Desarrollo web en entorno cliente

Unidad 6: Utilización del modelo de objetos del documento (DOM)

ÍNDICE

INTRODUCCIÓN

Durante todo el curso hemos mencionado en varias ocasiones el **modelo de objetos** del documento o **DOM**, pero no hemos visto realmente lo que es y cuáles son sus usos.

Durante esta unidad veremos su **definición** y principales **características**. Estudiaremos que son los elementos u objetos, y veremos sus **métodos** y **propiedades**.

Haremos hincapié en las diferencias que puede presentar el **DOM** en los distintos **navegadores** del mercado, y como podemos hacer para intentar minimizarlas.

Posteriormente nos meteremos de lleno en la **programación**, viendo cómo podemos acceder a los distintos elementos del DOM desde **código**, mediante el **lenguaje JavaScript**.

Finalmente veremos dos conceptos nuevos, como son las **aplicaciones multi-capa** y las **aplicaciones multi-cliente**, que hacen uso de todo lo que iremos viendo.

En esta unidad, una vez más, se requerirán conocimientos de otros módulos del ciclo, especialmente relacionados con la **programación tradicional** y la **programación web en HTML**.

OBJETIVOS / CAPACIDADES

En esta unidad de aprendizaje, las capacidades que más se van a trabajar son:

- ✔ **Analizar y utilizar los recursos y oportunidades de aprendizaje relacionadas con la evolución científica, tecnológica y organizativa del sector y las tecnologías de la información y la comunicación, para mantener el espíritu de actualización y adaptarse a nuevas situaciones laborales y personales.**

PROYECTO DE LA UNIDAD

Antes de empezar a trabajar el contenido, te presentamos la **actividad** que está relacionada con esta unidad de aprendizaje. Se trata de un **caso práctico** basado en una **situación real** con la que te puedes encontrar en tu puesto de trabajo. Con esta actividad se evaluará la puesta en práctica de los **criterios de evaluación** vinculados al resultado de aprendizaje que se trabaja en esta unidad. Para realizarla deberás hacer lo siguiente: lee el enunciado que te presentamos a continuación, dirígete al área general del módulo profesional, concretamente a la actividad de evaluación que se encuentra dentro de esta unidad, allí encontrarás todos los detalles sobre fecha y forma de entrega, objetivos... A lo largo de la unidad irás adquiriendo los conocimientos necesarios para ir elaborando este proyecto.

Enunciado:

Iluminando la Navidad

La empresa está contenta con las pequeñas pruebas que hemos hecho, así que ya nos podemos meter de lleno en el proyecto del árbol de Navidad virtual.

Nos han proporcionado una plantilla HTML (con estilos incluidos), a la que habremos de dotar de funcionalidad. El único requisito es no destrozar el trabajo de los diseñadores, y por tanto no tocar nada del HTML.

Se pide:

- Al pasar el ratón por encima de las bombillas o la estrella, si están encendidas hay que apagarlas y viceversa.
- Al pulsar la letra 'Q' se apagará o encenderá el interruptor virtual, o sea, cortaremos o encenderemos el suministro eléctrico del árbol.
- Utilizar el Modelo-Vista-Controlador para separar la aplicación en capas.
- La página se tiene que ver de igual manera en al menos 3 navegadores y sistemas operativos diferentes.

Árbol sin suministro eléctrico:

Árbol con luces y estrella encendidas:

1. EL MODELO DE OBJETOS DEL DOCUMENTO (DOM). COMPATIBILIDAD ENTRE NAVEGADORES

En varias de las unidades hemos ido hablando del **DOM**, y su uso con **JavaScript** en la creación de a**plicaciones web dinámicas** en entorno servidor. Pero, ¿qué es el DOM? Hablaremos en profundidad sobre ello durante esta unidad.

DESTACADO

El DOM (Document Object Model) es un estándar definido por la W3C (World Wide Web Consortium), que establece unas reglas o pautas para acceder y manipular documentos XML, y por extensión, también al HTML.

A continuación veremos algunos **aspectos relevantes sobre el DOM**:

- El estándar del W3C diferencia el DOM en tres **categorías**:
  - **Core DOM**: modelo estándar para todo tipo de documentos.
  - **XML DOM**: modelo estándar para documentos XML.
  - **HTML DOM**: modelo estándar para documentos HTML.
- **Función**: las pautas del DOM lo que hacen es dar **estructura** un documento XML en una estructura en **forma de árbol,** o lo que es lo mismo, en una jerarquía de nodos. Este árbol representa tanto el **contenido** de documento como la **relación** entre los **nodos** que lo conforman.

