TEMA 3.TECNOLOGÍAS DE DESARROLLO WEB

Objetivos:

- *Identificar los recursos necesarios para la puesta en marcha de un proyecto web.*
- *Conocer las diversas tecnologías que se pueden utilizar a la hora de desarrollar un Sistema de Información Web*
- *Comprender la importancia de seguir los estándares y las normas de estilo*
- *Ser capaz de decidir, a la hora de planificar un proyecto, qué tecnología/s usar y justificarlo adecuadamente.*

LO BÁSICO: DOMINIO Y UN ALOJAMIENTO

Lo normal en el desarrollo de la profesión es que nos llegue un potencial cliente, o un cliente habitual, y nos pida desarrollar una aplicación web (englobaremos en esta terminología tanto los sitios webs como los sistemas de información web). Normalmente, como se ha visto en el tema 2, no viene con los requisitos claros, y mucho menos cerrados.

En los sistemas de escritorio, tenemos claro que hay unos requisitos hardware que incluir en la especificación del proyecto, y en estos sistemas web la analogía sería el servidor, que como veremos, depende de algunos factores diferentes a los sistemas de escritorio. Pero también hay algo que puede hacer que el proyecto sea un éxito o un fracaso: el nombre de dominio.

Elegir el dominio correcto

Supongamos que llega un cliente cuyo negocio se llama "Alfombras iraníes Shan-Kai", y se empeña en que su dominio sea www.alfombrasiraniesshan-kai.com ¿Es un dominio adecuado? ¿Es fácil de recordar? ¿Se escribe fácilmente sin error? Claramente no.

Los nombres de dominio cortos y fáciles de escribir y leer son atractivos y fáciles de recordar, por lo que es algo que merece la pena meditar bien en un proyecto web. Claro, que lo más normal es que estas características ya las haya pensado alguien antes y www.alfombras.com esté reservado. Hay que echarle imaginación. Añadir adjetivos, un lema, en definitiva algo que distinga: www.alfombrasvoladoras.com … vaya, ocupado también. www.laalfombrademicasa.com está libre… pero algo largo. www.alfombrasiraines.com está libre, y es fácil, describe lo que queremos… ¡adjudicado!

Pues este proceso habría que hacerlo cuando se nos plantea un proyecto. Algunos trucos:

- Los guiones no son bien recibidos
- Evitar errores ortográficos comunes
- Incluir palabras inglesas en dominios españoles
- Usar nombres que no tienen que ver con el negocio
- Utilizar extensiones extrañas (.biz, .com.es)

Una cosa que hay que tener clara, al menos a la hora de reservar el dominio es el propósito de los primeros niveles:

- .org se ideó para organizaciones sin ánimo de lucro
- .com se pensó para empresas

- .edu es específico para entes educativos, y de hecho no es de libre reserva
- .es es el dominio propio de España

Hay que intentar hacer el esfuerzo de que el TLD (*top level domain,* dominio de primer nivel) ya nos dé información sobre el sitio web, y no cometer errores de usar el .com para una ONG, por ejemplo. Otra cosa distinta es usar TLD de pequeños países porque vienen bien, por ejemplo:

- .tv no es para la televisión, es el dominio de Tuvalu
- .fm no es para emisoras de radio, sino que se pensó para la Federación de Micronesia

Para reservar dominios, pues podemos acudir a cualquier registrador de los que hay en el mercado. A la hora de reservar el dominio, tendremos que rellenar varios datos, y es importante que tengamos control sobre todos y cada uno de estos aspectos:

- **Propietario** o **Titular** - Registrant : Persona física u organización que registra un nombre de dominio específico. El propietario tiene el derecho a utilizar el nombre de dominio por un periodo específico de tiempo, siempre que cumpla las condiciones del registrador y pague las cuotas. El registrante es el representante legal sobre el nombre de dominio.
- **Contacto Administrativo** - Administrative Contact: Persona física, entidad u organización autorizada para interactuar con el registrador en lugar del propietario del dominio. El contacto administrativo está capacitado para responder preguntas no técnicas relacionadas con el registro del nombre del dominio. En todos los casos, el contacto administrativo es visto como el punto de autoridad de contacto para el nombre del dominio.
- **Contacto de Pago** - Billing Contact: Persona física, entidad u organización destinada a recibir la factura del registro del dominio y de las posteriores renovaciones.
- **Contacto Técnico** - Technical Contact: La persona física u organización responsable para las gestiones técnicas de la zona delegada. Este contacto habitualmente mantiene los DNS para el dominio. El contacto técnico debe poder contestar las preguntas técnicas relacionadas con el nombre del dominio.

Muchas empresas utilizan la estrategia comercial de regalar el nombre del dominio con el alojamiento, pero a cambio, se sitúan ellos como propietarios o contacto administrativo del dominio, por lo que es casi imposible llevárselo a otro registrador más adelante.

Los dominios **.es** tienen una gestión diferente a los .com, .net, .org

ESNIC es el organismo oficial registrador de dominios .es, y, por ejemplo, no permite el cambio automático de los datos del propietario del dominio, sino que es un proceso con más garantías.

*La **Internet Corporation for Assigned Names and Numbers*** **(ICANN)** es el organismo sin ánimo de lucro que se encarga de gestionar los dominios globales y los servidores DNS raíz.

El alojamiento web

Para que nuestro sistema web funcione hace falta una máquina donde se ejecute, que esté funcionando un servidor HTTP y que sea localizable por las DNS. Esta última parte lo podemos arreglar teniendo un nombre de dominio y configurando los registros adecuadamente. La red está diseñada para ser una comunidad abierta y cualquiera puede montar un servidor en su casa, pero no es tan fácil.

Sobre dónde alojar la web, hay varias posibilidades, que dependerán del contexto en el que se desarrolle el proyecto. Podemos enumerar algunas posibilidades:

- Contratar un alojamiento compartido en un proveedor comercial.
- Contratar un servidor dedicado administrado por un proveedor comercial.
- Usar un servidor propio y alojarlo en un centro de datos.

- Usar un servidor propio y alojarlo en las instalaciones de la empresa.
- Alojar la web en un servidor gratuito.

Cada una de estas opciones tiene sus *pros* y sus *contras*, que pasamos a comentar, y que deberán ser tenidos en cuenta a la hora de desarrollar el proyecto.

Ejercicio 17. *Haz un estudio de mercado sobre los precios d*
diversos registradores de los siguientes dominios:
www.sibw201516.es
www.sibw201516.com
www.sibw201516.edu
www.sibw201516.org.es

Ejemplos de registradores: Dreamhost, CDmon, Arsys, Hostali

Ejercicio 18. Averigüe cómo se resuelve el problema qu
propietario de la marca "MiAsignaturaMola" y
miasignaturamola.es por un tercero que no tiene nada qu

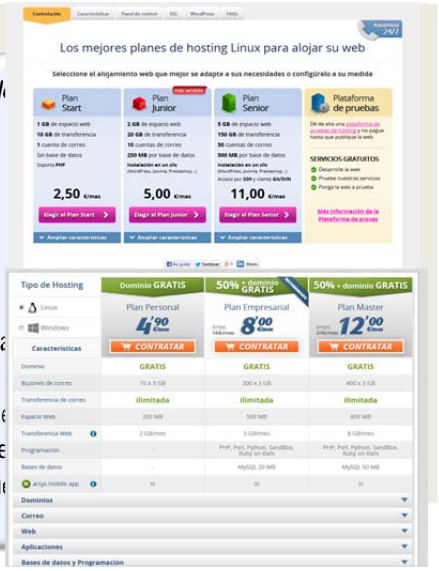


Ilustración 17: Ejemplo de ofertas comerciales de alojamiento compartido.

Alojamiento compartido en un proveedor comercial

Es sin duda lo más cómodo a la hora de un proyecto de un sitio web, especialmente si no requiere de una gran infraestructura técnica o software.

Hay multitud de empresas que ofrecen *hosting* con recursos limitados en cuanto a:

- Espacio web
- Espacio y número de bases de datos
- Espacio y número de cuentas de e-mail
- Transferencia mensual permitida

Obviamente, cuanto más podamos pagar, más prestaciones obtendremos.

En estos casos, el mantenimiento hardware y software del servidor es por cuenta del proveedor, y es aquí donde viene lo mejor (no tenemos que preocuparnos de actualizar el SO, ni el servidor, ni configurar cortafuegos, etc.) y lo peor, pues normalmente estos servicios sólo ofrecen lo "estándar", a saber: Php, MySQL, Posgres y con suerte Python o CGI. Si queremos un proyecto en Java, Ruby u otros lenguajes que requieran software específico, no es una buena solución.

Otro factor negativo es que nuestro sitio web no estará solo en el servidor, sino que compartirá recursos con otros 20, 40, 50 o 100 sitios web que, si alguno está muy mal programado o tiene mucho éxito, consumirá los recursos de forma totalmente egoísta.

Factores a tener en cuenta a la hora de decantarse por uno u otro:

- Horario de atención y tiempo de respuesta del servicio técnico
- Coste del exceso de transferencia consumida
- Nivel de respuesta de los servidores
- Tecnología software y hardware ofrecida
- Precio
- Flexibilidad en las prestaciones

El coste puede oscilar, entre 2,50€ y 30€ mes.

Servidor dedicado administrado por un proveedor comercial

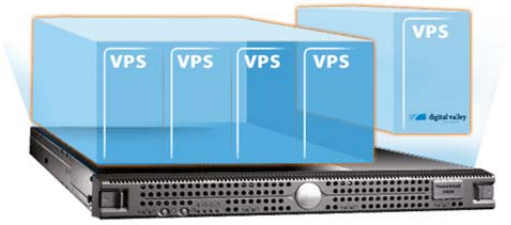
Este tipo de productos está pensado para webs que requieren pocos requisitos software pero bastante potencia hardware, bien porque reciban muchas visitas o porque se requiera que la dirección IP no sea compartida por ningún otro dominio (por cuestiones de certificados SSL).

Tiene la ventaja que no es necesario tener un administrador de sistemas contratado para la gestión del servidor, sino que se encarga el proveedor de ello. Por el contrario, al igual que los compartidos, el software que se puede ejecutar sobre él también viene determinado por la empresa que nos ofrece esos servicios.

Obviamente, su coste va acorde a las prestaciones que obtenemos, y es mucho más caro. Pensemos que el compartido no es más que un servidor cuyos costes se "trocean" entre todos los *hosting* alojados. Aquí se cubren con un único alojamiento, por lo que los costes pueden ir aproximadamente entre 100€ y 400€ al mes.

Ilustración 18: Ejemplos de servidores dedicados

Servidores virtuales



Un paso intermedio entre los servidores dedicados y los compartidos son los denominados servidores virtuales o VPS (. Estos productos hacen uso de la virtualización de máquinas (VMware, Xen, etc.) y permiten que cada *servidor virtual* tenga los recursos garantizados, independientemente que en una misma máquina física haya varios servidores virtuales.

Tienen la ventaja que se puede modificar fácilmente el número de núcleos de procesamiento, tamaño de disco, etc. , se optimizan recursos hardware (una sola máquina física tiene suficiente potencia para albergar más de un servidor virtual), y se simplifica la gestión de las copias de seguridad.

Usar un servidor propio y alojarlo en un centro de datos

Bajo esta modalidad vamos a incluir tanto el caso de que la máquina física sea nuestra, en cuyo caso hablaríamos de un *housing*, como si es un servidor –virtual o físico- administrado por nosotros o nuestra empresa.

Ilustración 19: Fachada del Cloud Center Andalucía.

