MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO WEB EN ENTORNO CLIENTE
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Dinamismo en el desarrollo de páginas web

11

ÍNDICE

OBJETIVOS

Comprender las limitaciones del lenguaje.

Saber identificar las ventajas de la librería.

Saber usar selectores.

Saber realizar comunicaciones asíncronas.

Saber realizar conversiones entre tipo de datos.

/ 1. Introducción y contextualización práctica

JavaScript es un lenguaje **muy potente** para poder **modificar** y **crear documentos web** de manera **dinámica**. Sin embargo, con el desarrollo de nuevas tecnologías se ha convertido en un lenguaje accesorio, por lo que otras tecnologías utilizan JavaScript como base. En la actualidad, existen herramientas que permiten crear código final en JavaScript a partir de un lenguaje de alto nivel. Además, existen librerías que están basadas en JavaScript, pues permiten ampliar el funcionamiento del lenguaje con nuevos añadidos.

Una de las **limitaciones** más grandes que tiene este lenguaje es **su sintaxis,** dado que, en ocasiones, hay que escribir gran cantidad de código para realizar pequeñas tareas. En un contexto en el que los nuevos lenguajes de programación se centran en escribir cada vez menos código, tiene sentido seguir el mismo enfoque con JavaScript. A lo largo de este tema presentaremos jQuery con el fin de facilitar el desarrollo de páginas web que utilicen JavaScript.

Fig. 1. Una web dinámica mejora la experiencia de usuario.

Escucha el siguiente audio que describe el caso práctico que resolveremos a lo largo de la unidad.

Audio intro. "Desarrollo web más fluido"

https://bit.ly/306RmoC

/ 2. Introducción a jQuery

JQuery es una **librería JavaScript** cuyo objetivo es permitir la utilización de dicho lenguaje en un sitio web de una manera más sencilla. De hecho, la filosofía de jQuery se asemeja mucho a los lenguajes actuales de programación: *Write less*, do more («escribe menos, haz más»). Para poder utilizar correctamente jQuery, es necesario conocer HTML, CSS y JavaScript.

JQuery es capaz de simplificar gran cantidad de tareas que requieren muchas líneas en JavaScript, logrando un código más compacto. Además, ofrece muchas funcionalidades por defecto. Otra de las ventajas que logramos con jQuery es simplificar las comunicaciones con AJAX (que estudiaremos en profundidad más adelante) y la manipulación del DOM. A través de jQuery, podemos realizar diferentes funcionalidades como manipulación del DOM y CSS, gestión de eventos HTML, acceso a librería de utilidades, efectos y animaciones, etc.

Audio 1. "JQuery y librerías externas"
https://bit.ly/32eecNw

Desde el punto de vista de la compatibilidad con los navegadores, jQuery es compatible con todos ellos. Actualmente, muchas de las nuevas librerías que se crean son capaces de trabajar directamente con jQuery en vez de con JavaScript. Para poder añadir jQuery a un sitio web, existen dos opciones:

- **Descargar** la librería desde la página oficial e incluir los scripts correspondientes.
- **Incluir** jQuery desde un **CDN** como, por ejemplo, Google.

```
<!--Incluyendo jQuery desde la libreria-->
<head>
<script src="jquery-3.4.1.min.js"></script>
</head>
<!--Incluyendo jQuery desde un CDN-->
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/
jquery.min.js"></script>
</head>
```

Código 1. Incluyendo jQuery desde la librería y desde un CDN.

/ 3. Sintaxis básica

La sintaxis de jQuery utiliza selectores tipo CSS para poder realizar tareas sobre el DOM-HTML y realizar acciones en JavaScript. La sintaxis básica se muestra a continuación:

```
$(selector).action()
```

Código 2. Sintaxis básica jQuery.

Analizando el código anterior:

- **El símbolo $** define o accede a jQuery.
- **A través del selector** le indicamos a jQuery los elementos HTML que se necesitan.
- **A través de la acción** ejecutamos una tarea sobre los elementos seleccionados.

