.in/api/users",
         dataType:"json",
         success: function(datosApi) {
            console.log(datosApi.data);
         }
         error: function(error) {
            //si todo sale bien, se agrega la funcionalidad aquí.
         },
      });
   });
});
```

- ***Paso 6*:** Al ejecutar el código anterior y si la conexión fue exitosa, en la consola del navegador el resultado debe ser:

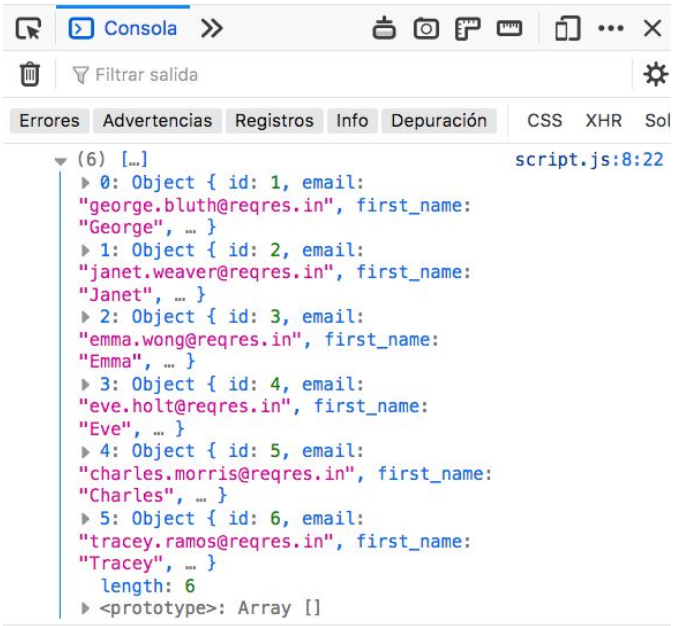

Imagen 6. Información proveniente de la API.
Fuente: Desafío Latam

- ***Paso 7*:** Ahora como ya tenemos la información proveniente de la API lista para usar, vamos a mostrar esa información dentro de la etiqueta con clase "resultado".

Por lo tanto, se debe captar el elemento por la clase “resultado” para luego mediante el método “append” de jQuery podremos agregar una simple etiqueta de párrafo para la data que retorna la API. Esta información viene encapsulada dentro de un arreglo, y dentro del arreglo son distintos objetos que contienen el contenido, entonces se hace necesario el uso de forEach para recorrer el arreglo.