Un Centro de Procesos de Datos es un espacio físico, con alta capacidad de transmisión de datos, con altas medidas de seguridad y con el acondicionamiento climático y eléctrico necesario para dar soporte a decenas de ordenadores conectados a la red. Uno de los principales de España es ESpanix (www.espanix.es) que se encuentra en España, en el CPD Banesto, y donde tienen sus máquinas los principales proveedores de servicios hosting del país. En Granada tenemos el CPD de Grupo

Trevenque, el Cloud Center Andalucía, situado a escasos 2kms de nuestra ETSIIT, con capacidad para albergar más de 5000 servidores físicos.

Las ventajas de tener nuestro propio servidor son claras: total autonomía en la configuración y gestión del servidor e independencia a la hora de elegir la tecnología software sobre la que ejecutar nuestro proyecto. Por supuesto, esto hace que sea más económico, a igualdad de prestaciones hardware, que un compartido o administrado, pero por el contrario tenemos que asumir los costes en horas de trabajo del mantenimiento del sistema.

Usar un servidor propio y alojarlo en las instalaciones de la empresa.

Ilustración 20: CPD de la UGR

Por cuestiones diversas, puede que al cliente o nuestra empresa no le convenza el hecho de tener la máquina física a kilómetros de distancia de sus instalaciones, y prefiera tenerlo controlado. En este caso, ya no es sólo tarea del cliente o nuestra empresa la gestión del software, sino que también habremos de incluir en nuestro proyecto los requisitos de conectividad, climatización, potencia física, etc. necesarios para garantizar la estabilidad del sistema.

Es el caso de la UGR o de grandes empresas que tienen *músculo* suficiente para poder disponer de las infraestructuras y recursos humanos necesarios para la gestión de dichos equipamientos.

Alojar la web en un servidor gratuito.

En los inicios de la Web, allá por la década de los noventa, los servicios de hosting web eran muy caros, y aparecieron lugares como *geocities.com*, *lycos.es, galeon.com* y otros muchos, que permitían alojar webs en HTML gratuitamente a cambio de que aparecieran banners de publicidad.

Hoy en día apenas quedan lugares que permitan alojar webs gratis, y realmente, gratis nunca es. Siempre hay que dar contrapartidas como: no poder usar un dominio propio, inclusión de publicidad, restricción de tecnologías, etc.

Por tanto, para un proyecto web serio, debería ser una opción totalmente descartada.

En resumen, podemos encontrar en el mercado soluciones de alojamiento web que se adecúen a nuestro proyecto, y transmitir a nuestra empresa o cliente la idoneidad o no de cada uno de ellos.

El uso de servicios "gratis" suele tener implicaciones de marketing o legales que pueden volverse en nuestra contra.

Un consejo: como Ingenieros de Software, dedíquense principalmente a lo que saben hacer bien: analizar, diseñar, programar y gestionar proyectos. Aunque conozcan algo de administración de sistemas, en la medida de lo posible deleguen en los profesionales para el día a día.

TECNOLOGÍAS WEB DEL LADO CLIENTE: HTML, CSS, JAVASCRIPT Y OTRAS

A la hora de desarrollar una aplicación basada en web, su propio nombre o requisito no funcional indica que se ha de ejecutar sobre un navegador web. Su arquitectura deberá ser cliente/servidor, pero tanto en el lado cliente como en el servidor, las posibilidades son inmensas en cuanto a los lenguajes y paradigmas a utilizar.

Un navegador web es capaz de interpretar y visualizar información codificada en HTML (HyperText Markup Language), organizarla según indique el CSS (Cascade Style Sheet), interpretar y ejecutar scripts (Javascript, VBScript©) e incluso mediante el uso de plugin de terceros, visualizar películas Flash© que ejecuten un programa hecho en lenguajes como Flex. También, gracias a un complemento, se pueden ejecutar *applet* Java©, o ejecutar un videojuego online hecho con Unity©.

Son tales las posibilidades que a la hora de planificar esta asignatura hemos tenido que seleccionar uno de los caminos posibles, y hemos optado por centrarnos en lo más común y a su vez más propio de la web: HTML + CSS + JS. ¿Por qué tres cosas?, pues porque cada una se encarga de una parte de lo que el navegador va a realizar:

- HTML codifica la estructura y el contenido de lo que queremos mostrar
- CSS define la estética y formato, independientemente del contenido
- JS contiene el código dinámico de la aplicación en el lado cliente

Esta enumeración anterior es muy importante, pues su cumplimiento o no denota la profesionalidad y conocimiento del que lo ha hecho y la calidad del producto desarrollado. No es lo mismo estructurar un documento (dividir su contenido en componentes distintos como encabezados, párrafos, listas, etc.) y darle formato (poner cursivas, cambiar colores, márgenes, etc.).

HTML5

HTML es un lenguaje de marcas. Una página HTML consiste en una mezcla de texto e imágenes enmarcados en etiquetas que indican al navegador su función lógica o semántica.

Por ejemplo, las dos siguientes líneas:

```
<h1>Sistemas de Información Basados en Web</h1>
<p> Sistemas de Información Basados en Web</p>
```

representan el mismo texto (“Sistemas de Información Basados en Web”) pero su función semántica en el documento HTML son diferentes. En la primera línea es un encabezado de primer nivel, y en el segundo caso es un párrafo normal. ¿Cómo se mostrarán? No nos interesa saberlo ahora, HTML tan sólo se debe ocupar del contenido, de la estructura semántica de la información.

¿Por qué tanto empeño en separar contenido de formato? Veamos varios motivos:

- Incrustar elementos de formato en el contenido conlleva un más costoso mantenimiento de lo desarrollado, pues los mismos valores pueden estar repetidos en varias etiquetas (márgenes, colores, fuentes, alineación, etc…)
- Un documento que contenga sólo contenido y organización lógica puede ser mostrado de diferentes maneras sin tener que tocarlo, tan sólo cambiando la parte estética.
- Los documentos lógicos transmiten significado. Por ejemplo, sería muy fácil realizar índices tan sólo leyendo los encabezados… y pensemos la utilidad de esto para un lector de pantalla para invidentes.

Por tanto, vamos a centrarnos en este apartado sólo en HTML, y en concreto en la versión HTML5 surgida en 2012.

El desarrollo de HTML5 ha sido coordinado por el W3C, y han intervenido los principales actores del negocio de la Web: AOL, Apple, Google, IBM, Microsoft, Mozilla, Nokia, Opera, etc.

Ha sido desarrollado pensando en que cualquier contenido necesario para la Web se pueda implementar sin necesidad de plugins adicionales, por lo que soporta nativamente reproducción de audio y video, visión 3D, animaciones, etc.

Además, como las versiones anteriores de HTML, es multiplataforma

Pasemos por tanto a desgranar los elementos más importantes de HTML5 que nos ayudarán a desarrollar un Sistema de Información Basado en Web

En el código 1 podemos ver un documento HTML con el conjunto mínimo de etiquetas exigido para que sea válido.

Se puede apreciar que las etiquetas son elementos enmarcados entre los símbolos < y >. En ese ejemplo podemos encontrarnos con las etiquetas `<html>`, `<head>`, `<meta>`, `<title>` y `<body>`.

La mayoría de éstas se encuentran por pares, pues su función es delimitar partes del contenido. Por tanto, podemos concretar que un elemento HTML se compone de:

```
<!DOCTYPE html>
<html>

<head>
<meta charset="UTF-8">
<title>Título del documento</title>
</head>

<body>
<!-- Aquí va el contenido..... -->
</body>

</html>
```

Código 1: Documento HTML mínimo

- Una etiqueta de *inicio*
- Una serie de *atributos*, que son pares elemento-valor que añaden información importante a la etiqueta (p.ej. charset="UTF-8")
- Contenido textual
- Una etiqueta de *fin*, que tiene el mismo nombre que la de inicio pero es precedida por el símbolo '/'. Algunos elementos carecen de etiqueta de fin, como es el caso de *meta*, pero son los menos.

```
<!DOCTYPE html>
```

es la declaración del tipo de documento, y en este caso es específica para HTML5. La declaración <!DOCTYPE> es necesaria pues es la que le indica al navegador con qué tipo de documento se va a enfrentar y cómo tiene que interpretarlo La del ejemplo es la estándar para HTML5, aunque hay muchas otras. Por ejemplo en la web de la UGR (www.ugr.es) encontramos la de XHTML1 estricto:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Ejercicio 19. *Accede a http://www.w3schools.com/tags/tag_doctype.asp y documenta en tus apuntes los diversos tipos de <DOCTYPE> aceptados por los navegadores hoy en día.*

La estructura global de un documento HTML es la delimitada por la etiqueta <html> y se compone de dos secciones principales:

- La cabecera, delimitada por la etiqueta <head>, que incluye información general sobre el documento, como su título (delimitado por <title>), información para buscadores (mediante etiquetas <meta>), información sobre el formateado de la página, con CSS incrustado o vinculado, scripts, etc.
- El cuerpo, delimitado por la etiqueta <body> que incluye el contenido real de la página.

En nuestro ejemplo del código 1 hay poco más que enseñar, de hecho no se vería más que una página en blanco. La línea `<!-- Aquí va el contenido..... -->` es ignorada, pues **los símbolos <! - - y - -> delimitan los comentarios.**

Pasamos por tanto a comentar algunos elementos fundamentales a la hora de desarrollar una página web en HTML5.

Hiperenlaces <a>

La etiqueta **<a>** define un hiperenlace, esto es, un contenido en el que al pulsar sobre él nos dirigimos a otra página o a otra parte dentro de la misma página.

El atributo más importante de la etiqueta <**a**> es el `href` que nos indica el destino del enlace. Si se define href, entonces se pueden incluir otros atributos como `hreflang, download, target, etc.`

Ejemplos:

```
<a href="http://www.google.com/" target="_blank">Google</a>
```

Nos muestra el texto Google y abrirá una nueva pestaña del navegador con la url www.google.com si hacemos click en dicho texto. ¿Cómo sabemos que es una nueva pestaña? Por el valor del atributo `target`, que es "`_blank`". Otros valores del atributo target son `_parent`, `_self` o `_top`.

```
<a href="notas.html">Ver Mis notas</a>
```

Nos muestra el texto Ver Mis notas y al hacer clic sobre él nos llevará a la página notas.html del mismo directorio en el que nos encontramos en la misma ventana y pestaña del navegador.

Un caso especial son los elementos <a> sin atributo href, pero con id. En este caso se refieren a marcas internas dentro la página, a las que se puede llegar por enlaces. Por ejemplo:

```
<a href="#frase1">Todo lo que des, lo recibas duplicado.</a>
```

nos llevaría al elemento HTML

```
<a id="frase1">Frase importante a no olvidar</a>
```

que puede estar antes o después del enlace, o incluso ser referido como "pagina.html#frase1" desde otro documento html.

Como se puede comprobar, es la etiqueta <a> la que permite a la Web ser lo que es hoy en día.

Imágenes <img>

El elemento imagen es de los denominados *independientes*, esto es, emplea una única etiqueta, **`<img>.`** Su sintaxis es:

```
<img src="url" alt="Texto alternativo">
```

Cuando el navegador se encuentra con una etiqueta de imagen, realiza una nueva petición HTTP al servidor para descargar el archivo especificado en el atributo `src` (de source).

El texto alternativo, indicado bajo el atributo alt es útil para mostrar información cuando el navegador no puede enseñar la imagen (porque ha desaparecido del servidor o porque se ha cortado la conexión), o cuando Google visita la página y queremos indicarle qué hay en ella.