TOMA NOTA

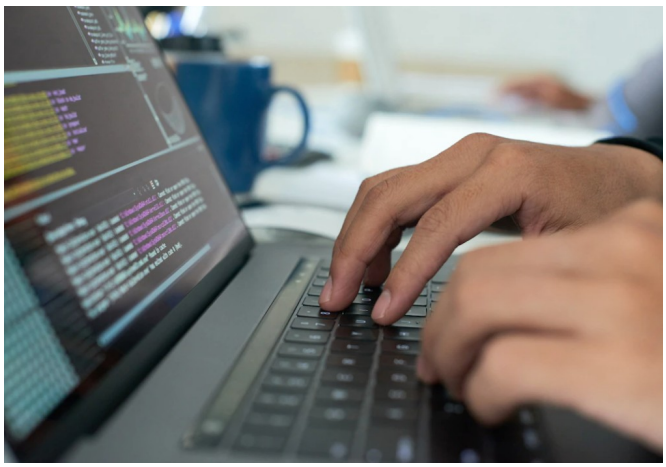
En esta unidad, estudiaremos el HTML DOM, que es el estándar para manipular y programas interfaces para HTML.

- **Propósito**: el propósito del HTML DOM es definir los elementos HTML como **objetos**, las **propiedades** de los **elementos HTML**, los métodos para acceder a estos elementos y eventos de éstos.

En otras palabras, el HTML DOM es el **estándar** para saber cómo acceder, modificar, añadir o borrar elementos HTML. Una vez conozcamos este estándar, podremos realizar todas estas manipulaciones con el lenguaje **JavaScript**.

2. OBJETOS DEL MODELO. PROPIEDADES Y MÉTODOS DE LOS OBJETOS

Según el estándar **W3 HTML DOM**, todo en un documento HTML es un nodo, u objeto.



- El **documento** entero es un nodo documento.
- Cada **elemento HTML** es un nodo elemento.
- El texto de cada **etiqueta HTML** es un nodo texto.
- Cada **atributo** de cada etiqueta es un nodo atributo.
- Todos los **comentarios** son nodos comentarios.

Cada nodo puede ser **creado**, **modificado** y **borrado** mediante el uso de **JavaScript**.

➔ **Jerarquía**: los nodos dentro de la **estructura del árbol** de nodos tienen una relación jerárquica entre ellos. Los conceptos **parent** (padre/madre), **child** (hijo/a) o **sibling** (hermano/a) son los que se utilizan para referirse a estas relaciones.

- ✔ En la jerarquía del árbol de **nodos**, el nodo que ocupa la parte superior se denomina root o raíz.
- ✔ Cada nodo tiene un **padre**, excepto el **nodo root**.
- ✔ Un nodo puede tener cero o más **hijos**.
- ✔ Los nodos **siblings** son nodos que tienen el mismo **parent**.

➔ **Propiedades**: para que todos estos nodos puedan ser **accesibles** (ya hemos comentado que lo haremos mediante JavaScript), hace falta una **interfaz** que nos suministre propiedades y métodos.

Las propiedades nos darán información del tipo:

- ✔ ¿Quién es tu padre?
- ✔ ¿Quién es tu hermano?
- ✔ ¿Quién es tu hijo?

➔ **Métodos**: los métodos nos servirán para cosas como:

- ✔ Insertar nodos.
- ✔ Borrar nodos.
- ✔ Cambiar atributos y valores.

Para saber todas las propiedades y métodos hemos de dirigirnos a la documentación disponible, que podrás encontrar en los recursos para ampliar.

Vídeo: document Object Model

Visualiza el siguiente vídeo sobre el Document Object Model (DOM):
https://www.youtube.com/embed/jgU3Wn0Txec

3. REPRESENTACIÓN DE LA PÁGINA WEB COMO UNA ESTRUCTURA EN ÁRBOL

Ya sabemos que un documento **HTML** se estructura en **etiquetas**. Éstas son precisamente las que conforman la estructura del DOM.

Según el DOM cada etiqueta de una página HTML es un objeto, que pueden tener a su vez hermanos o hijos, como hemos visto en el capítulo anterior.

Todos estos objetos son accesibles empleando **JavaScript**, lo cual podemos usar para modificar la página. Entraremos en más detalles acerca de esto en los siguientes capítulos.