```
$(document).ready(function(){
   $('button').on('click',function(){
      $.ajax({
         type:"GET",
         url:"https://reqres.in/api/users",
         dataType:"json",
         success: function(datosApi) {
            console.log(datosApi.data);
            datosApi.data.forEach(element => {
               $('.resultado')
                  .append(`<p>${element.id}-${element.email}-
                        ${element.first_name}-${element.last_name}-
                        ${element.avatar}</p>`);
             })
         },
         error: function(error) {
             //si algo sale mal, se agrega la funcionalidad aquí.
         },
      });
   });
});
```

- ***Paso 8***: Al ejecutar el código anterior en nuestro navegador web y hacer clic sobre el botón “Mostrar Información”, el resultado debería ser:

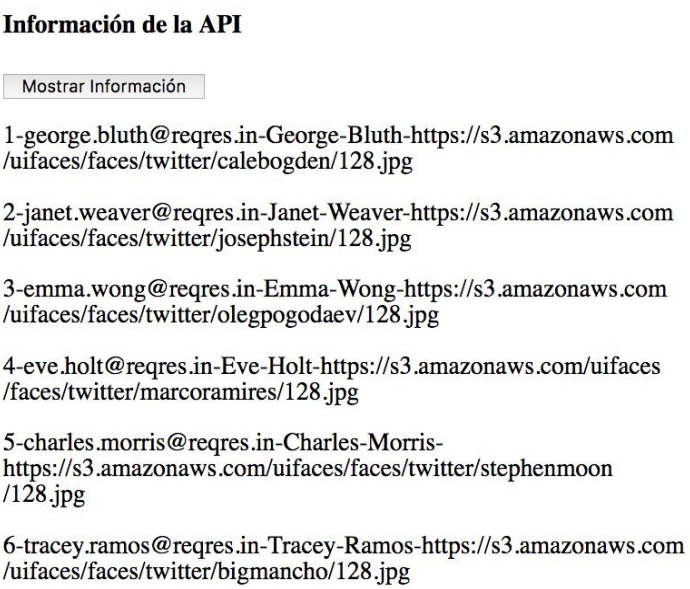

Imagen 7. Mostrando información proveniente de la API en el documento web.
Fuente: Desafío Latam

Ejercicio propuesto (2)

Para la siguiente dirección web (perteneciente a una API): https://jsonplaceholder.typicode.com/users. Realiza la respectiva conexión mediante AJAX y luego muestra la información solo cuando un usuario haga click sobre un botón.

Plugins

Competencias

- Crear plugins propios con jQuery para ampliar las funcionalidades de una aplicación web.
- Integrar plugins existentes en algún proyecto para incluir recursos externos en un proyecto web.

Introducción

Si has llegado a este punto y has puesto en práctica los ejercicios realizados hasta ahora, no me cabe duda que tendrás ya una pequeña idea de las cosas que se pueden hacer con el framework. Habrás comprobado que con pocas líneas de código, se pueden hacer diversos efectos y dotar a la página de interacción con el usuario, pero quizás todavía te sientas un poco perdido a la hora de encarar el desarrollo de problemas más complejos con los que podrás enfrentarte.

Es acá donde le damos un punto final (por ahora) a la infinita posibilidad de cosas que se pueden hacer con jQuery, hablaremos de los Plugins, que son fragmentos de código que amplían las funcionalidades de los proyectos web. Estas herramientas son muy utilizadas en la actualidad y facilitan la utilización de estándares para trabajar con recursos de terceros.

Entendiendo el Concepto de Plugin

Los plugins son la utilidad que pone jQuery a disposición de los desarrolladores para ampliar las funcionalidades del framework. Por lo general, servirán para hacer cosas más complejas y necesarias para resolver necesidades específicas, pero las hacen de manera que puedan utilizarse en el futuro en cualquier parte y por cualquier web.

En la práctica un plugin no es más que una función que se añade al objeto jQuery (objeto básico de este framework que devuelve la función jQuery para un selector dado), para que a partir de ese momento responda a nuevos métodos. Como ya sabemos, en este framework todo está basado en el objeto jQuery, así que con los plugins podemos añadirle nuevas utilidades.

¿Te imaginas creando tus propias capas de funcionalidades añadidas a jQuery?
En este capítulo te explicaremos cómo diseñar tus propias soluciones y utilizar soluciones populares trabajadas por terceros.

Creación de un Plugin

Para poder crear un plugin es necesario aplicar y conocer ciertas reglas que son necesarias para respetar la estructura de un plugin como tal. Por ejemplo:

- El archivo que crees con el código de tu plugin lo debes nombrar como jQuery.[nombre de tu plugin].js. Por ejemplo jQuery.desaparece.js.

- Añade las funciones como nuevos métodos por medio de la propiedad fn del objeto jQuery, para que se conviertan en métodos del propio objeto jQuery.

- Dentro de los métodos que añades como plugins, la palabra "this" será una referencia al objeto jQuery que recibe el método. Por tanto, podemos utilizar "this" para acceder a cualquier propiedad del elemento de la página con el que estamos trabajando.

- Debes colocar un punto y coma ";" al final de cada método que crees como plugin, para que el código fuente se pueda comprimir y siga funcionando correctamente. Ese punto y coma debes colocarlo después de cerrar la llave del código de la función.

- El método debe retornar el propio objeto jQuery sobre el que se solicitó la ejecución del plugin. Esto lo podemos conseguir con un return this; al final del código de la función.

- Se debe usar this.each para iterar sobre todo el conjunto de elementos que puede haber seleccionados. Recordemos que los plugins se invocan sobre objetos que se obtienen con selectores y la función jQuery, por lo que pueden haberse seleccionado varios elementos y no sólo uno. Así pues, con this.each podemos iterar sobre cada uno de esos elementos seleccionados. Esto es interesante para producir código limpio, que además será compatible con selectores que correspondan con varios elementos de la página.

- Asigna el plugin siempre al objeto jQuery, en vez de hacerlo sobre el símbolo $, así los usuarios podrán usar alias personalizados para ese plugin a través del método noConfict(), descartando los problemas que puedan haber, si dos plugin tienen el mismo nombre.

Una vez conocidas estas reglas fundamentales, revisemos un ejemplo:

```
    jQuery.fn.parpadea = function() {
       this.each(function(){
          elem = $(this);
          elem.fadeOut(250, function(){
             $(this).fadeIn(250);
          });
       });
       return this;
    };
```

Ya tenemos nuestro primer plugin. Como puedes observar en el ejemplo, para generar una función del plugin debemos invocar la función jQuery.fn.[nombre_de_la_función] de esta forma jQuery entenderá que es una función diseñada por tí y propia. En particular esta función genera una especie de parpadeo suave en un elemento.

Bueno, ahora la queremos usar, entonces es sencillo:

```
    $(document).ready(function(){
       //parpadean los elementos de class CSS "parpadear"
       $(".parpadear").parpadea();

       //añado evento click para un botón. Al pulsar parpadearán los
elementos de clase parpadear
       $("#botonparpadear").click(function(){
          $(".parpadear").parpadea();
       })
    })
```

Ejercicio guiado: Creación de un plugin de datos financieros

Desarrollar un plugin que permita consultar el valor de la UF, utilizando la API https://mindicador.cl/api de datos financieros. Utilizar el siguiente extracto del HTML:

```
<div id="miCaja">
   Valor de la UF
</div>
```

- ***Paso 1*:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js.

- ***Paso 2*:** En el index.html, escribir la estructura básica de un documento HTML, incorporar el enlace o CDN de jQuery, que nos permita incluir esta librería en nuestro ejemplo, agregar el extracto del código HTML entregado en el enunciado y enlaza el archivo externo denominado "script.js", como se muestra a continuación:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"><
/script>
   <title>Document</title>
</head>
<body>
   <div id="miCaja">
       Valor de la UF
   </div>
   <script src="script.js"></script>
</body>
</html>
```

- ***Paso 3*:** Crear un archivo nuevo con el nombre de: jQuery.datosFinancieros.js dentro de la carpeta de trabajo, luego, crear el esquema básico y definir un nombre, por ejemplo datosFinancieros, que fue el nombre que se le dio al archivo en sí.

```
jQuery.fn.datosFinancieros = function() {
    return this;
};
```

- ***Paso 4:*** Dentro de nuestro plugin llamar a la API como lo hicimos anteriormente (podemos copiar y pegar el código)

```
jQuery.fn.datosFinancieros = function() {
    $.ajax({
        type:"GET",
        url:"https://mindicador.cl/api",
        dataType:"json",
        success: function(data) {
        }
    });
   return this;
};
```

- ***Paso 5:*** Agregar el valor de la UF al elemento que use nuestro plugin y lo haremos a continuación:

```
jQuery.fn.datosFinancieros = function() {
  var element = this;
    $.ajax({
        type:"GET",
        url:"https://mindicador.cl/api",
        dataType:"json",
        success: function(data) {
              element.append(
                 `<span>${data.uf.valor}</span>`
              )
        }
    });
   return this;
};
```

- **Paso 6:** Creamos una variable element que le dimos el valor de this, esto debido a que si usamos this dentro de la función success, el this en ese contexto pertenece $.ajax y no a nuestro plugin, aclarado esto ahora podemos usar las bondades de jQuery para poder agregar un valor a nuestros elementos html. Ahora que tenemos nuestro plugin usémoslo llamándolo desde el archivo script.js. Mientras que en el

archivo index.html, se debe agregar exactamente abajo de la inclusión del script.js, un nuevo enlace para el plugin mediante la instrucción: <script src="jQuery.datosFinancieros.js"></script>.

```
$(document).ready(function(){
   $('#miCaja').datosFinancieros();
});
```

- ***Paso 7:*** Si todo está bien deberías ver lo siguiente:

Valor de la UF 28687.8

Imagen 8. Resultado final al implementar el plugin.
Fuente: Desafío Latam

Ejercicio propuesto (3)

Crea un plugin con jQuery para conectarse, traer y mostrar la información en el documento web para al siguiente api: https://jsonplaceholder.typicode.com/users. Realiza la respectiva conexión mediante AJAX y muestra la información solo cuando un usuario haga click sobre un botón.

Integración de Plugins Externos basados en jQuery a un Proyecto

Muchos desarrolladores en el mundo, han aprovechado la potencia de jQuery para crear sus propios plugins basados en jQuery para generar proyectos más complejos y específicos. Estos proyectos complejos a su vez, los más populares siempre tienen soporte, son open source y abiertos para su uso. Permiten a equipos de desarrollo de pequeñas y medianas empresas, tener un punto de inicio para sus proyectos.

La integración de un plugin en un proyecto puede ser muy sencilla, la forma que recomendamos es que lo integres vía CDN (Content Delivery Network). En resumen, CDN es una url en donde el código de la librería es accesible a través de dicho link, de esa forma te aseguras que tu integración estará siempre en línea.

Es muy importante que antes de integrar el plugin tengas integrada la librería de jQuery, los script html los lee de forma secuencial, si no encuentra jQuery en tu proyecto antes que el plugin, tendrás errores al momento de ejecutar las funciones particulares que te ofrece el plugin.

Algunos Plugins Comunes

CanvasJS

Este plugin es uno de los más populares porque te permite crear gráficas diversas, simplemente con añadir dicho plugin y la librería de jQuery. Por ende, se utilizó en el ejercicio anteriormente realizado. Por lo general, las librerías complejas que existen para generar gráficas son muy pesadas o requieren una instalación previa, tienden a sobrecargar el servidor en donde se encuentra alojada una aplicación web. En este sentido, CanvasJS es muy interesante puesto que es una librería ligera y sencilla de implementar.

Ejercicio guiado: CanvasJS

Implementar la librería de CanvasJS con gráficos de columnas, con datos ya preestablecidos directamente en el código de JavaScript. Por lo tanto, en este caso el ejercicio solicita graficar el consumo de frutas tropicales en kilogramos por persona al año (datos ficticios), basados en la información suministrada por el ente encargado de llevar las estadísticas, siendo la información la siguiente: “Papaya: 23, Naranaja: 15, Platano: 25, Mango: 30 y Guayaba: 20”. A partir de la documentación facilitada por la página web CanvasJS, crear la estructura de la gráfica a utilizar usando el siguiente extracto de código HTML, extraído de la documentación de CanvasJS:

```
<div id="chartContainer" style="height: 300px; width: 100%;"></div>
```

- ***Paso 1*:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js.