3.1. Documento preparado

Para poder comenzar a **utilizar jQuery** es necesario que el **documento HTML** esté totalmente **cargado** y el **DOM preparado**. JQuery proporciona métodos para saber cuándo la librería está cargada y el documento listo para poder utilizarse a través de la API de la librería.

Este mecanismo evita que se pueda utilizar jQuery antes de que el documento haya finalizado la carga en su totalidad, es decir, antes de que el documento esté listo. Por lo tanto, es una buena práctica esperar a que el documento esté **completamente cargado** y **preparado** antes de comenzar a trabajar con él. Supongamos que realizamos el manejo del documento a través de jQuery sin haber esperado a la carga total del documento. En este caso, algunas de las funcionalidades básicas de jQuery posiblemente no funcionen correctamente. Por ejemplo, si intentamos ocultar un objeto, puede fallar; si queremos obtener el tamaño de una imagen, puede devolver tamaños incoherentes, etc.

```
$(document).ready(function(){
 //Cuando se ejecuta este código, el documento está preparado
});
```

Código 3. Documento preparado.

Vídeo 1. "Documento no preparado"
https://bit.ly/32cd7Gd

3.2. Selectores

Los selectores son los **elementos básicos** para poder utilizar **jQuery** correctamente. Se trata de la parte más importante de esta librería. Un selector en jQuery permite que el usuario pueda seleccionar y manipular elementos HTML con facilidad. Los selectores jQuery permiten buscar dentro del documento HTML elementos a través de su nombre, identificador, clase, tipos, atributos, etc.

Se dice que los **selectores** en **jQuery** están fundamentados en los selectores **CSS**. La notación básica de cualquier sentencia en jQuery sigue la forma: **$()**. Gracias a los selectores, jQuery puede simplificar en gran medida el código JavaScript empleado, pues jQuery nos facilita esta tarea. Por ejemplo, supongamos que queremos ocultar todos los contenedores **(div)** de un documento HTML:

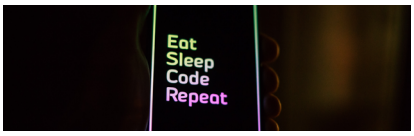

Fig. 2. JQuery permite codificar de una forma más sencilla.

```
$(document).ready(function(){
  $("div").hide();
});
```

Código 4. Ocultar contenedores div.

Como podemos observar en el ejemplo anterior, una vez el **documento ha finalizado su carga**, se utiliza un selector para **encontrar** todos los **contenedores** y llamar a la función **hide()** proporcionada por el API de jQuery para poder ocultar los elementos devueltos por el selector. Si quisiéramos realizar esto a través de **JavaScript**, sería necesario buscar todos los elementos, guardarlos en una lista y utilizar un bucle para cambiar su propiedad de visibilidad. Con la utilización de jQuery, todo esto queda en prácticamente una línea. JQuery también permite utilizar selectores que reduzcan el número de elementos que se pueden obtener. Es posible seleccionar elementos utilizando el atributo de identificador incluido en el documento HTML para una determinada etiqueta.

3.3. Selector con ID y CLASS

Supongamos que en el ejemplo del Código 3 **no queremos ocultar todos los contenedores**, sino únicamente el contenedor id="test":

```
$(document).ready(function(){
  $("#test ").hide();
});
```

Código 5. Ocultar contenedor 'test'.

Como podemos observar, utilizando **'#' podemos indicar** que el elemento que situamos a continuación **hace referencia** a un identificador **HTML**. En otras ocasiones, nos puede interesar seleccionar todos aquellos elementos que pertenezcan a una determinada clase. Para ello, se puede utilizar en jQuery la siguiente sintaxis:

```
$(document).ready(function(){
  $(".test ").hide();
});
```

Código 6. Ocultar elementos de una misma clase.

La granularidad de **los selectores** puede ser modificada utilizando el **API de jQuery.** Por ejemplo, nos ofrece mecanismos para seleccionar el elemento actual a través del puntero this.

También es posible referirnos a determinados atributos dentro de una etiqueta HTML. Asimismo, nos puede ofrecer la posibilidad de coger todos aquellos elementos de un determinado tipo que se encuentran dentro de un formulario, entre otras.