Si además queremos especificar el ancho y alto con el que queremos mostrar la imagen, pues tan sólo hay que decirlo:

```
<img src="url" alt="Texto alternativo" width=”100px” height=”300px”>
```

El uso de imágenes en una página web ha de tratarse con cierto cuidado, pues hay que distinguir entre dimensiones y tamaño. Podemos caer en la tentación de usar una imagen original de 1024x768 píxeles y 2MB de tamaño para una miniatura de 50x30 píxeles, lo que a todas luces es excesivo. Lo que hay que hacer en ese caso es una versión reducida de la imagen, que quizá nos ocupe 15-20KB y utilizarla para mostrar la miniatura.

Una ventaja de introducir el ancho y el alto de la imagen en el código es que el navegador, a pesar de no haber recibido el archivo con la imagen aún, conoce sus dimensiones y puede ir organizando el contenido de la página.

Ejercicio 20. *Documente en sus apuntes los siguientes formatos de imágenes: GIF, PNG-24 y JPEG, destacando para cada uno los siguientes aspectos: tipo de compresión (con/sin pérdida), máximo de colores, manejo de transparencias y para qué es más indicado cada formato.*

Ejercicio 21. *¿Qué formato utilizaría para incluir en su documento HTML:*

a. *Una fotografía de la Alhambra?*
b. *El logotipo de la ETSIIT?*
c. *El logo de la Junta de Andalucía sobreimpreso en un mapa de color marrón?*

Ejercicio 22. *Analice la portada de www.elmundo.es y estudie el tamaño de cada una de sus fotografías. Para realizar este ejercicio, te puede ayudar la extensión Webdeveloper de Firefox.*

Elementos de Texto

Todo texto se agrupa en párrafos y tiene una serie de encabezados que sirven para organizar las secciones. Pues el texto HTML se organiza también igual, mediante sus etiquetas:

- Las cabeceras se definen mediante etiquetas que van de **<h1>** a **<h6>**
- Los párrafos se definen mediante la etiqueta **<p>**

Cuestiones importantes:

- ✓ Las cabeceras **no se usan** para poner el texto más grande o en negrita. Se usan para indicar que el texto enmarcado por <hX> </hX> es una cabecera.

- ✓ Las cabeceras son usadas por los motores de búsqueda para indexar la web
- ✓ Los saltos de párrafo se indican acabando un párrafo (</p>) y comenzando otro, no insertando un salto de línea **
**

Otros elementos de descripción de texto interesantes son las listas, numeradas y no numeradas. Las listas numeradas se indican con la etiqueta <**ol**> (de *ordered* list) mientras que las no numeradas se indican con <**ul**> (*unordered* list). Ambas están formadas por elementos o ítems indicados con la etiqueta **<li>**. En el código 2 se puede ver cómo cambia la visualización usando <ul> o bien <ol>. También se ve cómo es posible anidar las listas.

```
<ul>
<li>Primer ítem</li>
<li>Segundo ítem</li>
<li>Tercer ítem</li>
</ul>
<ol>
<li>Primer ítem</li>
<li>Segundo ítem</li>
<li>Tercer ítem</li>
<ul>
<li>Tres uno</li>
<li>Tres dos</li>
</ul>
</ol>
```

- Primer ítem
- Segundo ítem
- Tercer ítem

1. Primer ítem
2. Segundo ítem
3. Tercer ítem
    - Tres uno
    - Tres dos

Código 3: Listas en HTML

```
<table border="1">
  <thead>
    <tr>
      <th>Mes</th>
      <th>Ingresos</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td>Total</td>
      <td>1800,00€</td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td>Enero</td>
      <td>1000,00€</td>
    </tr>
    <tr>
      <td>Febrero</td>
      <td>800,00€</td>
    </tr>
  </tbody>
</table>
```

Mes	Ingresos
Enero	1000,00€
Febrero	800,00€
Total	1800,00€

Código 2: Tablas en HTML. A la derecha, salida obtenida.

Las tablas. <table>

Vamos a intentar detenernos brevemente en las tablas, una de las etiquetas peor tratadas históricamente en la Web y que aún es usada por algunos desarrolladores no muy avezados o anclados en el pleistoceno informático para maquetar ¡¡cuando no es su propósito!!

Como su propio nombre indica, las tablas se deben utilizar única y exclusivamente para mostrar información de forma tabulada, es decir, organizada por filas y columnas.

Una table se divide en filas (englobadas por la etiqueta <**tr**>), y cada fila se divide en celdas (delimitadas con la etiqueta <**td**>). `td` viene de "table data," y es el lugar donde se muestra el contenido de la celda, que pude ser texto, enlaces, imágenes, formularios e incluso otra tabla.

En el código 3 podemos ver una tabla completa, con la mayoría de las etiquetas posibles:

- **<table>** delimita la tabla
- **<thead>** delimita el encabezado de la tabla. Tiene la particularidad que cada celda no viene delimitada por <td> sino por <**th**>
- **<tfooter>** delimita el pie de la tabla.
- **<tbody>** delimita el cuerpo principal de la tabla, donde se encuentran los datos.

Si queremos que una celda ocupe más de dos columnas habrá que utilizar el atributo `colspan="n"` siendo n el número de columnas que queremos que ocupe.

```
<td colspan="2">Mira, ocupo dos columnas</td>
```

*Agrupando elementos y *

La etiqueta permite definir bloques de información, a modo de contenedor que agrupe otros elementos HTML. Su utilidad principal vendrá dada por la información semántica y organizativa que aporta, pues el navegador forzará a un salto de línea tras el

Sin embargo, cuando se usa conjuntamente con CSS permite aplicar estilos a grandes bloques de contenido, definir las dimensiones del bloque, sus propiedades de fuente, color, e incluso su posición en la página.

La etiqueta es un elemento que se denomina *inline*, es decir, que no produce salto de línea cuando se utiliza, y sirve como contenedor de texto dentro de un párrafo o un ítem de lista. Se suele utilizar combinado con CSS para aplicar estilos especiales a sólo ciertas palabras del texto.

Formularios

Los formularios son una parte del estándar HTML imprescindible para el desarrollo de Sistemas de Información Basados en Web. Los usuarios interactúan con estos componentes, introducen información y pulsan un botón de envío. El navegador recopila los datos y los envía al servidor para su procesamiento. Algunas veces, las modificaciones o los datos introducidos en el formulario desencadenan acciones Javascript que modifican la apariencia de la página.

Todo formulario HTML está enmarcado en un elemento `<form>`, y todo lo que hay dentor es interpretado como parte del formulario

```
<form>
…
</form>
```

Dentro de un formulario se puede incluír cualquier etiqueta HTML, pero lo más importante son aquellas relativas a su función inherente: recopilar información, a saber:

`<input type="checkbox"/>`	Casilla de verificación
`<input type="text"/>`	Cuadro de texto de una línea
`<input type="password"/>`	Cuadro de texto de una línea que oculta lo escrito
`<textarea>…</textarea>`	Cuadro de texto de varias líneas
`<input type="radio"/>`	Selección única entre varias opciones
`<input type="submit"/>`	Botón de envío del formulario
`<input type="image"/>`	Imagen que hace las veces de botón de envío
`<input type="reset"/>`	Botón que reinicia el formulario
`<input type="button"/>`	Botón que al pulsar permite desencadenar un script

`<select>`
`<option>…</option>`
…
`<select>`
	Lista en la que se pueden elegir una o más opciones

`<label>`	Es la etiqueta de un <input>
`<fieldset>`	Permite agrupar elementos
`<legend>`	Etiqueta o nombre para un fieldset enmarcado

Todos los elementos tienen varios atributos:

- `Name`. Identifica a la variable que almacenará el valor introducido
- `Value`. Permite dejar un valor por defecto o el texto del "submit"

Ejemplos:

```
<form>
<input type="radio" name="especialidad" value="IS">Ing. Software<br>
<input type="radio" name="especialidad" value="SI">Sist. Información<br>
</form>
```

Ing. Software
Sist. Información

Código 5 Ejemplo Input Radio

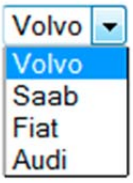
```
<form>
<h3>Aficiones</h3>
<input type="checkbox" name="aficion" value="Fútbol">Fútbol<br>
<input type="checkbox" name="aficion" value="Ajedrez">Ajedrez<br>
</form>
```

Aficiones

Fútbol
Ajedrez

Código 4 Ejemplo Input CheckBox

```
<form>
<form action="">
<select name="coche">
<option value="volvo">Volvo</option>
<option value="saab">Saab</option>
<option value="fiat">Fiat</option>
<option value="audi">Audi</option>
</select>
</form></form>
```



Código 7 Ejemplo Lista

```
<form action="">
<fieldset>
<legend>Datos personales:</legend>
<label>Nombre</label> <input type="text" size="30"><br>
<label>Apellidos</label> <input type="text" size="30"><br>
</fieldset>
```


```
</form>
```

Código 6 Ejemplo Fieldset y Label

Ejercicio 23. *En http://www.w3schools.com/html/html_forms.asp podrá usted complementar sus apuntes sobre los formularios HTML.*

Novedades en HTML5

HTML5 es la más reciente versión del estándar HTML. En el consorcio que lo ha desarrollado se encuentran AOL, Apple, Google, IBM, Microsoft, Mozilla, Nokia, Opera, y cientos de empresas

Las novedades más importantes que trajo bajo el brazo HTML5 fueron:

- El elemento `<canvas>` para dibujado 2D y 3D
- Reproducción nativa de `<video>` y `<audio>`
- Soporte para almacenamiento local
- Elementos específicos con semántica: `<article>`,`<footer>`,`<header>`,`<nav>`,`<section>`
- Nuevos controles: calendario, fecha, hora, e-mail, URL, etc.

Otro detalle interesante es que se puede insertar en una página HTML directamente código MathML o SVG.

Ejercicio 24. *En http://www.w3.org/TR/html5-diff/ hay un resumen de las principales novedades de HTML5 y sus diferencias con sus predecesores. Complete sus apuntes con lo que considere más relevante, y aplíquelo en el desarrollo de las prácticas de la asignatura.*

Algo también muy interesante es la desaparición de ciertas etiquetas que "invitaban" a generar un código despreciable, a saber: `<b>`, `<font>`, `<frame>`, `<center>` o `<big>`.

Uno de los detalles más interesantes y menos conocidos es la capacidad de usar almacenamiento local. Básicamente es como tener una base de datos en el lado cliente, para uso que anteriormente se dejaban en manos de cookies:

- Recuperar el estado de la aplicación al reiniciar el navegador
- Mantener datos en caché
- Guardar preferencias de usuario

Esta gestión de datos en el lado cliente se realiza mediante Javascript, accediendo al DOM de HTML5.

Obviamente, no todo es tan maravilloso, hay algunas pequeñas pegas, como puede ser la falta de un mecanismo de caducidad de la información (como tienen las cookies) o la imposibilidad de encriptar la información.