- ***Paso 2*:** En el index.html, escribir la estructura básica de un documento HTML, incorporar el enlace o CDN de jQuery y CanvasJS, los cuales permitirán incluir estas librerías, luego, agregar el extracto del código HTML entregado en el enunciado y enlaza el archivo externo denominado “script.js”, como se muestra a continuación:

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width,
initial-scale=1.0">
      <title>CanvasJS</title>
</head>
<body>
      <div id="chartContainer" style="height: 300px; width: 100%;"></div>
      <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"><
/script>
      <script
src="https://canvasjs.com/assets/script/jquery.canvasjs.min.js"></script
>
      <script src="script.js"></script>
</body>
</html>
```

- ***Paso 3:*** Ahora en el archivo script.js, crear la estructura necesaria para poder graficar los datos indicados en el enunciado sobre el consumo de las frutas tropicales. En este caso, utilizamos como primera instrucción de JavaScript el evento load de window, que permitirá ejecutar la función que le definamos cuando el documento quede cargado. Luego dentro de esta función, se captura el id del elemento HTML para poder activar el método correspondiente a CanvasJS, denominada `CanvasJSChart`, al cual le pasaremos toda la configuración necesaria para mostrar la gráfica.

```
window.onload = function () {
  $("#chartContainer").CanvasJSChart(options);
};
```

- ***Paso 4:*** Crear las opciones que se le pasan al método, estas opciones serán un objeto con todos los datos, como el caso del título de la gráfica, el título y tamaño del título que tendrán los ejes de las abscisas y las ordenadas. Igualmente, el arreglo `data` que llevará el tipo de gráfico, en este caso `column` y los puntos a gráficas `dataPoints`, en donde el `label` es para las abscisas y las "y" para las ordenadas.

```
window.onload = function () {
  var options = {
      title: {
         text: "Gráfico de columnas con jQuery CanvasJS",
      },
      axisX:{
         title : "Frutas Tropicales",
         titleFontSize: 12
      },
      axisY:{
         title : "Consumo Kg/persona/año",
         titleFontSize: 12
      },
      data: [
         {
            type: "column",
            dataPoints: [
               { label: "papaya", y: 23 },
               { label: "naranja", y: 15 },
               { label: "platano", y: 25 },
               { label: "mango", y: 30 },
               { label: "guayaba", y: 20 },
            ],
         },
      ],
  };
  $("#chartContainer").CanvasJSChart(options);
};
```

- ***Paso 5:*** Al ejecutar el archivo index.html en el navegador web, encontraremos el siguiente resultado:

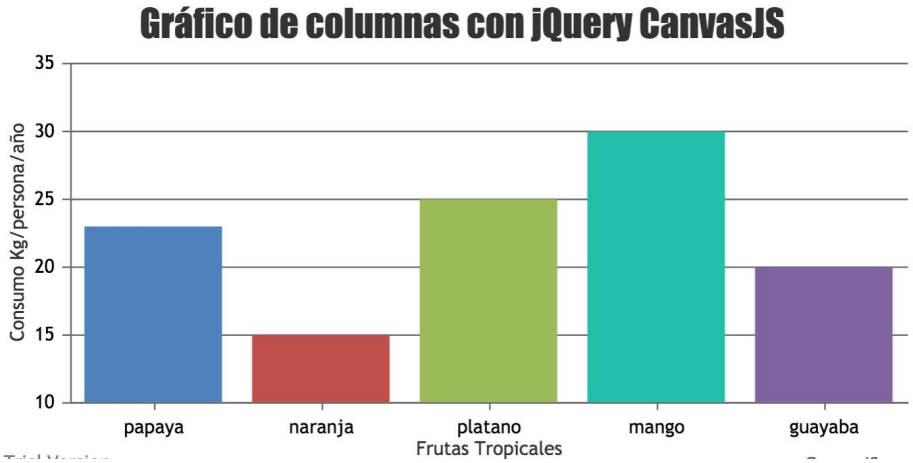

Imagen 9. Resultado de la aplicación.
Fuente: Desafío Latam

En el ejemplo anterior podemos observar una forma de aplicar la librería a un proyecto, si revisan la documentación oficial, notarán que sus gráficas se pueden customizar y añadir una enorme cantidad de parámetros en formato JSON.

Ejercicio guiado: jQuery, AJAX y CanvasJS

Realizar un ejercicio implementado jQuery, AJAX y CanvasJS. El cual, consiste en mostrar la información del dolar según la API: https://mindicador.cl/api/dolar, en una gráfica dispuesta y configurada con CanvasJS, dicha configuración y documentación de la gráfica se puede encontrar en: jQuery Charts & Graphs from JSON Data Using AJAX. A partir de la documentación facilitada por la página web, se creará la estructura de la gráfica a utilizar. Para mostrar la gráfica, puedes usar el siguiente extracto de código HTML extraído de la documentación de CanvasJS:

```
<div id="chartContainer" style="height: 300px; width: 100%;"></div>
```

- ***Paso 1*:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js.

- ***Paso 2*:** En el index.html, escribir la estructura básica de un documento HTML, incorporar el enlace o CDN de jQuery y CanvasJS, los cuales, permitirán incluir estas librería en nuestro ejemplo, luego, agregar el extracto del código HTML entregado en el enunciado y enlaza el archivo externo denominado “script.js”, como se muestra a continuación:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"><
/script>
   <script
src="https://canvasjs.com/assets/script/jquery.canvasjs.min.js"></script
>
   <title>Usando CanvasJS</title>
</head>
<body>
   <div id="chartContainer" style="height: 300px; width: 100%;"></div>
   <script src="script.js"></script>
</body>
</html>
```

- ***Paso 3***: Ya ubicados en el archivo script.js, se debe incorporar la función de jQuery, luego dentro de ella, realizamos la conexión mediante AJAX a la API indicada y mostremos en consola los datos recibidos, en este caso se guardarán primero en una variable para futuras implementaciones.

```
$(document).ready(function(){
      $.ajax({
         type:"GET",
         url:"https://mindicador.cl/api/dolar",
         dataType:"json",
         success: function(datos) {
            let datosApi = datos.serie;
            console.log(datosApi);
         },
         error: function(error) {
            console.log(error)
         }
      });
});
```

- ***Paso 4***: Al ejecutar el código anterior y revisar la consola del navegador web, el resultado encontrado debería ser:

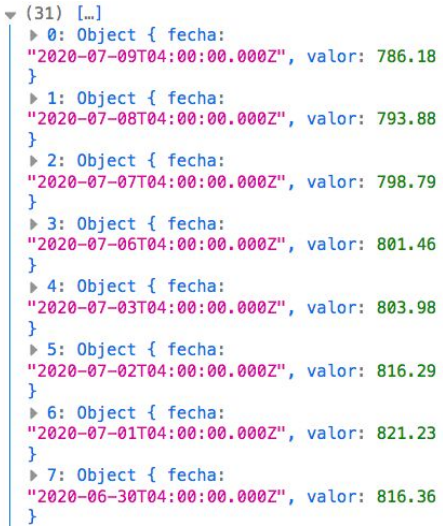

Imagen 10. Resultado en consola de la API.
Fuente: Desafío Latam

- ***Paso 5***: Ya comprobada la conexión con la API y que la información está llegando correctamente, pasamos a agregar la configuración para la gráfica, esta consiste en dos variables, una que será el arreglo que contenga la información extraída de la API mediante un ciclo repetitivo "for" y otra será un objeto con la configuración básica para la visualización de los datos en la gráfica, esta configuración es extraída de la documentación de CanvasJS :

```
$(document).ready(function(){
      var dataPoints = [];

      var options =  {
      animationEnabled: true,
      theme: "light2",
      title: {
            text: "Daily Sales Data"
      },
      axisX: {
            valueFormatString: "DD MMM YYYY",
      },
      axisY: {
            title: "USD",
            titleFontSize: 24,
            includeZero: false
      },
      data: [{
            type: "spline",
```

- ***Paso 5*:** Ahora solo queda cargar la información de la API dentro de la variable creada en el paso anterior del tipo arreglo y unir todo el código. Esto se puede lograr mediante una estructura repetitiva o ciclo for, lo cual, permitirá separar los datos exactos que necesita la gráfica, como son el valor del dólar y la fecha correspondiente a ese valor.

```
$(document).ready(function(){
      var dataPoints = [];

      var options =  {
         animationEnabled: true,
         theme: "light2",
         title: {
            text: "Daily Sales Data"
         },
         axisX: {
            valueFormatString: "DD MMM YYYY",
         },
         axisY: {
            title: "USD",
            titleFontSize: 24,
         },
         data: [{
            type: "spline",
            dataPoints: dataPoints
         }]
      };
      $.ajax({
         type:"GET",
         url:"https://mindicador.cl/api/dolar",
         dataType:"json",
         success: function(datos) {
            let datosApi = datos.serie;
            console.log(datosApi);
            for (var i = 0; i < datosApi.length; i++)
                  dataPoints.push({
                  x: new Date(datosApi[i].fecha),
                  y: datosApi[i].valor
                  });
            }
            $("#chartContainer").CanvasJSChart(options);
         },
         error: function(error) {
```

```
            console.log(error)
        }
    });
});
```

- ***Paso 6*:** Al ejecutar el código anterior, el resultado en nuestro navegador web será la gráfica con las tendencias del dólar a lo largo del tiempo:

Imagen 11. Resultado de la aplicación.
Fuente: Desafío Latam

Ejercicio propuesto (4)

Partiendo del ejercicio realizado anteriormente, muestra ahora la información del euro con respecto al peso chileno mediante un gráfico. La API tiene la opción de traer las tendencias del euro solamente.