En ocasiones, cuando el número de selectores aumenta significativamente, puede ser necesario tener el código de cada selector en determinadas funciones para poder facilitar la reutilización y mejorar la modularidad del código. Como vemos, son los mismos principios que seguimos cuando desarrollamos un código en JavaScript. **JQuery** nos ofrece la posibilidad de **establecer** estas **funciones JavaScript** en archivos separados, de tal forma que podamos utilizarlas fuera de jQuery y en otros contextos.

/ 4. Caso práctico 1: "Carga del documento"

Planteamiento: Tras analizar un código de una web creada con jQuery, vemos que se realiza la detección de la carga de la siguiente forma:

```
<body onload="initWeb()">
</body>;
```

Código 7. ¿Error en la carga?

Nudo: ¿Crees que se adapta a la sintaxis de jQuery? ¿Puede generar algún error?

Fig. 3. Para el correcto funcionamiento de la web todas las librerías deben estar cargadas.

Desenlace: Como hemos visto en el apartado del operador sobre la carga de un documento jQuery, para garantizar que la ejecución se realice correctamente, hay que esperar al evento de carga de la librería. En el código expuesto, se detecta la carga del cuerpo de la web, sin esperar a la carga total del documento web (lo que incluye las librerías). Si la función initWeb() utiliza funciones de las librerías JavaScript incluidas, el comportamiento no será el esperado. En el siguiente ejemplo se muestra una posible solución:

```
<body>
....
</body>

<script>

$( document ).ready(function() {
   initWeb();
});
</script>
```

Código 8. Detección de la carga del documento.

Con el enfoque propuesto, se puede ver que esperamos a que JQuery detecte que la carga del documento web ha finalizado utilizando la función ready.

/ 5. Eventos

JQuery también permite **asignar** y **responder** a **eventos** de los **documentos HTML**. Como sabemos, los diferentes usuarios de la página pueden generar diferentes tipos de eventos, que serán lanzados por HTML e implementados en JavaScript. En este contexto, **jQuery** nos permite **identificar** y **modificar** el **comportamiento** de **los eventos** cuando se lanzan. JQuery nos permite responder a eventos que se pueden generar del ratón, teclado, formulario y ventana. Es posible que exista algún evento que no esté contemplado en el API de jQuery y que, por ello, sea necesario utilizar directamente JavaScript. Recordemos que jQuery no implementa todo el API de JavaScript, sino que se centra en aquellas funcionalidades más utilizadas. A través de jQuery es posible añadir eventos utilizando selectores. También es posible implementar las acciones a realizar cuando ocurre un determinado evento. Veamos el siguiente ejemplo:

```
//Añadimos el evento click a todos los botones
$("button").click();
//Cuando se detecta un evento click, ejecutamos la acción
$("button").click(function(){
  console.log("Hola!!");
});
```

Código 9. Eventos en jQuery.

Como se puede observar, el manejo de eventos en **jQuery** se divide en **dos partes:** asignación de eventos y definición de las acciones. En la siguiente tabla se muestran los eventos más utilizados en jQuery.

EVENTO	DESCRIPCIÓN
ready()	Permite saber cuándo se ha cargado el documento
click()	Se centra en determinar cuándo se realiza un clic
dblclick()	Permite controlar el doble clic
mouseenter()	Se centra en controlar cuándo entra el ratón
mousedown()	Detecta cuando se mantiene el botón pulsado

Tabla 1. Eventos más utilizados en jQuery.

5.1. Eventos con el método 'on'

Este método permite establecer **diferentes manejadores de eventos** para los elementos seleccionados a través del selector utilizado, indicando para ello el evento asociado con un literal. Como se puede intuir, hay que tener bastante cuidado al especificar el literal, pues un fallo en su tecleo puede generar un comportamiento indeseado o, incluso, no ejecutar nada. En ocasiones, este tipo de errores suelen ser bastante tediosos de depurar pues, como ya sabemos en JavaScript, y por extensión en jQuery, los errores se producen en ejecución y la información que nos ofrece el depurador es bastante limitada.

```
$("p").on({
 mouseenter: function(){
  $(this).css("background-color", "lightgray");
 },
 mouseleave: function(){
  $(this).css("background-color", "lightblue");
 },
 click: function(){
  $(this).css("background-color", "yellow");
 }
});
```

Código 10. Eventos con el método 'on'.