Finalmente, a falta de que usted complete sus apuntes con los elementos introducidos por el estándar HTML5 en sus apuntes, mostramos en la figura __ un gráfico tomado de www.html5doctor.com/semantics que le guiará a la hora de construir un sitio web:

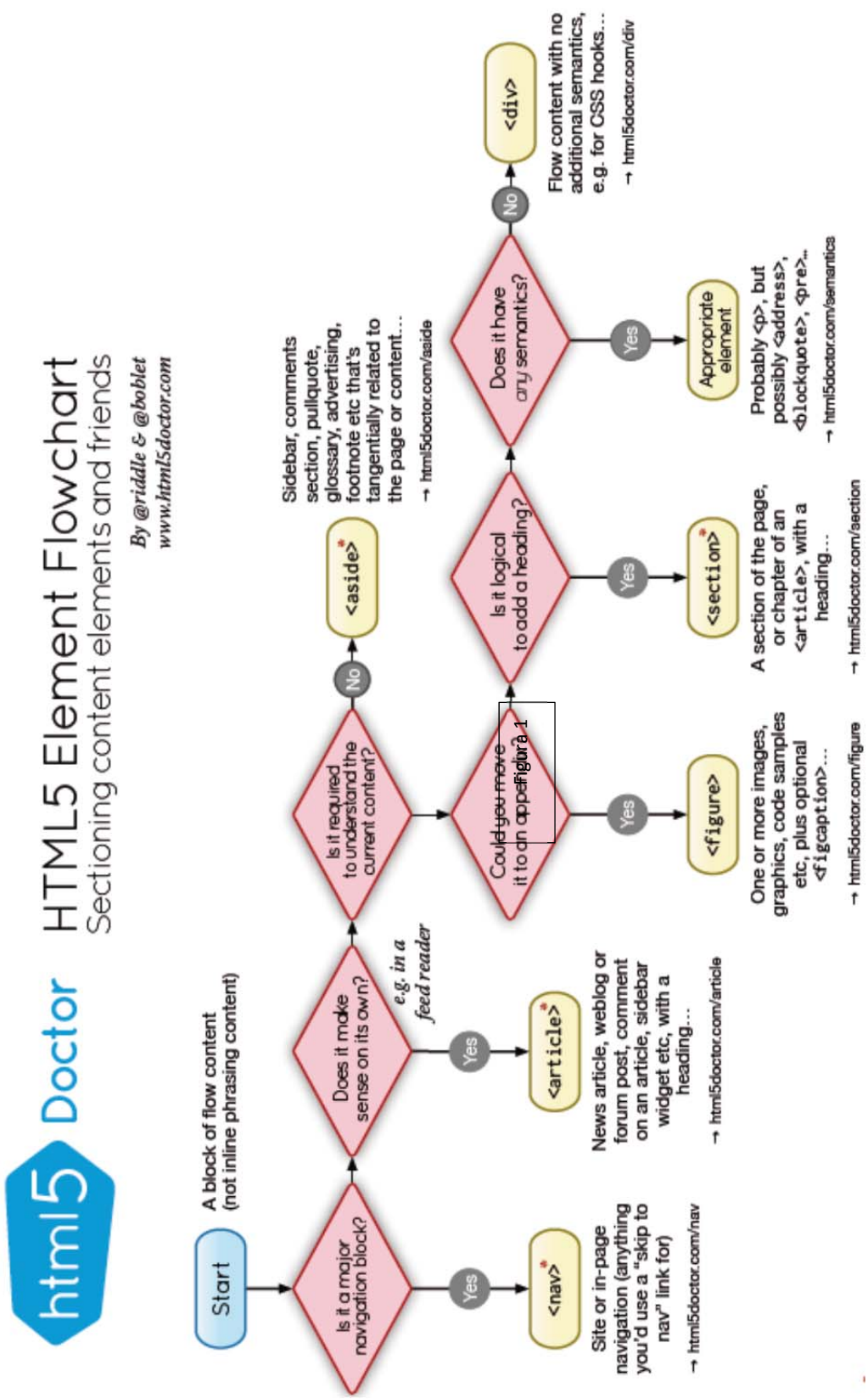


Ilustración 21: Diagrama de decisión sobre estructuras semánticas de HTML5

Las hojas de estilo se conocen por el nombre de su estándar, CSS (*Cascading Style Sheets*) y permiten definir la apariencia de los elementos de una página Web.

En el código HTML de una página hay normalmente una serie de directivas que indican al navegador que existe un estilo para mostrar dicho código HTML. Estas hojas de estilo se pueden aplicar de tres formas distintas:

- Mediante una hoja de estilo externa almacenada en un archivo independiente. Se indica mediante la etiqueta <link> dentro de la sección <head>:

```
<link rel="stylesheet" type="text/css" href="/estilo.css" />
```

- Mediante una hoja de estilo interna, incrustada en el propio documento HTML dentro de la sección <head>, como lo que nos encontramos en la página de inicio de la UGR:

```
<style type="text/css" media="all" >
  a.banner:hover{
   background-image:url(/img/banners/4Lcongresosugr_generico.png)
  }

 li.model-modern-normal_amarillo {
background-image:url(/img/banners/modern-normal_amarillo.png)
}
</style>
```

- Directamente dentro de la etiqueta de inicio de un elemento HTML, como también vemos en la página de inicio de UGR

```
<div id="fondo-dialog" style="display: none;"></div>
```

Ejercicio 25. *De las tres formas de aplicar estilo a HTML, ¿Cuál considera más correcta? ¿Por qué?*

Anatomía de las reglas CSS

Las hojas de estilo contienen sólo una cosa: reglas, y todas tienen la misma estructura:

```
selector { propiedad:valor }
```

Por ejemplo, si queremos que los encabezados de primer nivel tengan un tamaño de fuente de 12px y sean de color azul, pondríamos:

```
h1 { color:blue;font-size:12px;}
```

Aunque lo normal es escribirlo de una forma más legible:

```
h1 {
   color:blue;
   font-size:12px;
}
```

El selector identifica el tipo de contenido al que se desa dar formato. La propiedad identifica el tipo de formato que quiere aplicarse, y el valor establece un rango para dicha propiedad. Las propiedades se separan por ;.

También es posible crear una única instrucción de formato que afecte a varios elementos distintos. Por ejemplo, si queremos que todos los encabezados hasta nivel sean de color #24fe21 haríamos lo siguiente:

```
h1,h2,h3 {
    color: #24fe21;
}
```

Los comentarios se incluyen entre los símbolos `/*` y `*/`.

Los selectores de clase y de ID

¿Qué ocurre si sólo queremos aplicar un estilo a un encabezado de una sección muy concreta? Con las reglas anteriores se aplica a todos los h1 por igual, sin discriminar el contexto.

HTML y CSS proporcionan los selectores de clase y de ID, que permiten identificar elementos concretos de la página Web. Su sintaxis es como sigue:

```
<div class="menu">
 <ul class="listamenu" >
       <li id="seleccionado"> primero </li>
       <li> segundo </li>
 </ul>
</div>
```

De esta forma , podemos tener un CSS que especifique:

```
/* Selector de clase: se aplica a todos los elementos class="menu" */
    .menu {
     background: #111111; /* color de fondo gris oscuro */
      font-size: 12px;
    }

/* selector de clase: se aplica sólo a los <ul> de clase "listamenu"*/
    ul.listamenu {
     list-style-type: none;  /* sin marcador de lista · */
    }

/* CSS anidado:
se aplica sólo a los <li> contenidos en <ul> de clase "listamenu"*/
    ul.listamenu li {
     font-weight: bold; /* negrita */
    }

/* Selector de ID: se aplica al element que tenga ID="seleccionado"*/
    #seleccionado {
    font-size: 14px; /* letra 14px (en lugar de 12px del resto) */
    }
```

La diferencia entre `class` e `id` es que sólo puede haber un elemento en todo el documento HTML con un ID concreto, sin embargo, sí puede haber varios elementos de una misma clase.

En el caso de este ejemplo, si tenemos otras reglas en el CSS para los elementos <ul> en general, y hay conflicto con lo especificado para la clase listamenu, prevalece lo indicado para la clase.

Ejercicio 26. *Interprete el siguiente código. ¿De qué color aparecerán los elementos de la lista? ¿De qué tamaño?*

```
<head>
<style>
em+strong {color:red;}
.nivel1 {color:green;}
.nivel2 {color:blue;}
ul ul .nivel2 { font-size: x-small; }
ul ul li { font-size: x-small; color: red;}
</style>
</head>
<body>
<h1><em>Título</em>:<strong>Anidamiento y agrupamiento </strong>
de selectores </h1>
<ul>
     <li class=”nivel1”> Agrupamiento </li>
     <li class=”nivel1”> Anidamiento </li>
     <ul>
          <li class=”nivel2”> Anidamiento de hijos</li>
          <li class=”nivel2”> Anidamiento de adyacentes</li>
          <li > Anidamiento común</li>
     </ul>
</ul>
</body>
```

Buenas reglas a la hora de escribir código CSS

- Los selectores se nombran en minúsculas
- El nombre de los selectores debe ser específico y claro
- El nombre de las clases no debe describir una característica visual
- Los nombres de clases e identificadores deben seguir una visión semántica
- Separar palabras mediante guiones o mayúsculas
- Agrupar las reglas según su selector
- Estructurar visualmente los atributos

El modelo de cajas

Para entender los estilos CSS, lo más adecuado es pensar en cualquier elemento como una caja, cuyas dimensiones y atributos pueden ser controlados para producir gran cantidad de efectos. Las propiedades de la caja se muestran en la figura ___:

Todas las propiedades mostradas se pueden adaptar y modificar para prácticamente cualquier etiqueta HTML, si bien son aquellas que delimitan bloques en las que más se usan (div, span, p, h, img, etc.).

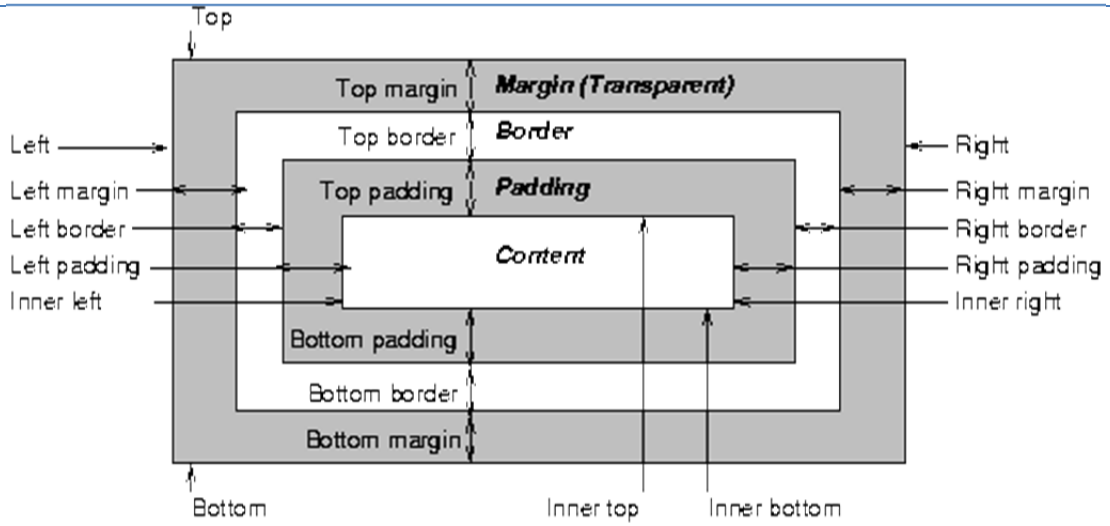


Ilustración 22: Modelo de caja CSS

Generalmente, las páginas HTML usan un modelo de diseño flotante: el contenido fluye desde la parte superior de la ventana del navegador a la parte inferior. El navegador coloca los elementos correlativamente para mostrarlos consecutivamente. Al usar propiedades CSS, los elementos "salen" de ese flujo y se organizan según las distintas reglas definidas.