Muuri

Te gustaría realizar componentes dinámicos que funcionan en base al concepto drag & drop (arrastrar y soltar) Muuri es la solución. Es un plugin muy completo creado a base de jQuery que permite generar contenedores y arrastrarlos hacia otro contenedor, manipulando desde la interfaz del usuario la disposición de los elementos.

Ejercicio guiado: Muuri

Crear dos bloques definidos, uno al lado del otro, con un texto en su interior, considerar que estos bloques se puedan arrastrar e intercambiar de posición entre ellos, partiendo del siguiente código:

```
<div class="grid">
   <div class="item">
      <div class="item-content">
        Esto puede ser cualquier texto.
      </div>
   </div>
   <div class="item">
      <div class="item-content">
        <div class="my-custom-content">
            Y aquí también!
        </div>
      </div>
   </div>
</div>
```

- ***Paso 1*:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js.

- ***Paso 2*:** En el index.html, escribir la estructura básica de un documento HTML, incorporar el enlace o CDN de jQuery y Muuri. Al igual que en el primer plugin que revisamos, es necesario incluirlo después que jQuery, de esta forma evitamos que la aplicación no funcione. Luego, agregar el extracto del código HTML entregado en el enunciado y enlaza el archivo externo denominado “script.js”:

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width,
initial-scale=1.0">
      <link rel="stylesheet" href="style.css">
      <title>Usando Muuri</title>
</head>
<body>
      <div class="grid">
```

```
         <div class="item">
            <div class="item-content">
               Esto puede ser cualquier texto.
            </div>
         </div>
         <div class="item">
            <div class="item-content">
               <div class="my-custom-content">
                  Y aquí también!
               </div>
            </div>
         </div>
      </div>
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"><
/script>
   <script
src="https://cdn.jsdelivr.net/npm/muuri@0.9.0/dist/muuri.min.js"></scrip
t>
   <script src="script.js"></script>
</body>
</html>
```

- ***Paso 3***: Crear el estilo necesario para poder contener los elementos en cuadros, agregando un color de fondo, ancho, alto, posición y como se mostrará cada uno de esos elementos.

```
.grid {
  position: relative;
}
.item {
  position: absolute;
  width: 100px;
  height: 100px;
  z-index: 1;
  background: #000;
  color: #fff;
}
.item.muuri-item-dragging {
  z-index: 3;
}
.item.muuri-item-releasing {
  z-index: 2;
```

```
}
```

- ***Paso 4:*** Queda entonces por trabajar con el archivo script.js, en donde lo primero a realizar es crear una variable que contenga un nuevo objeto de Muuri (no es un nombre al azar, la librería indica que debe ser creado así). En este objeto, se indica a cuál elemento del DOM le vamos a activar las características de Muuri, en este caso los que tengan clase `grid`, además se entrega la configuración o comportamiento que deben tener los elementos a los cuales le aplicamos la librería. Como el caso de `dragEnabled`, quien activa la opción de arrastrar y soltar, o `dragReleaseEasing`, quien indica cuál sería el tipo de efecto en el movimiento de los bloques. Las configuración del comportamiento de los elementos se encuentran mejor especificadas en la documentación de la librería, disponible en Muuri Doc.