En el ejemplo anterior, se muestra un código en el que se utiliza **un selector jQuery para obtener todos los párrafos** del documento. A cada uno de ellos se le asignan diferentes eventos utilizando el método on. De esta forma, se incluyen diferentes manejadores para diferentes tipos de eventos que cambian el color de fondo, utilizando diferentes propiedades CSS.

Fig. 4. El método 'on' permite añadir varios manejadores de eventos.

/ 6. Efectos y comunicaciones con jQuery

El API de **jQuery** permite realizar **efectos de una forma más sencilla** que utilizando directamente JavaScript y CSS. En ocasiones, muchos de estos efectos están más relacionados con CSS que con JavaScript. Como veremos, existen efectos para animar, ocultar o incluso para realizar transiciones y desplazamientos.

6.1. Ocultar/mostrar

JQuery ofrece d**os métodos para poder ocultar y mostrar** determinados componentes a través de los selectores *hide()* y *show()*. A través del mismo API, es posible establecer la velocidad de esta animación para mejorar la experiencia de usuario. A veces, puede ser necesario detectar cuándo finaliza el cambio de estado de un elemento y, para ello, se incorpora una función *callback* a ambas funciones. Las *callbacks* pueden utilizarse para llevar a cabo la tarea que estimemos oportuna en nuestro código cuando finaliza la animación.

```
$("button").click(function(){
 $("p").hide(500);
});
```

Código 11. Efecto ocultar.

En el ejemplo anterior se utiliza un **selector** para **tomar todos los botones de nuestra página** y **asignar** un evento **'clic'**. La función asociada con la detección de este evento permite ocultar todos los párrafos del documento HTML con un tiempo de animación de 500ms. Es muy común que en un código en el que utilicemos las funciones *hide()/show()* necesitemos alternar entre ellas según estemos en un estado u otro. JQuery ofrece la función *toggle()*, que permite visualizar un componente si estaba oculto previamente o viceversa. Veamos un ejemplo de su funcionamiento:

```
$("button").click(function(){
 $("p").toggle(500);
})
```

Código 12. Función 'toggle'.

Vídeo 2. "Efectos con jQuery"
https://bit.ly/2Ol41yv

6.2. Deslizamiento

JQuery permite crear efectos de desplazamiento sobre los elementos. Es posible especificar el tiempo de animación y el código a ejecutar cuando la animación finaliza a través de una callback. Existen diferentes tipos de animaciones para el desplazamiento: hacia arriba, hacia abajo y hacia arriba o hacia abajo.

```
$("#container").click(function(){
 $("#menu").slideDown();
 $("#menu").slideUp();
 $("#menu").slideToggle();
});
```

Código 13. Desplazamiento.

En el ejemplo anterior, se muestra el funcionamiento de las tres llamadas de desplazamiento para mostrar u ocultar un menú. La función *slideToggle()* es capaz de guardar el estado de visualización de un elemento, permitiendo visualizarlo u ocultarlo según sea necesario.

6.3. Comunicaciones con jQuery

JQuery proporciona **muchos métodos para trabajar con AJAX** (como veremos más adelante) de una manera sencilla. Con jQuery podemos realizar peticiones en formato texto, HTML, XML o en JSON. Es posible conectar a un servidor utilizando el protocolo HTTP y los métodos de conexión GET y POST. La ventaja de utilizar AJAX con jQuery es que nos podemos olvidar de las diferencias y particularidades que presentan cada uno de los navegadores.

```
$("button").click(function(){
 $.get("test.php", function(data, status){
  console.log("Data: " + data + "\nStatus: " + status);
 });
});
```

Código 14. Comunicaciones con jQuery.