La propiedad float permite extraer el elemento al que se aplica del flujo normal de HTML, y el resto de contenidos que se muestran posteriormente en el código "flotan" o se distribuyen alrededor de ese elemento flotante.

```
<!DOCTYPE html>
<html>
<head>
<style>
.cajaIzda
{
float:left;
width:110px;
height:90px;
margin:5px;
border: 1px;
background: #000;
color: #fff;
}
</style>
</head>

<body>
<h3>Ejemplo float</h3>
<p> Lo que va antes del div que flota a la izquierda
no se ve afectado por las propiedades de este div.
<div class="cajaIzda">
<p> Caja flotante </p>
</div>
<p>El texto que va después del div fluye a la derecha
de la caja que está float:left;.</p>

</body>
</html>
```

Ejemplo float

Lo que va antes del div que flota a la izquierda no se ve afectado por las propiedades de este div.

Caja flotante

El texto que va después del div fluye a la derecha de la caja que está float:left;.

Ilustración 23: Ejemplo de elemento flotante.

Otra posibilidad para posicionar las cajas es colocarlas en una posición fija. Para ello, se indica la posición `absolute` y se marcan las propiedades de posición `top, left, bottom` y `right.`

Pasamos a mostrar algunos atributos de los más usados en CSS y que pueden servir para la elaboración de las prácticas

Atributo	Descripción	Valores
Atributos de fuentes		
color	Indica el color del texto	#RRGGBB
font-size	Tamaño de la fuente	Unidades (px, em) xx-small, x-small, large, etc.
font-family	Familia de la tipografía	serif | sans-serif | monospace |cursive | fantasy | cualquier otra
font-weight	Anchura de los caracteres	Normal | bold | bolder | lighter 400 | 700 | …
font-style	Estilo de la fuente	normal |italic | oblique
Atributos de párrafos		
line-height	Espaciado entre líneas	normal | unidades (px, em)
text-decoration	Decoración del texto	none |underline | overline | line-through
text-align	Alineación del texto	left | right |center |justify
text-indent	Sangrado de párrafo	unidades (px, em)
text-transform	Transformar el texto	capitalize | uppercase | lowercase | none
Atributos de fondo		
background-color	Color de fondo	#RRGGBB
Background-image	Imagen de fondo	URL de la imagen
Atributos de tablas		
caption-side	Posición del título	Top | bottom
border-spacing	Espaciado entre los bordes	unidades (px, em)
Atributos de visibilidad		
overflow	Comportamiento del contenido si se desborda en la caja	visible | hidden | scroll | auto
visibility	Visibilidad de las cajas	visible | hidden | collapse
display	Muestra una caja con diferentes estilos	inline | block | none | list-item |table | inherit | inline-table | etc…
Atributos de listas		
list-style-type	Estilo del marcador	Disc | circle | square | decimal | lower-roman |upper-roman | lower-alpha | none…
list-style-image	Imagen aplicable a los elementos de la lista	url("http:///...")

Precedencia de estilos

La precedencia de estilos es una manera de indicar que un estilo definido prevalece por encima de otro definido en el mismo o en otro archivo CSS. El criterio general es que prevalece el definido en la regla más específica.

Además de este criterio, se puede obligar que un atributo se aplique por encima del resto de reglas, utilizando la declaración `!important` justo antes del punto y coma que cierra la regla:

```
p {
 background-color: red!important;
 background-color: yellow;
}
```

Ejercicio 27. *En la red hay una ingente cantidad de material para aprender CSS, desde lo más básico hasta avanzados efectos disponibles en algunos navegadores. Complemente sus apuntes con el curso del W3C school: http://www.w3schools.com/css/*

Javascript

JavaScript es un lenguaje de programación que permite la ejecución de scripts en el lado cliente de nuestro sistema Web. Apareció en 1995, soportado por el navegador Netscape Navigator 2, e Internet Explorer empezó a tolerarlo a partir de IE3.

¿Qué se puede hacer con Javascript?

- Insertar o modificar contenido de forma dinámica en una página web. Se puede modificar tanto el contenido HTML como las propiedades CSS.
- Recoger información del navegador y equipo cliente.
- Reaccionar a eventos generados en el navegador
- Comunicarse con el servidor sin necesidad de recargar la página Web.

Toda secuencia de comandos se encuentra dentro un bloque `<script>` insertado en el documento HTML:

- En la sección <body>, en cuyo caso se ejecuta el script en cuanto se cargue el código.
- En la sección <head>, que hará que el script se ejecute antes de que el navegador procese parte alguna del código HTML.

Ejemplo:

```
<!DOCTYPE html>
<html>
<head>
       <title> Ejemplo </title>
</head>
<body>
 <h1> Ejemplo de JavaScript.</h1>
 <script type=”text/javascript”>
       alert(“Esto es una ventana emergente con mensaje.”);
 </script>
</body>
</html>
```

Si se mostrara en un navegador el código anterior, el script muestra un mensaje por pantalla cuya ventana se cerrará al pulsar el botón “Aceptar”, que viene por defecto en la función `alert`.

JavaScript reconoce como fin de sentencia el salto de línea o el punto y coma , lo que hace opcional este elemento tan típico de C o Java.

Vamos a comentar algunas nociones básicas de JavaScript, que deberán ser complementadas con el trabajo autónomo en prácticas y la búsqueda de bibliografía, abundante por cierto tanto en la red como en la biblioteca del centro.

Declaración de variables

Para crear una variable en JavaScript se utiliza la palabra clave `var` seguida del nombre de la variable. Hay que destacar que JavaScript, al igual que C, es sensible a las mayúsculas. Por supuesto, es posible inicializarlas a la vez que se declaran, pero nótese que no es un lenguaje fuertemente tipado, que incluso dispone de operador de concatenación de texto (el símbolo +)

```
var ejemplo=”Texto de ejemplo”;
var concatenacion=ejemplo+” concatenado”;
```

Veamos un ejemplo algo más complejo, en el que se muestra la fecha actual:

```
<!DOCTYPE html>
<html>
<head>
       <title> Ejemplo JavaScript 2 </title>
</head>
<body>
 <h1> Ejemplo de JavaScript - Fecha.</h1>
 <p>
 <script type=”text/javascript”>
       var fechaActual = new Date();  // 1
       var mensaje = “Hoy es “;       // 2
       document.write (mensaje + fechaActual.toDateString()); // 3
 </script>
 </p>
</body>
</html>
```

Lo más reseñable es que en la línea etiquetada como // 3, JavaScript incluye en el HTML, en la posición del script, la cadena de texto formada por la concatenación del mensaje “Hoy es “ con el string de la fecha actual, obtenido mediante el método `toDateString()`. Fíjese que esto ya nos indica que JavaScript es orientado a objetos.

Funciones

Las funciones JavaScript suelen declararse en la sección <head> de la página, y luego son invocadas desde dentro del contenido <body>, por ejemplo

```
<!DOCTYPE html>
<html>
<head>
       <title> Ejemplo JavaScript 2 </title>
<script>
       function getMensajeConFecha(mensaje) {
             var fechaActual = new Date();
             return mensaje + fechaActual.toDateString();
       }
</head>
<body>
 <h1> Ejemplo de JavaScript - Fecha.</h1>
 <p>
 <script type=”text/javascript”>
     // Llamamos a la función definida en el head
       document.write (getMensajeFecha(“Hoy es “));
 </script>
 </p>
</body>
</html>
```

Otra opción, mucho más elegante, es guardar el código de las funciones en un archivo externo, y referenciarlo en la sección head:

```
<script src=”funciones.js”></script>
```

HTML Dinámico y Javascript

A finales de los 90, apareció el concepto de HTML dinámico, una nueva forma de pensar en el diseño y la ejecución de una página Web. Con el modelo de objetos de un documento HTML (DOM), cada uno de los elementos de la página son objetos jerarquizados que pueden ser accedidos, y sus propiedades modificadas, mediante el uso de JavaScript.

Antes de poder manipular un objeto en la página Web, es fundamental poder identificarlo de forma unívoca, y para ello lo mejor es usar el atributo `id` que ya vimos en el apartado de CSS alguna de sus utilidades.

```
<h1 id="encabezado1"> Bienvenidos </h1>
```

Una vez tenemos con id el elemento, se puede obtener el objeto de dicho elemento con JavaScript utilizando el método `document.getElementById()`. Básicamente, `document` es un objeto que representa al documento HTML completo, es una variable global que tiene el navegador y siempre está disponible.

En nuestro caso, podríamos modificar el contenido del elemento h1 anterior con las siguientes líneas:

```
var objetoTitulo = document.getElementById("encabezado1");
objetoTitulo.innerHTML = "Bienvenidos todos";
```

Todos los objetos HTML comparten algunos atributos comunes, como los mostrados en la siguiente tabla:

Propiedad	Descripción
`className`	Permite recuperar o establecer el atributo `class`, utilizado para aplicar estilos CSS.
`innerHTML`	Permite leer o cambiar el HTML del interior de un elemento.
`parentElement`	Proporciona el objeto HTML en el que está contenido el elemento.
`style`	Aúna todos los atributos CSS del elemento HTML. Devuelve un objeto con las propiedades CSS como atributo.
`tagName`	Proporciona el nombre del elemento HTML (p.ej. si es un `<img>` devuelve `img`)

El uso del DOM de HTML5 permite conseguir animaciones y efectos muy conseguidos con simples rutinas.

Eventos

Los eventos son notificaciones que envía un elemento HTML cuando ocurre algo. Por ejemplo, JavaScript asigna a todo elemento <a> un evento llamado `onmouseover`. Cuando un visitante pasa el puntero del ratón sobre dicho elemento, se inicia el evento:

```
<p> En un párrafo <a href="#" onmouseover="alert('Hey!')">ten cuidado</a></p>
```

En la siguiente tabla mostramos los eventos más utilizados y los elementos HTML a los que se aplican:

Evento	Descripcióin	Elementos
`onClick`	Se inicia al hacer click	Casi todos
`onMouseOver`	Se inicia al pasar el puntero sobre el elemento	Casi todos
`onMouseOut`	Se inicia al alejar el puntero del elemento	Casi todos
`onKeyUp`	Se inicia al soltar una tecla pulsada	`<select>,<input>, <textarea>,<a> <button>`
`onFocus`	Se inicia cuando un control recibe el foco	`<select>,<input>, <textarea>,<a> <button>`
`onChange`	Se inicia cuando cambia un valor en un control de entrada	`<select>,<input type=”text”>, <textarea>`
`onLoad`	Se inicia cuando el navegador termina de cargar una página o elemento	`<img>, <body>`

jQuery

El propósito de jQuery es facilitar el uso de JavaScript en nuestro sistema web, ya que simplifica algunas cosas relativamente complejas de JavaScript como las llamadas AJAX o la manipulación del DOM.

Lo primero que hay que hacer es descargar una versión de jQuery, bien la de producción (comprimida) o la de desarrollo, ambas descargables desde jQuery.com. A continuación, ya podemos incluir la librería en nuestro HTML:

```
<head>
<script src=”jquery-1.10.2.min.js”></script>
</head>
```

Otra opción, para no descargarla, es incluirla de forma remota desde un servidor de Google o de Microsoft:

```
<script
src=”http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js”>
</script>
```

Sintaxis

Con jQuery se seleccionan (query) elementos HTML y se realizan acciones sobre ellos. La sintaxis básica es:

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

Donde encontramos

- El signo $, que define un acceso mediante jQuery
- El (`selector`) que busca el elemento HTML
- La `action()` que hay que realizar sobre el elemento.