```
var grid = new Muuri('.grid',
   {
      dragEnabled: true,
      dragReleaseEasing: 'ease',
   },
);
```

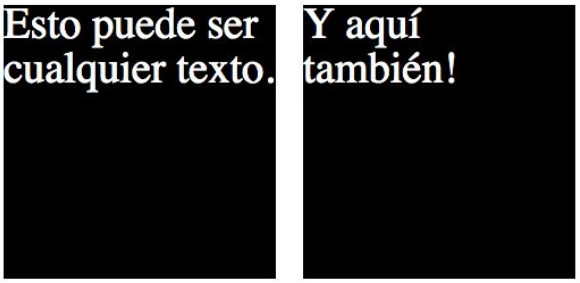

Imagen 12. Resultado de la aplicación.
Fuente: Desafío Latam

Ejercicio propuesto (5)

Construye un sitio que muestre los números del 1 al 10 y permita utilizar la función de arrastrar y soltar, que ofrece Muuri.

Ahora es tu turno de investigar algunos plugins interesantes, la gama de posibilidades es infinita, con jQuery puedes hacer proyectos pequeños, complejos y de cualquier tema. La base siempre está en cómo aplicar las herramientas. Te invitamos a ser proactivo y aprender con estas herramientas, por tus medios. Puedes partir por la documentación oficial de jQuery y consultar el sitio de jQuery Script, que tiene mucha información relevante.

Resumen

AJAX, permite al usuario poder acceder a información directa sin la necesidad que la página tenga que recargar o iniciar una nueva vista para poder mostrar la información solicitada por el usuario. Esto significa que el rendimiento del sitio web mejora en velocidad, debido a que las consultas se hacen de forma asíncrona y precisa.

Las APIs son un medio simplificado para conectar tu proyecto a través del desarrollo de aplicaciones nativas de la nube, pero también le permiten compartir sus datos con clientes y otros usuarios externos. Las APIs públicas representan un valor comercial único, porque simplifican y amplían la forma en que se conecta con sus partners y, además, pueden rentabilizar sus datos. Además, las APIs públicas son esenciales para el desarrollo de código abierto.

Existen muchos más recursos disponibles, pero esta base te permitirá potenciar tus proyectos e integrar recursos tanto externos como de servicios propios, utilizando estas herramientas para ofrecer aún mayores posibilidades a los visitantes.

Solución de los ejercicios propuestos

1. Utiliza Postman para realizar una conexión, traer y mostrar la información de la siguiente API: https://reqres.in/api/users.

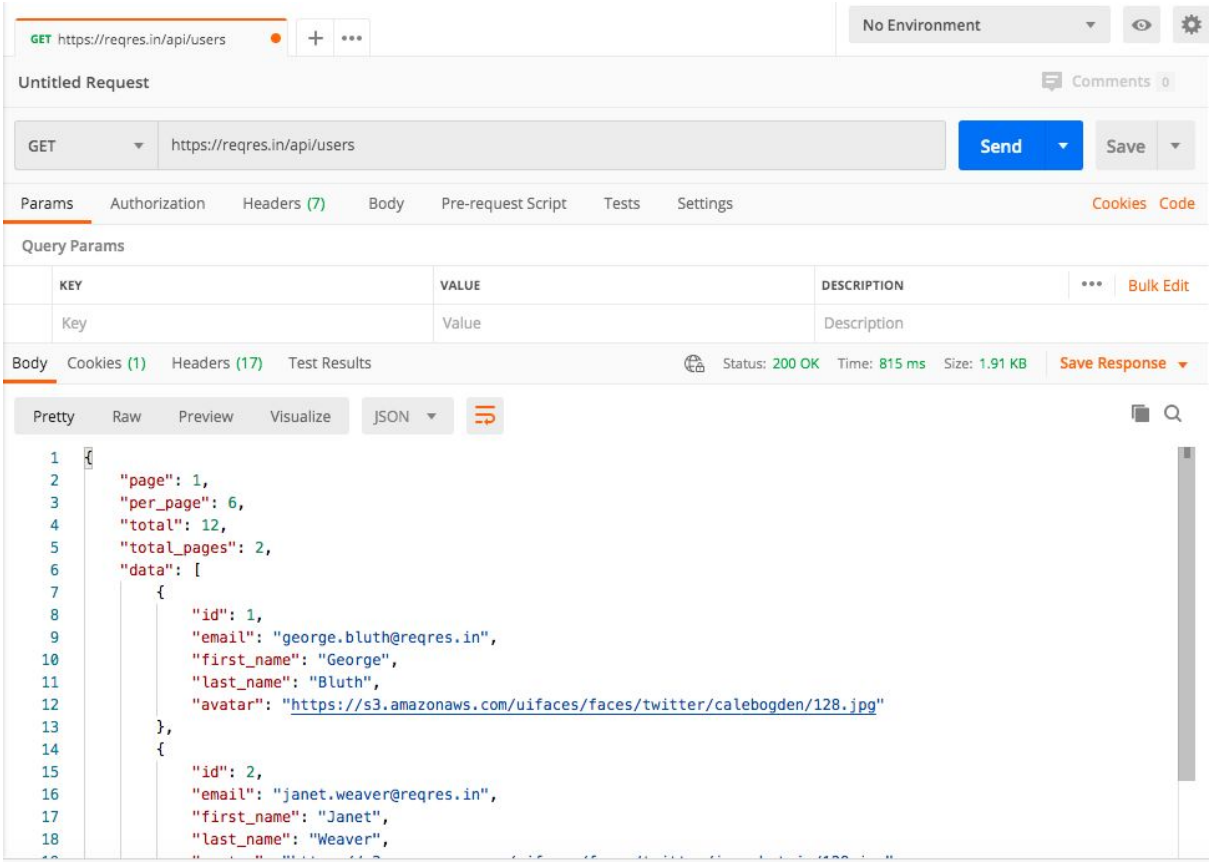

Imagen 13. Solución ejercicio propuesto 1
Fuente: Desafío Latam

2. Para la siguiente dirección web (perteneciente a una API): https://jsonplaceholder.typicode.com/users. Realiza la respectiva conexión mediante AJAX y luego muestra la información solo cuando un usuario haga clic sobre un botón.