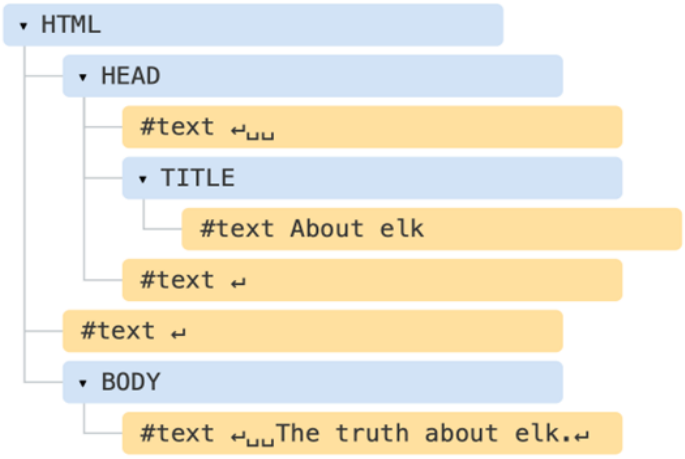
Lo que veremos ahora es como se representa una página **web** como una **estructura de árbo**l.

Imaginemos el siguiente documento HTML:

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>About elk</title>
    </head>
    <body>
        The truth about elk.
    </body>
</html>
```

Esto quedaría representado en el DOM de la siguiente manera, en la que cada elemento de la página web se convierte en un nodo del **DOM**.

*Especialmente, nos encontramos con los **nodos principales**, que son las **etiquetas**, y que son los que se consideran propiamente como los objetos del **DOM**.*



*Cada nodo del árbol es un **objeto**, con sus **propiedades** y **métodos**, como hemos visto anteriormente.*

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Qué representan las etiquetas en un documento HTML?

a. Objetos del DOM.

b. Texto visible en la página.

c. Atributos de estilo.

Lee el enunciado e indica la opción correcta:

¿Qué representa cada nodo en la estructura de árbol del DOM?

a. Etiquetas HTML.

b. Objetos del DOM.

c. Atributos de estilo.

Lee el enunciado e indica la opción correcta:

¿Qué podemos hacer con JavaScript en relación al DOM?

a. Modificar la página web.

b. Crear etiquetas HTML.

c. Cambiar el lenguaje de programación.

4. ACCESO AL DOCUMENTO DESDE CÓDIGO

Ahora que ya conocemos un poco mejor lo que es el DOM y de que está compuesto, vamos a ver cómo podemos utilizarlo con el lenguaje **JavaScript**.

Dado el siguiente ejemplo:

```
<html>
<head>
    <title>DOM Tutorial</title>
</head>
<body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
</body>
</html>
```

Utilizaremos los métodos y propiedades para averiguar la **relación entre los distintos nodos**.

Código: averiguar relación entre nodos

```
<script>
        document.write("El body tiene " + document.body.children.length +
" hijos<br/>");
        document.write("El primer hijo del body es " +
document.body.firstElementChild.nodeName + ", es de tipo " +
        document.body.firstElementChild.nodeType + " y el contenido es: "
+ document.body.firstElementChild.innerText + "<br/>");
        document.write("El segundo hijo del body es " +
document.body.firstElementChild.nextElementSibling.nodeName + "<br/>");
        document.write("El padre del title es " +
document.getElementsByTagName("title")[0].parentNode.nodeName + "<br/>");
        document.write("El primer elemento del documento es " +
document.firstElementChild.nodeName + " y este elemento tiene 2 hijos: "
+
        document.firstElementChild.firstElementChild.nodeName + " y " +
document.firstElementChild.lastElementChild.nodeName);
    </script>
```

Hemos de fijarnos en cómo hemos accedido a las propiedades de los elementos que nos interesan.

- **Acceder al nieto**: fijémonos también en que, al tratarse de una jerarquía de árbol, si queremos acceder al nieto de un elemento hemos de referirnos a **elemento.primerHijo.primerHijo**, y así sucesivamente hasta encontrar el elemento deseado.
- **Inspectores de propiedades**: trabajar con el DOM no es complicado, pero a veces puede ser un poco confuso saber quién es padre de quien, o qué atributos tiene un elemento. Para ello, los navegadores modernos, como Chrome, incorporan i**nspectores de propiedades**.
- **Objeto de tipo HTML**: cuando preguntamos al DOM, por ejemplo, los hijos de un elemento `<body>`, lo que hace es retornarnos un objeto de tipo HTML **CollectionObject**. Este objeto tiene una propiedad **length**, que ya conocemos, y un par de métodos: **item()** y **namedItem()**. En el siguiente ejemplo vemos como funcionan.

Ejemplo: objeto CollectionObject

En el ejemplo podemos ver como utilizamos los distintos métodos y propiedades para acceder a todos los elementos de la página, y por lo tanto, del DOM.