En el ejemplo anterior, se muestra un **ejemplo de petición GET** utilizando AJAX. Al ejecutar la *callback*, se muestran los valores por la consola del navegador. Esta sintaxis es mucho más sencilla que la original de AJAX.

/ 7. Caso práctico 2: "Efectos visuales"

Planteamiento: Nuestro cliente desea crear una web donde la interacción y los efectos visuales atraigan al usuario. Por ello, nos encarga el diseño de esta web, que tiene que ser visualizada de manera semejante en todos los navegadores.

Nudo: ¿Cómo podrías realizar esto? ¿Podrías añadir algún efecto?

Desenlace: Como hemos visto en el apartado sobre los efectos en jQuery, es posible utilizar la librería para introducir efectos sobre los menús y sobre determinadas partes de la web para mostrar/ocultar elementos, incluir transiciones, etc. Además, gracias a jQuery, podemos introducir todo esto de una manera relativamente sencilla. Desde el punto de vista de la interacción, podemos utilizar la función on para incluir el número de manejadores que necesitemos y simplificar la tarea de gestión de eventos. Con todo esto, cumpliríamos con los requisitos del cliente, pues jQuery ofrece compatibilidad con todos los navegadores. En el siguiente ejemplo, se muestra cómo podemos realizar esto mediante el método on de JQuery.

Fig. 5. El método 'on' permite añadir mayor cantidad de eventos de manera sencilla.

```
$( "#table1 tbody tr" ).on( "click", function() {
 console.log( $( this ).text() );
});
$( "#container tbody" ).on( "click", "tr", function() {
 console.log( $( this ).text() );
});
```

Código 15. Efectos visuales con el método 'on'.

Utilizando este método, se pueden añadir tantos eventos como sean necesarios. De esta forma, es posible mejorar la experiencia de usuario tanto a nivel visual como de interacción.

/ 8. Resumen y resolución del caso práctico de la unidad

A largo de este tema **hemos presentado la librería JQuery** con el fin de facilitar el desarrollo en el lado del cliente. Con esta librería, se simplifican muchas de las tareas que ofrece de forma nativa JavaScript como, por ejemplo, la selección de elementos, la gestión de eventos y su definición.

Hemos podido observar que el manejo del documento se realiza de una forma más ágil y obviando las particularidades de cada uno de los navegadores. JQuery es una de las librerías principales sobre la que se construye gran parte de las *frameworks* existentes en la actualidad para el desarrollo en el lado del cliente.

Además, con jQuery **hemos visto que podemos añadir efectos sobre los estilos y sobre los elementos** en un documento HTML como, por ejemplo, ocultar o visualizar elementos. **Desde el punto de vista de las comunicaciones, hemos visto que al utilizar jQuery se simplifica significativamente la cantidad de código** que hay que ejecutar para una petición AJAX.

Resolución del caso práctico de la unidad

En esta situación, el cliente nos pide que utilicemos un lenguaje de *scripting* para todo, sin tener en cuenta que cada tecnología se utiliza para cosas diferentes. HTML debe utilizarse para el maquetado, CSS para la apariencia visual y JavaScript para comunicaciones e interacción con el usuario. Por lo tanto, lo que nos comenta el cliente deberá aclararse antes de comenzar el desarrollo. Además, cuando nos pide que utilicemos solo JavaScript «porque el desarrollo es más rápido» también se equivoca. Podríamos recomendarle utilizar una librería de terceros como jQuery, ya que nos ofrece la posibilidad de manejar el DOM HTML y CSS de una manera más fácil, garantizando que la web estará online en la fecha establecida.

Fig. 6. Para aprovechar todo el potencial de JQuery es necesario conocer su API.

/ 9. Bibliografía

Aubry, C. y Lancker, L. (2017). *jQuery: el framework JavaScript para sitios dinámicos e interactivos.* Barcelona: Eni.

Duckett, J., Ruppert, G. & Moore, J. (2014). *JavaScript & jQuery: interactive front-end web development*. Indianapolis: Wiley.