```
$(document).ready(function(){
      $('button').on('click',function(){
      $.ajax({
            type:"GET",
            url:"https://jsonplaceholder.typicode.com/users",
            dataType:"json",
            success: function(datosApi) {
            console.log(datosApi);
            datosApi.forEach(element => {

$('.resultado').append(`<p>${element.id}-${element.name}-${element.usern
ame}-${element.email}-${element.phone}</p>`);
            })
            },
            error: function(error) {
            console.log(error);
            },
      });
      });
 });
```

3. Crea un plugin con jQuery para conectarse, traer y mostrar la información en el documento web para al siguiente api: https://jsonplaceholder.typicode.com/users. Realiza la respectiva conexión mediante AJAX y luego muestra la información solo cuando un usuario haga clic sobre un botón.

Archivo index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"><
/script>
   <title>Document</title>
</head>
<body>
   <div id="miCaja"></div>
   <script src="script.js"></script>
```

```
   <script src="jQuery.datosGenerales.js"></script>
</body>
</html>
```

Archivo jQuery.datosGenerales.js

```
jQuery.fn.datosGenerales = function() {
      var element = this;
      $.ajax({
      type:"GET",
      url:"https://jsonplaceholder.typicode.com/users",
      dataType:"json",
      success: function(data) {
            console.log(data)
            data.forEach(datos => {
                  element.append(`<p>ID: ${datos.id} - Name:
${datos.name} - UserName: ${datos.username}</p>`);
            });
      }
      });
   return this;
};
```

Archivo script.js

```
$(document).ready(function(){
      $('#miCaja').datosGenerales();
});
```

4. Partiendo del ejemplo realizado anteriormente, muestra ahora la información del euro con respecto al peso chileno mediante un gráfico. La API tiene la opción de traer las tendencias del euro solamente.

```
$(document).ready(function(){
      var dataPoints = [];

      var options =  {
      animationEnabled: true,
      theme: "light2",
      title: {
            text: "Tendencias del Euro"
      },
      axisX: {
            valueFormatString: "DD MMM YYYY",
      },
      axisY: {
            title: "EUR",
            titleFontSize: 24,
            includeZero: false
      },
      data: [{
            type: "spline",
            yValueFormatString: "€#,###.##",
            dataPoints: dataPoints
      }]
      };

      $.ajax({
      type:"GET",
      url:"https://mindicador.cl/api/euro",
      dataType:"json",
      success: function(datos) {
            let datosApi = datos.serie;
            console.log(datosApi);
            for (var i = 0; i < datosApi.length; i++) {
                  dataPoints.push({
                  x: new Date(datosApi[i].fecha),
                  y: datosApi[i].valor
                  });
            }
            $("#chartContainer").CanvasJSChart(options);
      },
      error: function(error) {
            console.log(error)
      }
      });
```

```
});
```

5. Construye un sitio que muestre los números del 1 al 10 y permita utilizar la función de arrastrar y soltar, que ofrece Muuri.

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width,
initial-scale=1.0">
      <link rel="stylesheet" href="style.css">
      <title>Usando Muuri</title>
</head>
<body>
<div class="grid grid-1">
  <div class="item">
    <div class="item-content">1</div>
  </div>
  <div class="item">
    <div class="item-content">2</div>
  </div>
  <div class="item">
    <div class="item-content">3</div>
  </div>
  <div class="item">
    <div class="item-content">4</div>
  </div>
  <div class="item">
    <div class="item-content">5</div>
  </div>
  <div class="item">
    <div class="item-content">6</div>
  </div>
  <div class="item">
    <div class="item-content">7</div>
  </div>
  <div class="item">
    <div class="item-content">8</div>
  </div>
  <div class="item">
```

```
    <div class="item-content">9</div>
  </div>
  <div class="item">
    <div class="item-content">10</div>
  </div>
</div>
   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"><
/script>
   <script
src="https://cdn.jsdelivr.net/npm/muuri@0.9.0/dist/muuri.min.js"></scrip
t>
   <script src="script.js"></script>
</body>
</html>
```

CSS

```
body {
  padding: 0;
}
.grid {
  position: relative;
}
.item {
  position: absolute;
  width: 200px;
  height: 200px;
  line-height: 200px;
  margin: 5px;
  z-index: 1;
}
.item.muuri-item-hidden {
  z-index: 0;
}
.item.muuri-item-releasing {
  z-index: 2;
}
.item.muuri-item-dragging {
  z-index: 3;
}
.item-content {
  position: relative;
  font-family: sans-serif;
  width: 100%;
  height: 100%;
```

```
  text-align: center;
  border: 1px solid #F44336;
  border-radius: 3px;
  font-size: 25px;
  color: #F44336;
  cursor: pointer;
}
.item.muuri-item-dragging .item-content,
.item.muuri-item-releasing .item-content {
  background: #FFCDD2;
}
```

JS

```
var grid1 = new Muuri('.grid-1', {
  dragEnabled: true,
  dragContainer: document.body,
  dragSort: function () {
    return [grid1]
  }
});
```