```
<script>
    var x, i, l;
    x = document.body.children;
    l = x.length;
    for (i = 0; i < l; i++) {
        document.write(x.item(i) + "<br>");
        document.write(x.item(i).tagName + "<br>");
        document.write(x.item(i).innerHTML + "<br>");
        document.write("<hr>");
    }
</script>
```

5. CREACIÓN Y MODIFICACIÓN DE ELEMENTOS

Además de leer las propiedades de los elementos y los nodos, los métodos que nos proporciona la interfaz Node, nos permiten crear y eliminar elementos.

Veamos otro **ejemplo** donde intentaremos añadir **párrafos <p>** con un nodo de texto y un atributo de color, dentro de un **contenedor **. Después tendremos la opción de borrar el último párrafo que hayamos añadido.

Código: creación y modificación de elementos

```
<html>


<body>
    <button id="add">Añadir</button>
    <button id="del">Eliminar</button>
    <div id="container"></div>
    <script>
        document.querySelector("#add").addEventListener("click",
addNewNode, false);
        document.querySelector("#del").addEventListener("click",
deleteLastNode, false);
        function addNewNode() {
            var elem = document.createElement("p");
            var txt = document.createTextNode("Soy un párrafo.");
            var att = document.createAttribute("style");
            att.value = "color: red;";
            elem.appendChild(txt);
            elem.setAttributeNode(att);
document.querySelector("#container").appendChild(elem);
        }
        function deleteLastNode() {
            var container = document.querySelector("#container");
            if (container.lastElementChild)
```

```
                container.removeChild(container.lastElementChild);
        }
    </script>
</body>

</html>
```

A continuación, desglosaremos este **código**:

- **Botones**: como vemos si ejecutamos la página, consta de dos **botones**:
  - ✔ Uno que **añade** un nuevo **párrafo**.
  - ✔ Otro que **elimina** el último párrafo creado.

  Para ello creamos dos **funciones**, un para cada botón, y dentro de ellas hacemos uso de los métodos **createElement()** y **removeChild(),** respectivamente.

- **Modificar estructura**: hay algunas cosas a tener en cuenta, cuando intentamos modificar la estructura **DOM** mediante **JavaScript**:
  - ✔ El **árbol** del DOM se genera cuando la página se carga.
  - ✔ No se pueden realizar **modificaciones** en el DOM hasta que la página está cargada.
  - ✔ Una buena idea es encapsular todas las funciones de manipulación dentro del evento. Por ejemplo:

```
window.onload = function() {
    init();
    doSomethingElse();
};
```

Actividad de aprendizaje 1: árbol de navidad

La actividad se elaborará de forma colaborativa.

Nuestra empresa está tan contenta con nuestra evolución en los proyectos que hemos ido realizando que ha decidido que empecemos a trabajar en proyectos más visuales.

Ahora mismo, están trabajando en un proyecto para representar un árbol de Navidad, de forma virtual.

Para ello nos piden una primera actividad, en la que dados los siguientes archivos (A1.html, star_on.gif, star_off.gif) tenemos que realizar un archivo JavaScript, para conseguir encender la estrella cuando pulsemos el botón.

Las instrucciones del ejercicio están en el propio archivo html.

Entrega el archivo .js en el espacio habilitado para ello.

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Cómo se conoce a los elementos hermanos en DOM?

a. Parent.

b. Sibling.

c. Child.

Lee el enunciado e indica la opción correcta:

¿Cómo podemos añadir un nuevo hijo en el elemento?

a. createElement().

b. createChild().

c. appendChild().

Lee el enunciado e indica la opción correcta:

El árbol del DOM se genera…

a. Cuando la página se carga.

b. Con el primer evento de la página.

c. Cuando la página se empieza a cargar.

6. EL MODELO DE EVENTOS. ASOCIACIÓN DE EVENTOS

RECUERDA

Como vimos en una unidad anterior, nos referimos a eventos como acciones que ocurren en una aplicación web. Ésta es capaz de escuchar esos eventos, y actuar en consecuencia, en función de lo que ocurra en la página.

Con los eventos de **HTML DOM** se permite a **JavaScript** registrar diferentes **manipuladores de evento**s (handlers) en los distintos elementos que contiene un documento HTML.

TOMA NOTA

Usualmente estos eventos son utilizados mediante funciones, que no serán ejecutadas hasta que el evento ocurra. Por ejemplo, una pulsación de teclado o un clic de ratón.

Podemos referirnos a la **documentación** de **W3C** para consultar una **lista de eventos**, pero algunos de los más utilizados pueden ser:

- **Click**: ocurre cuando el usuario clica sobre un elemento de la página.
- **Dblclick**: ocurre cuando el usuario hace doble clic sobre un elemento de la página.
- **Drag**: ocurre cuando un elemento es arrastrado.
- **Focus**: ocurre cuando un elemento recibe el foco, es decir, se vuelve activo.
- **Keypress**: ocurre cuando el usuario realiza una pulsación en una tecla.
- **Wheel**: ocurre cuando el usuario realiza scroll mediante la rueda del ratón.

Como vemos, estos eventos suelen ir asociados a los distintos **elementos** que nos encontremos en una página. Así, por ejemplo, podríamos diferenciar si el usuario realiza un clic sobre un campo de texto o un link de la página.

Vídeo: eventos

Visualiza el siguiente vídeo en el que hablamos sobre eventos.

7. PROGRAMACIÓN DE EVENTOS. EVENTOS DEL DOM EN LOS DIFERENTES NAVEGADORES. PERSONALIZACIÓN DE EVENTOS EN TIEMPO DE EJECUCIÓN

Lamentablemente, en cuanto a la gestión de eventos se refiere, normalmente existen muchas **diferencias** en los distintos **navegadores**. Esto provoca que un desarrollador tenga que tener especial cuidado con el uso de los **eventos**, los **métodos** y las **propiedades** que se usan.

Dependiendo del navegador en que se ejecuta la página de JavaScript el resultado podría ser diferente, por lo que es un tema a tener muy en cuenta.

Una posible solución para actuar de manera más o menos **independiente** del navegador, sería crear una estructura parecida a la siguiente:

```
function crearEvento(elemento, tipoEvento, funcion) {
    if (elemento.addEventListener) {
        elemento.addEventListener(tipoEvento, funcion, false);
    } else if (elemento.attachEvent) {
        elemento.attachEvent("on" + tipoEvento, funcion);
    } else {
        elemento["on" + tipoEvento] = funcion;
    }
}
var boton = document.getElementById("boton1");
crearEvento(boton, 'click', function () { alert("hola") });
```

La **idea** es la siguiente:

- En lugar de intentar añadir un evento a un elemento de manera tradicional, creamos una **función envolvente**, a la cual llamaremos, pasándole el elemento, el evento a capturar y la función a ejecutar.
- Dentro de la función intentamos **añadir el listener** con el método addEventListener(), el utilizado en los navegadores actuales.
- Si esto no funciona, probamos otra alternativa, que sería añadirlo mediante el **método attachEvent()**.
- Finalmente, si ninguna de las dos anteriores ha funcionado probaremos a añadirlo asignando el **evento en línea**, como un atributo más del elemento.

Actividad de aprendizaje 2: encender estrellas

Siguiendo con el proyecto del árbol de Navidad, ahora nos piden trabajar sobre los eventos.

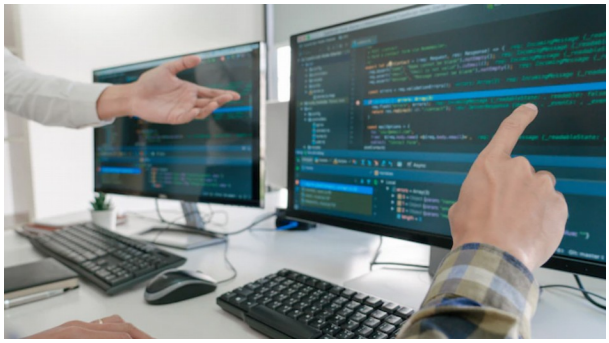
Tendremos una cuadrícula con estrellas, que habremos de ir encendiendo al pasar el ratón por encima.

Al igual que en la actividad anterior tienes los tres archivos (A2.html, star_on.gif, star_off.gif), y las instrucciones en el propio html.

Entrega el archivo .js en el espacio habilitado para ello.

8. DIFERENCIAS EN LAS IMPLEMENTACIONES DEL MODELO. MOTORES DE NAVEGACIÓN (USER-AGENT). DIFERENCIAS ENTRE NAVEGADORES

Hemos visto que el modelo **DOM** puede actuar diferente dependiendo del navegador sobre el que se ejecute la aplicación. Es por ello que aparece el **concepto de BOM**.



DESTACADO

El BOM (Browser Object Model) es la estructura que permite a JavaScript comunicarse con el navegador.

A continuación, veremos algunos **aspectos relevantes** relacionados con el **BOM**:

- **I**. Así como el DOM es un estándar muy bien definido, el **BOM** en cambio, **depende de cada navegador** o **user-agent** (entendiendo que puede existir el mismo navegador para diferentes dispositivos como PC, móviles, consolas…).
- **II**. Como hemos visto, la **raíz** del DOM es el elemento document, en cambio en el **BOM** nos encontramos con que la raíz es el elemento global window.
- **III**. Como todos los elementos cuelgan de **window**, nos podemos permitir el lujo de hacer cosas como **alert** ("Hola"), en lugar de **window.alert** ("Hola").

  Para saber todas las propiedades y métodos del objeto window puedes consultar la referencia de **MDN**.

Con los objetos del BOM podemos hacer cosas interesantes como el siguiente ejemplo, en el cual intentaremos calcular la **rotación** del dispositivo en función de los **acelerómetros**.

Código: calcular rotación

No nos importará muy bien de qué dispositivo se trata, ya que vamos a basarnos en **información** que nos dará el propio **navegador**. Sin embargo, sí que incluiremos unas restricciones para excluir los navegadores de dispositivos sin **acelerómetro**, basándonos, por ejemplo, en el sistema operativo.