Por ejemplo:

- `$(this).hide()` oculta el elemento actual
- `$("p").hide()` oculta todos los elementos <p>
- `$(".test").hide()` oculta los elementos de clase "test"
- `$("#test").hide()` oculta los elementos de id "test".

Otros selectores pueden ser:

Syntax	Description
$("*")	Seleciona todos los elementos
$(this)	Selecciona el element HTML actual
$("p.intro")	Selecciona los elementos <p> de class="intro"
$("p:first")	Selecciona el primer elemento <p>
$("ul li:first")	Selecciona el primer elemento del primer
$("ul li:first-child")	Selecciona el primer elemento de todos los
$("[href]")	Selecciona todos los elementos con un atributo href
$("a[target='_blank']")	Selecciona aquellos elementos <a> con un atributo target con valor "_blank"
$("a[target!='_blank']")	Selecciona aquellos elementos <a> con un atributo target con valor distinto a "_blank"
$(":button")	Selecciona los elementos <button> e <input> de tipo type="button"
$("tr:even")	Selecciona las líneas de tabla impares
$("tr:odd")	Selecciona las líneas de tabla pares

El evento document ready

Todo código jQuery debe tener la siguiente instrucción:

```
$(document).ready(function(){

   // los métodos jQuery van aquí...

});
```

Esta instrucción indica que el código jQuery se ejecute cuando el código HTML se ha cargado completamente y el árbol DOM se ha generado. Esto debe ser así ya que jQuery trabaja con los elementos que hay en la página referenciados a través del DOM. Si alguno no se ha cargado, obtendríamos un error.

Las líneas anteriores se pueden resumir en

```
$(function(){
   // los métodos jQuery van aquí...
});
```

Gestión de eventos

Los eventos son acciones del usuario a las que puede responder la página web, como pasar el ratón por encima de un elemento, seleccionar una opción de un radio button o hacer clic sobre un elemento. Algunos de los eventos del DOM de HTML son los siguientes:

Eventos de ratón	Eventos de teclado	Eventos de formulario	Eventos de documento / ventana
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

La mayoría de los eventos del DOM tienen un método equivalente en jQuery. Por ejemplo, para asignar un evento clic a todos los párrafos de una página podríamos escribir:

```
$(“p”).click();
```

Pero claro, ahí no estamos haciendo nada, o mejor dicho, no decimos qué hacer cuando se haga clic. Lo suyo es dotar de una función al evento, para reaccionar a la acción del usuario:

```
$("p").click(function(){
        $(this).hide(1000);
});
```

En este caso, cuando se hace clic en un párrafo, se oculta, tardando 1000 milisegundos en ocultarse.

Hay dos funciones jQuery muy interesantes que merece la pena detenerse en ellas. Son `bind/unbind` y `toggle`.

`bind()/unbind()`

`bind()` permite asociar a un elemento un número ilimitado de eventos, o una misma función a varios elementos distintos. Podríamos decir que permite una mejor estructuración del código, mediante la definición de las funciones separada de su vinculación a eventos.

```
<head>
<script src=”jquery-1.10.2.js”></script>
<script>
 $(document).ready(function() {
             var accion=function() {
                   //… lo que haya que hacer
             }

             $(“p”).bind(“click mouseenter”, accion);
});
</script>
</head>
```

toggle()

`toggle()` es otro método muy usado, ya que se le pasan dos funciones y ejecuta alternativamente la primera o la segunda función según sea el primer clic (o todos los impares) o el segundo (o todos los pares).

```
<head>
<script src=”jquery-1.10.2.js”></script>
<script>
 $(document).ready(function() {
            var ponerrojo=function() {
                  $(this).css(“color”,”#ff0000”);
            }
            var ponerazul=function() {
                  $(this).css(“color”,”#0000ff”);
            }
            $(“p”).toggle(ponerrojo,ponerazul);
});
</script>
</head>
```

Manipulación del DOM (Document Object Model)

El DOM (Document Object Model) de W3C es una interfaz independiente de plataforma y lenguaje que permite a los programas y scripts acceder dinámicamente y modificar el contenido, estructura y estilo de un documento.

jQuery dispone tres sencillos métodos para manipular el dom:

- `text()`, que cambia o devuelve el contenido textual de los elementos seleccionados.
- `html()`, que cambia o devuelve el contenido de los elementos seleccionados (incluyendo las etiquetas HTML)
- `val()`, que establece o devuelve el valor de los campos de formulario

Ejemplo:

```
$("#btn1").click(function(){
  alert("Text: " + $("#test").text());
});
$("#btn2").click(function(){
  alert("HTML: " + $("#test").html());
});
```

Los tres métodos pueden tener como parámetro una función de callback, es decir, un comportamiento más complejo con dos parámetros: el índice del elemento actual en la lista de todos los seleccionados y el valor original, devolviendo la función la cadena que se desee:

Ejemplo.

```
$("#btn2").click(function(){
  $("#test2").html(function(i,origText){
    return "Valor anterior: " + origText + " Nuevo valor: <b> Hola!</b>
    (índice: " + i + ")";
  });
});
```

Otra función interesante es `attr()` que permite consultar o modificar el valor de un atributo. Esto nos permite por ejemplo modificar el destino de un enlace:

```
$("button").click(function(){
  $("#etsiit").attr("href","http://etsiit.ugr.es");
});
```

E incluso modificar varios atributos a la vez:

```
$("button").click(function(){
   $("#etsiit").attr( {
            "href":"http://etsiit.ugr.es",
            “title”:”ETS Ings Informática y Teleco”
        } );
});
```

Inclusión/eliminación de elementos

Con jQuery es fácil añadir/eliminar elementos o contenido a la página HTML, para lo que se dispone de cuatro métodos:

- `append()` que inserta contenido al final de los elementos seleccionados.
- `prepend()` que inserta contenido al principio de los elementos seleccionados
- `after ()` inserta contenido tras los elementos seleccionados
- `before()` inserta contenido antes del elemento seleccionados.

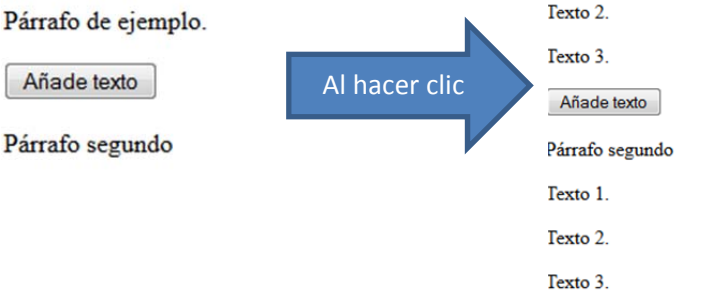
Ejemplo:

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js">
</script>
<script>
function appendText() {
 var txt1="<p>Texto 1.</p>";         // variable HTML párrafo
 var txt2=$("<p></p>").text("Texto 2."); // Inserta texto con jQuery
 var txt3=document.createElement("p");  // Crear párrafo con DOM

 txt3.innerHTML="Texto 3.";
 $("p").append(txt1,txt2,txt3); // Añade los elementos uno tras otro
}
</script>
</head>
<body>

      <p>Párrafo de ejemplo.</p>
      <button onclick="appendText()">Añade texto</button>
      <p>Párrafo segundo.</p>

</body>
</html>
```




```
<p>Párrafo de ejemplo.
   <p>Texto 1.</p>
   <p>Texto 2.</p>
   <p>Texto 3.</p>
</p>
<button  onclick="appendText()">
Añade texto </button>
<p>Párrafo Segundo
   <p>Texto 1.</p>
   <p>Texto 2.</p>
   <p>Texto 3.</p>
</p>
```

jQuery también permite eliminar elementos dinámicamente del DOM de la página, mediante dos funciones:

- `remove()` que elimina el objeto seleccionado (y sus hijos)
- `empty()` que elimina los hijos del elemento seleccionado.

Por lo que podemos inferir que no es lo mismo

```
$("#div1").remove();
```

que

```
$("#div1").empty();
```

o

```
$("p").remove(".italic");
```

jQuery y CSS

Con jQuery es trivial manipular el CSS de los elementos mediante los siguientes métodos:

- addClass() que añade una o más clases al elemento seleccionado
- removeClass() que elimina una o más clases del elemento seleccionado
- toggleClass() que alterna añadir/eliminar una o varias clases de un elemento
- css() devuelve o establece un atributo de estilo

Por ejemplo:

```
$("button").click(function(){
  $("h1,h2,p").addClass("blue");
  $("div").addClass("important");
  $("h3").addClass("important blue");
});
```

Para obtener el valor de una propiedad CSS utilizamos el método css()

```
$("p").css("background-color");
```

Que acompañado de un segundo argumento o una lista de argumentos sirve para dar valor/es a dicha propiedad:

```
$("p").css("background-color",”#bbbbbb”);
$("p").css({"background-color":”#bbbbbb”, “font-size”:”200%”);
```

AJAX

AJAX no es un lenguaje de programación, sino que es la tecnología que permite intercambiar datos con un servidor, como partes de una página web, sin recargar la página completa. Su acrónimo viene de *Asynchronous JavaScript and XML*. Ejemplos de uso de AJAX podemos encontrar hoy día en casi todas las webs: Gmail, youtube, Facebook etc.

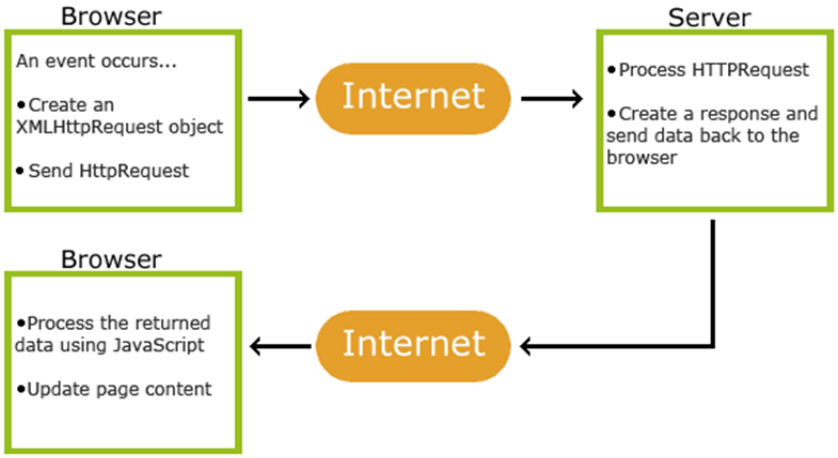


Ilustración 24: : Esquema de la gestión de una llamada AJAX

AJAX está basado en estándares de Internet, ya que usa una combinación de:

- Objetos XMLHttpRequest para intercambiar datos asíncronamente con el servidor
- JavaScript/DOM para mostrar o interactuar con la información
- CSS para dar formato a los datos
- XML para formatear los datos para su transferencia.

Ejemplo de AJAX: las sugerencias de búsqueda de Google

La clave de todo AJAX es el objeto XMLHttpRequest, soportado por todos los navegadores modernos (IE7+, Firefox, Chrome, Safari y Opera):

```
xmlhttp=new XMLHttpRequest();
```

Este objeto `xmlhttp` permitirá enviar una petición al servidor mediante los métodos `open()` y `send()`:

Método	Descripción
`open(method,url,async)`	Especifica la petición, la URL y si la petición se debe realizar de forma asíncrona o no: - *method*: GET o POST - *url*: localización del archive en el servidor - *async*: true (asíncrono) o false (síncrono)
`send(string)`	Envía la petición al servidor. - *string*: Usado solo para las peticiones POST

Quizá la mejor forma es verlo con un ejemplo, sencillo, que nos permite cambiar un texto sin necesidad de recargar la página al hacer clic en un botón:

```
<!DOCTYPE html>
<html>
<head>
<script>
   function loadXMLDoc() {
    var xmlhttp;
    xmlhttp=new XMLHttpRequest();
    xmlhttp.onreadystatechange=function()  {
       if (xmlhttp.readyState==4 && xmlhttp.status==200)    {
        document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
       }
     }
     xmlhttp.open("GET","nuevo_texto.txt",true);
     xmlhttp.send();
   }
</script>
</head>
<body>

<div id="myDiv">
<h2>Este texto es lo que vamos a cambiar</h2>
</div>

<button type="button" onclick="loadXMLDoc()">Cambiar</button>

</body>
</html>
```

Código 8: Llamada AJAX básica

La clave de todo está en la línea `xmlhttp.open("GET","nuevo_texto.txt",true)`. El valor *true* hace que el cambio de contenido HTML del div "myDiv" no se realice hasta que no obtengamos la respuesta del servidor. Por eso, la función que efectúa el cambio está asignada al evento *onreadystatechange* del objeto *xmlhttp*, pero no se bloquea la ejecución del script en ningún momento.

Si pusiéramos el valor false en el parámetro de sincronía, tendríamos que poner la llamada que modifica el código HTML justo después del send, y ésta sería bloqueante:

```
xmlhttp.send();
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

La respuesta del servidor

Para obtener la respuesta del servidor podemos usar dos atributos del objeto `xmlhttp`:

- `responseText`, que obtiene la respuesta como una cadena
- `responseXML`, que obtiene la respuesta como un XML

El `reponseText` hemos visto en el ejemplo del código 8 que se usa para tomar los datos tal cual llegan del servidor. Si obtenemos del servidor un documento XML, y queremos *parsearlo* como un objeto XML, entonces hay que usar `responseXML`, y utilizar métodos y atributos de manipulación del árbol XML:

```
xmlDoc=xmlhttp.responseXML;
txt="";
x=xmlDoc.getElementsByTagName("ARTIST");
for (i=0;i<x.length;i++)   {
  txt=txt + x[i].childNodes[0].nodeValue + "<br>";
  }
document.getElementById("myDiv").innerHTML=txt;
```

El evento `onreadystatechange`

Cuando se envía una petición al servidor, lo normal es efectuar una serie de acciones en función de la respuesta obtenida. El evento *onreadystatechange* se dispara cada vez que la propiedad *readyState* cambia. En realidad, hay tres propiedades o atributos del objeto XMLHttpRequest que son muy importantes:

Propiedad	Descripción
onreadystatechange	Guarda una function (o el nombre de la function) que se llama automáticamente cada vez que la propiedad readyState cambia
readyState	Almacena el estado de XMLHttpRequest con alguno de estos valores: 0: petición no iniciada 1: conexión con el servidor establecida 2: petición recibida 3: petición procesada 4: petición finalizada y respuesta preparada.
status	200: "OK" 404: Página no encontrada

JSON

JSON es JavaScript Object Notation, y es una sintaxis para el almacenamiento e intercambio de información textual.

JSON es independiente del lenguaje, es decir, que aunque use la sintaxis JavaScript para definir los objetos, existen librerías JSON para diversos lenguajes de programación.

En lugar de usar un *parser*, como en XML, se puede usar la función `eval()` nativa de JavaScript para generar los objetos a partir del texto JSON.

Vamos a adentrarnos un poco en JSON con un ejemplo:

```
<script>
    var JSONObject= {
        "name":"John Johnson",
        "street":"Oslo West 555",
        "age":33,
        "phone":"555 1234567"
    };
    document.getElementById("jname").innerHTML=JSONObject.name;
    document.getElementById("jage").innerHTML=JSONObject.age;
    document.getElementById("jstreet").innerHTML=JSONObject.street;
    document.getElementById("jphone").innerHTML=JSONObject.phone;
</script>
```

JSON en cierto modo puede considerarse parecido a XML porque es texto plano, legible por un humano, organizado jerárquicamente y puede ser *parseado* por JavaScript y transmitido usando AJAX.

Pero también es distinto a XML ya que no tiene metaetiquetas, es más breve en su definición, utiliza arrays para describir la información y puede ser *parseado* con la función eval() de JavaScript, sin palabras reservadas.

En el contexto de aplicaciones AJAX, JSON es más rápido y fácil que SML, ya que tan sólo hay que recuperar la cadena y evaluarla, mientras que con XML hay que usar las funciones del DOM de XML para navegar por el documento recuperado.

Sintaxis JSON

Las reglas sintácticas de JSON son un subconjunto de las de JavaScript:

- los datos están en pares nombre/valor
  ```
  “nombre”: “Javier”
  ```
- los datos se separan por comas
- las llaves engloban objetos
  ```
  { “nombre”: “Javier”, “apellidos”:”Melero” }
  ```
- los corchetes engloban arrays
  ```
  {
  “profesores”: [
   { “nombre”:”Eladio”, “apellidos”:”Garví”},
   { “nombre”: “Javier”, “apellidos”:”Melero” }
  ]
  }
  ```

La extensión de los archivos JSON es .json y su tipo MIME es “application/json”.

Convertir de objeto JSON a cadena

Suponiendo que recuperamos un archivo JSON con el texto:

```
{
“profesores”: [
 { “nombre”:”Eladio”, “apellidos”:”Garví”},
 { “nombre”: “Javier”, “apellidos”:”Melero” }
]
}
```

```
var txt= xmlhttp.responseText;
```

Podremos convertirlo a objetos JavaScript con la sentencia:

```
var profes=eval(“(“+txt+”)”);
```

Y modificar el HTML con el manejo habitual de JavaScript:

```
document.getElementById("fname").innerHTML = profes.profesores[0].nombre;
document.getElementById("lname").innerHTML =
profes.profesores[0].apellidos;
```

TECNOLOGÍAS WEB DE SERVIDOR

Desde su creación, la Web ha evolucionado desde un entorno de documentos hipermedia estáticos hasta una plataforma capaz de ejecutar complejas aplicaciones, como las aplicaciones de comercio electrónico. En este tipo de sistemas Web, las páginas se componen dinámicamente extrayendo el contenido de un cierto origen de datos y construyendo el resultado final en tiempo real.

Mediante la ejecución de programas o scripts en el servidor se puede:

- editar, cambiar o añadir cualquier contenido dinámicamente
- responder a peticiones o datos enviados por el usuario mediante formularios HTML
- acceder a bases de datos y devolver el resultado al navegador
- personalizar una página Web según los gustos individuales del usuario
- proporcionar una mayor seguridad ya que el código no es accesible desde el navegador.

Tradicionalmente se identifican tres capas en el desarrollo de entornos clientes/servidor, como son los sistemas Web:

- **Capa de presentación o vista.** Es la que ve el usuario, y se centra en el formateo de la información enviada por el servidor. Se ubica en el lado cliente, aunque parte de ella se puede preparar en el servidor.
- **Capa de negocio o controlador.** Es la que conoce y gestiona las funcionalidades del sistema. Es donde se reciben las peticiones del usuario y se reciben las respuestas.
- **Capa de datos o modelo.** Es la capa donde residen los datos y la encargada de acceder a los mismos.

El diseño de cada una de las capas influye a la hora de seleccionar uno u otro modelo de programación en la Web. Es posible clasificar las distintas aproximaciones en función del tamaño de los componentes o de la naturaleza del servicio ofrecido.

- **Según el tamaño de los componentes.** Se habla de configuraciones *fat client/thin server* cuando el mayor peso de la aplicación se ejecuta en el cliente, siendo el servidor un mero almacenador de datos. Las configuraciones *fat server/thin client* la funcionalidad asociada al cliente es meramente mostrar la información que gestiona el servidor.
- **Según la naturaleza del servicio.** Podemos encontrarnos con servidores de ficheros, servidores de bases de datos, servidores de transacción, servidores de objetos o simples servidores web. Cada tipo requiere una arquitectura distinta.

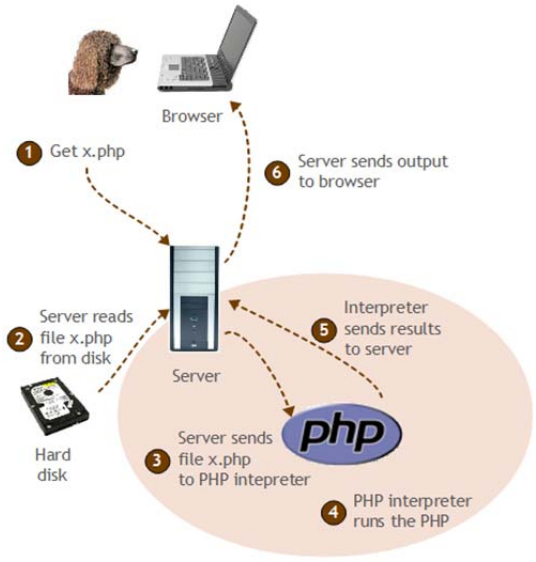


Ilustración 25: El proceso de generación de páginas dinámicas.

Existen múltiples alternativas a la hora de ejecutar código en el servidor como pueden ser los *lenguajes de scripting* (PHP, ASP, Perl, Python o JSP), cuya característica principal es que le código es interpretado y se intercala con una plantilla de código HTML. Otra posibilidad es el uso de enlaces a programas y componentes ejecutables (CGI, JSP, Servlets) de tal forma que el código ejecutado está almacenado en elementos precompilados y ejecutados de forma independiente.

CGI, Common Gateway Interface

Una de las primeras soluciones a la creación de contenido dinámico en la Web es el uso del CGI (1997), que es un estándar para la ejecución de aplicaciones en el servidor que permite la transmisión de información del cliente hacia este ejecutable instalado en el servidor web.

Cada vez que se recibe una petición de usuario se genera un proceso que ejecuta la aplicación solicitada, que normalmente estará en el directorio cgi-bin del servidor. De hecho, al incluir cgi-bin en la URL dice al servidor que es un ejecutable y no un script PHP o una página HTML estática.

De hecho, a Apache hay que decirle que cargue el módulo de ejecución de CGIs y dónde se encuentran estos:

```
LoadModule cgi_module modules/mod_cgi.so
ScriptAlias /cgi-bin/ /usr/local/apache2/cgi-bin/
```

Para programar CGI se suele usar PERL, por su simplicidad, aunque nada impide que se desarrolle el programa en C, C++ o Java. Ejemplo de programa CGI simple:

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
print "Hello, World.";
```

PHP

PHP (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML. Se creó en 1994 y se ha convertido en un estándar de facto para la creación de páginas web dinámicas.

Bien, pero ¿qué significa realmente? Un ejemplo nos aclarará las cosas:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <title>Ejemplo</title>
   </head>
   <body>
      <?php
         echo "¡Hola, soy un script de PHP!";
      ?>
   </body>
</html>
```

El código de PHP está encerrado entre las etiquetas especiales de comienzo y final <?php y ?> que permiten entrar y salir del "modo PHP".

Para su correcto funcionamiento es necesario que el servidor web (Apache, IIS, AOL, etc.) tenga instaladas las librerías PHP.

Una de las características destacables es su capacidad de conexión con la mayoría de servicios de bases de datos como MySQL, PostgreSQL, Oracle, etc.