```
var regexp = /(android|iPad|iPhone|iPod)/i;


if (regexp.test(navigator.userAgent)) {
    if (window.DeviceOrientationEvent) {
        window.addEventListener( "deviceorientation", handleOrientation,
true);
    } else {
        document.getElementById( "data").innerText = "Your browser
doesn't support motion events!";
    }
} else {
    document.getElementById("data").innerText = "Your device is " +
navigator.userAgent + " and has not accelerometer!";
}


function handleOrientation(event) {
    alpha = event.alpha;
    beta = event.beta;
    gamma = event.gamma;
   document.querySelector("#data").innerHTML = "Alpha: " + alpha +
"<br/>Beta: " + beta + "<br/>Gamma: " + gamma;
}
```

En el ejemplo usamos una expresión regular (vistas en la unidad anterior) para comprobar que usamos un dispositivo Android, iPhone, iPad o iPod, descartando así el resto. A partir de aquí, intentamos capturar la orientación del dispositivo mediante diferentes métodos y propiedades.

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Qué significan la siglas BOM?

a. Browser Object Mechanism.

b. Browser Object Model.

c. Beta Object Model.

Lee el enunciado e indica la opción correcta:

¿Cuál es el evento que se da al arrastrar un elemento?

a. Focus.

b. Wheel.

c. Drag.

Lee el enunciado e indica la opción correcta:

¿Cómo se denominan también las aplicaciones multi-cliente?

a. Cross-Browser.

b. Multi-capa.

c. MVC

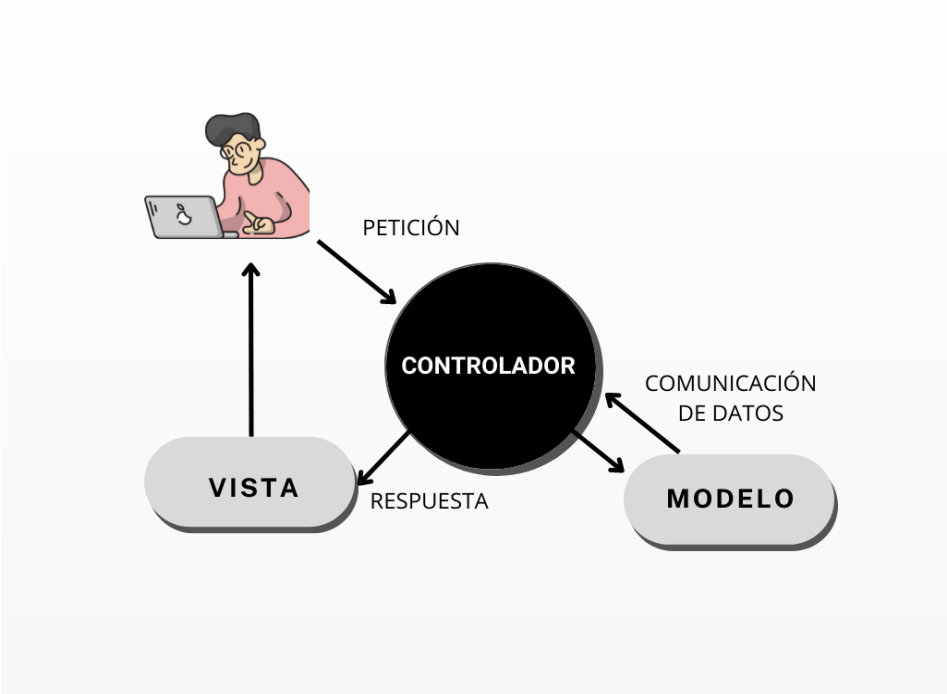
9. DESARROLLO DE APLICACIONES WEB EN CAPAS. SEPARACIÓN DE LAS FACETAS DE CONTENIDO, ASPECTO Y COMPORTAMIENTO DE LAS APLICACIONES

En programación, cuando hablamos del desarrollo de aplicaciones en capas nos referimos al hecho de programa de tal manera que ciertos aspectos de la aplicación sean lo más independientes posible.

Se trata de la **técnica de desarrollo** utilizada habitualmente, ya que posee muchas **ventajas**. Entre otras:

- Mejora la **modularización** del código.
- Permite la **separación** de datos, interfaz y comportamiento.
- Mejora del **trabajo en equipo**.
- Permite la **especialización** de los distintos miembros del equipo.

En concreto la estructura de capas más utilizada es la que se conoce como **MVC** (Modelo-Vista-Controlador).



A continuación, veremos las **capas** de las que se **compone el MVC** y como se trasladan en las **aplicaciones web**:

- ➔ **Capas del MVC**: podemos resumir estas **tres** capas en:
  - ✔ **Modelo**: se encarga de gestionar la información y datos de la aplicación.
  - ✔ **Vista**: se encarga de mostrar la información al usuario, mediante la interfaz.
  - ✔ **Controlador**: se encarga de realizar las distintas operaciones, funciones y cálculos, con los datos del modelo, para posteriormente mostrarlos al usuario mediante la vista.
- ➔ **Capas MVC en aplicaciones web**: en el caso específico de las aplicaciones web, hablaríamos de un modelo de 3 capas, en el cual:
  - ✔ El **modelo**, usualmente, son los **datos** guardados en algún servidor remoto.
  - ✔ La **vista** se refiere a la **interfaz** de usuario. Es lo que conocemos propiamente como la página web en sí.

- ✔ El **controlador** podría ser el **código de script**, JavaScript en nuestro caso, que se encarga de todo lo referente a cálculos e interacción con el cliente.

Vídeo: modelo vista controlador

Visualiza el siguiente vídeo sobre el patrón Modelo Vista Controlador:
https://www.youtube.com/embed/zhSDjntidws

10. DESARROLLO DE APLICACIONES MULTI-CLIENTE

Ya hemos visto que las aplicaciones web pueden actuar **diferente** del **navegador** en el cual se ejecuten, y hemos visto **algunas** técnicas para poder paliar estos efectos.

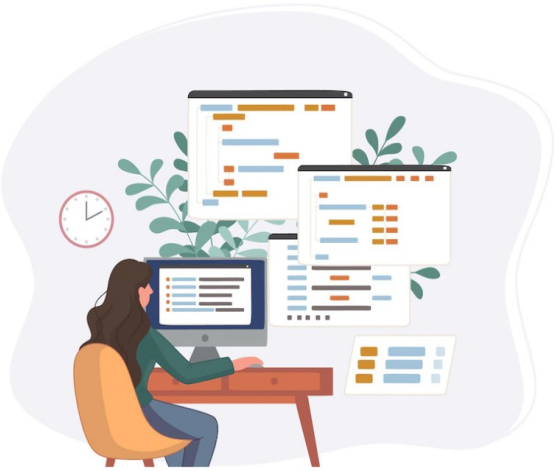
En relación a esto, durante este capítulo hablaremos de las aplicaciones **multi-cliente** o **cross-browser**, que no son otra cosa que aplicaciones que están pensadas para que se vean exactamente igual en cualquier navegador web.

Pese a que el W3C define estándares para HTML, CSS o JavaScript, el problema es que las empresas que desarrollan los navegadores, a menudo las interpreta de manera diferente, agregando funcionalidades o etiquetas que no estás reflejadas en el estándar.

A continuación, veremos algunos **aspectos relevantes** sobre las **aplicaciones multi – cliente**:

Una de las claves para poder desarrollar aplicaciones multi-cliente es tener muy claras las **diferencias entre los navegadores**, para así poder adaptar el código en consecuencia. Veamos algunos aspectos relevantes con respecto a las diferencias entre los navegadores:

- **JavaScript**: esto aplicado a **JavaScript**, se traduce en que podemos ejecutar partes del código en función del navegador que estemos utilizando, como ya hemos visto en algún capítulo anterior. Por ejemplo, podríamos comprobar si estamos ejecutando la aplicación en Internet Explorer de la siguiente manera:



```
// Comprobamos si es un navegador IE
if (navigator.appName.indexOf("Explorer") != -1) {

// Es un navegador W3C
} else {

}
```

- **Comprobaciones**: lo ideal, de todas maneras, es usar estas **comprobaciones**, que contienen la lógica multi-cliente, dentro de una función separada, y luego envolverla con otra función de nivel superior, que será la que llamemos finalmente, creando así la sensación de que el código es realmente independiente del navegador.

Actividad de aprendizaje 3: diferencias entre navegadores

Al ejecutar el último proyecto, que encendía las estrellas a medida que pasábamos por encima, nuestro supervisor se ha dado cuenta de que no funcionaba de manera correcta en su navegador. Lamentablemente sólo lo habíamos probado en el nuestro, pero él tiene una versión mucho más antigua.

Ponte de acuerdo con un compañero y realizad un pequeño documento explicando los pasos que habría que hacer para adaptar una aplicación para que se pueda ejecutar en diversos navegadores de la misma forma.

Entrega un archivo .pdf en el espacio habilitado para ello.

Vídeo: propiedades del navegador

Visualiza el siguiente vídeo en el que hablamos sobre las propiedades del navegador.

RESUMEN

Prácticamente toda esta unidad ha versado acerca del **concepto de DOM**, su significado, utilización y características.

Hemos visto cómo un documento HTML se puede transformar en una **estructura de árbol jerárquica**, que representa todos sus elementos, mediante el **DOM**.

Estos elementos, además, tienen sus propiedades y métodos, que definen la manera de utilizarlos, y como podemos trabajar con ellos a través de **JavaScript**.

Hemos avanzado con el **modelo de eventos**, que ya vimos en capítulos anteriores, viendo como lo podemos enlazar con los **elementos** del DOM, para poder trabajar de forma más sencilla.

Finalmente hemos trabajado con la **programación multi-capa**, viendo el **Modelo-Vista-Controlador**, y con las aplicaciones **multi-cliente** o **cross-browser**, intentando hacer lo más independiente posible el código del navegador.

En resumen, hemos visto la mayoría de las posibilidades que nos ofrece el DOM en el trabajo con **aplicaciones web** desde el punto de vista del cliente.

RECURSOS PARA AMPLIAR

PÁGINAS WEB

- HTML DOM Element Object: *https://www.w3schools.com/jsref/dom_obj_all.asp* [Consulta agosto 2022].
- HTML DOM Event Objects: *https://www.w3schools.com/jsref/obj_events.asp* [Consulta agosto 2022].
- Node – Web APIs. MDN: *https://www.w3schools.com/jsref/obj_events.asp* [Consulta agosto 2022].

BIBLIOGRAFÍA

PÁGINAS WEB

- JSON - JavaScript | MDN: *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON* [Consulta agosto 2022].
- W3C: *https://www.w3.org/* [Consulta agosto 2022].

GLOSARIO

- **Encapsular**: se tratar de independizar un código lo máximo posible, de tal manera que no importe el cómo está hecho, si no lo que hace.
- **User-Agent**: aplicación informática que accede a la WWW. Típicamente, un navegador web.
- **W3C**: world Wide Web Consortium, o Consorcio WWW, que gerena estándares y recomendaciones a nivel Web.
- **XML**: eXtensible Markup Language, lenguaje de marcas desarrollado para almacenar datos de forma jerárquica.