Principales características

Las variables de PHP se identifican con el símbolo $ seguido del nombre de una variable, y no necesitan ser declaradas explícitamente. Tampoco tienen un tipo definido hasta que no se les asigna un valor, y el tipo puede cambiar si se asigna un valor de otro tipo:

```
$var=15; // $var es entera
$var=”Otra cosa” //var es un string
```

PHP es sensible a mayúsculas, por lo que `$variable` y `$Variable` no son lo mismo.

Las constantes se definen mediante la función `define()`, y a diferencia de las variables, no les precede el símbolo $

```
define(“PI”, 3.141592);
echo PI;
```

Es posible consultar mediante `isset($var)` si la variable tiene un valor no nulo, o mediante `empty($var)` si tiene algún valor.

Hay variables reservadas que automáticamente toman el valor del entorno de ejecución o de la petición HTTP, como pueden ser:

```
<?php
if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE) {
   echo 'Está usando Internet Explorer.<br />';
}
?>
```

Otras variables globales interesantes son $_GET , $_POST, $_FILES, $_COOKIE, $_SESSION, $_REQUEST o $_ENV. Permiten accede a elementos de la petición HTTP o a variables de entorno del sistema.

Podríamos extendernos en este apartado hasta configurar un tutorial completo, pero la red y la biblioteca están repletas de cursos, manuales y tutoriales, por lo que no nos extenderemos más y remitimos al alumno a las referencias bibliográficas de la asignatura.

Include() y require()

Es posible insertar código PHP guardado en archivos ajenos al que se ha invocado mediante el uso de las funciones `include` y `require`. La diferencia entre una y otra radica en su comportamiento ante los errores de ejecución: `include()` genera un Warning y `require()` produce un error fatal y detiene la ejecución del script.

Matrices

Un array en PHP es realmente un mapa ordenado, un tipo de datos que asocia valores con claves. Este tipo está optimizado para varios usos diferentes; puede ser usado como una matriz real, una lista (vector), una tabla asociativa (una implementación de un mapa), diccionario, pila, cola, etc. Ya que los valores de un array pueden ser otros arrays también son posibles arrays multidimensionales.

Un array puede ser creado usando el constructor del lenguaje `array()`. Éste toma un cierto número de parejas clave => valor como argumentos.

```
array(
   clave  => valor,
   clave2 => valor2,
   clave3 => valor3,
   ...
   )
```

La coma después del último elemento del array es opcional y se puede omitir. Esto normalmente se hace para arrays de una sola línea, esto es, es preferible array(1, 2) que array(1, 2,).

A partir de PHP 5.4 se puede usar la sintaxis de array corta, que reemplaza array() con [].

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

// a partir de PHP 5.4
$array = [
    "foo" => "bar",
    "bar" => "foo",
];
?>
```

Un ejemplo donde se muestra por pantalla cómo evoluciona el array:

```
<?php
// Crear un array simple.
$array = array(1, 2, 3, 4, 5);
print_r($array);

// Ahora elimina cada elemento, pero deja el mismo array intacto:
foreach ($array as $i => $value) {
    unset($array[$i]);
}
print_r($array);

// Agregar un elemento (note que la nueva clave es 5, en lugar de 0).
$array[] = 6;
print_r($array);

// Re-indexar:
$array = array_values($array);
$array[] = 7;
print_r($array);
?>
```

Generando la siguiente salida:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)
Array
(
)
Array
(
    [5] => 6
)
Array
(
    [0] => 6
    [1] => 7
)
```

Más info: http://www.php.net/manual/es/language.types.array.php

Frameworks PHP

Desarrollar una web en PHP desde cero es una labor bastante ardua y repetitiva. Es por ello por lo que existen frameworks o librerías que permiten reutilizar código para las tareas más comunes y desarrollar de forma ágil toda la estructura de clases del sistema.

Un framework se diseña pensando en proporcionar una estructura para esos elementos comunes (interacción con la BD, capa de presentación, lógica de la aplicación), y lo normal es que se siga un paradigma MVC (Modelo-Vista-Controlador).

Los frameworks para PHP más comunes son:

- Zend (www.zend.com)
- Symfony (www.symfony.com)
- CakePHP (www.cakephp.org)
- CodeIgniter (www.ellislab.com/codeigniter)

Aunque hay muchos más. Lo normal es que uno se decante por uno u otro por comentarios y sobre todo, por conocer a alguien que habla maravillas de él.

ASP (Active Server Pages) y ASP.net

Es una tecnología propietaria y de código cerrado de Microsoft, y está diseñado especialmente para ser utilizado con IIS(Internet Information Server). El lenguaje utilizado con ASP era Visual Basic en su versión de *scripting (VBScript), y el utilizado en ASP.NET* es *C#* o cualquiera de los lenguajes .NET, que es el paraguas bajo el que Mi crosoft presentó un conjunto de tecnologías en 2002. Es un framework en el cual cada componente puede estar desarrollado en cualquiera de los lenguajes que desarrolla, pues todos los lenguajes convergen en un common Language Runtime (CLR), que gestiona el recolector de basuras, el chequeo de tipos, la depuración y la gestión de excepciones.

Para cada lenguaje .NET, el CRL tiene un compilador que pasa a este código intermedio, llamado ahora *Intermediate Language (IL)*. Antes de su ejecución, los programas IL se compilan a código máquina de forma incremental mediante un compilador *Just-in-Time*. Podríamos pensar que esto es parecido a Java, pero los programas en IL no son nunca interpretados, sino que son ejecutados.

Este código intermedio supone una serie de restricciones comunes a todos los lenguajes:

- No hay sobrecarga de operadores
- No hay punteros
- Los identificadores no son sensibles a mayúsculas

Curiosamente, aunque C# permite sobrecarga de operadores, se recomienda no usarlo si se va a interactuar con sistemas desarrollados en otro lenguaje, al igual que ocurre con los nombres de las variables (C# es sensible a mayúsculas).

Por tanto, es un poco complicado en tan poco espacio, tan siquiera comentar sucintamente esta tecnología, que daría para una asignatura completa. Tan sólo vamos a ver un ejemplo en c# y otro en VB tomado de W3Schools:

```
<!-- Single statement block -->
@{ var myMessage = "Hello World"; }

<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@{
var greeting = "Welcome to our site!";
var weekDay = DateTime.Now.DayOfWeek;
var greetingMessage = greeting + " Today is: " + weekDay;
}
<p>The greeting is: @greetingMessage</p>
```

Código 9 Script ASP.NET en C#

```
<!-- Single statement block  -->
@Code dim myMessage = "Hello World" End Code

<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@Code
dim greeting = "Welcome to our site!"
dim weekDay = DateTime.Now.DayOfWeek
dim greetingMessage = greeting & " Today is: " & weekDay
End Code

<p>The greeting is: @greetingMessage</p>
```

Código 10. Script ASP.NET en VB.NET

Abundante documentación sobre programación JAVA se encuentra en la web del profesor Chua Hock Chuan, de la Nanyang Technological University de Singapur

http://www3.ntu.edu.sg/home/ehchua/programming/index.html

Java Servlets

Los servlets Java son programas que se ejecutan en servidores que atienden al protocolo HTTP. La funcionalidad viene dada por los paquetes `javax.servlet` y `javax.servlet.http` , que proporcionan las interfaces y clases necesarias.

Los servlets normalmente heredan de la clase `HttpServlet` y sobrecargan los métodos `doGet` y `doPost`, que son los encargados de recibir y atender las peticiones HTTP.

Al contrario que los lenguajes de script vistos anteriormente que intercalan el ejecutable con código HTML, en los servlets toda la respuesta al navegador debe generarse dinámicamente, por lo que hay que realizar numerosas llamadas a `out.println(...)`

Además del código Java, hay que generar un XML que permita al servidor saber a qué URLs corresponde la ejecución de cada Servlet. Lo vemos con un *Hola Mundo,* invocable como `http://localhost:8080/helloservlet/saluda`:

```
// File "<CATALINA_HOME>\webapps\helloservlet\WEB-INF\src\mypkg\HelloWorldExample.java"
package mypkg;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldExample extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest request, HttpServletResponse response)
           throws IOException, ServletException {

    // Establecer el MIME type del mensaje de respuesta.
    response.setContentType("text/html;charset=UTF-8");

    // Obtener un element de impresion para generar la salida.
    PrintWriter out = response.getWriter();


    // Escribir el mensaje de respuesta en un documento HTML
    try {
      out.println("<!DOCTYPE html>");  // HTML 5
      out.println("<html><head>");
      out.println("<meta http-equiv='Content-Type' content='text/html;charset=UTF-8'>");
      String title = “Hola Mundo");
      out.println("<title>" + title + "</title></head>");
      out.println("<body>");
      out.println("<h1>" + title + "</h1>");  // Escribe "Hola Mundo"
      // Pone una imagen enlaada para refrescar
      out.println("<a href='" + request.getRequestURI() + "'>
                        <img src='images/return.gif'></a>");
      out.println("</body></html>");
    } finally {
      out.close();  // Siempre cerrar la salida
    }
  }
```

Código 12: Servlet Hola Mundo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

  <!—Guardado en <CATALINA_HOME>\webapps\helloservlet\WEB-INF\web.xml -->

   <servlet>
     <servlet-name>HelloWorldServlet</servlet-name>
     <servlet-class>mypkg.HelloWorldExample</servlet-class>
   </servlet>


   <servlet-mapping>
     <servlet-name>HelloWorldServlet</servlet-name>
     <url-pattern>/saluda</url-pattern>
   </servlet-mapping>
</web-app>
```

Código 11: Descriptor de Aplicaciones Web para el servidor que contiene el código 11

Para poder ejecutar un Servlet en un servidor web hay que tener activo algún módulo de los existentes en el mercado que sea capaz de ejecutarlos. El más común es Apache Tomcat, si bien se suelen usar servidores de aplicaciones , como GlassFish.

Ejercicio 28. *En el contexto del desarrollo de apliaciones Web, CATALINA no es un nombre de mujer. ¿Podrías explicar qué es y para qué sirve?*

JSP

Java Server Pages (JSP) es una tecnología que permite al desarrollador insertar código Java en páginas HTML insertando etiquetas JSP especiales entre los marcadores <% y %>.

Un componente JSP es un tipo de servlet Java que se diseña para realizar el rol de interfaz de usuario para una aplicación web. Los JSPs son archivos de texto que combinan HTML, elementos XML y acciones y comandos JSP embebidos.

Comparado con el clásico CGI, JSP ofrece varias ventajas:

- Aumenta el rendimiento, al insertar el código dinámico en páginas HTML
- JSP se compila antes de su procesamiento por el servidor (Perl es interpretado)
- JSP se construyen sobre la API de los Servlets, por lo que se tiene acceso a toda la API de Java.
- JSP se puede utilizar en combinación con los servlets, que gestionarían la lógica de negocio y el modelo.

Además, comparado con aplicaciones desarrolladas como servlets puros, es mucho más legible el código al tener las acciones intercaladas con el HTML.

El proceso habitual de ejecución se muestra en la ilustración 23, y se describe de la siguiente manera:

1. El navegador realiza una petición HTTP al servidor
2. El servidor reconoce que la petición se refiere a un archivo .jsp y redirige la petición al motor JSP (Tomcat)
3. El motor JSP carga la página JSP y la convierte en un servlet de forma automática.
4. El motor JSP compila el servlet y reenvía la petición original al motor de servlets.
5. Durante la ejecución del servlet se genera la salida en formato HTML y se envía al servidor web
6. El servidor web responde a la petición original HTTP

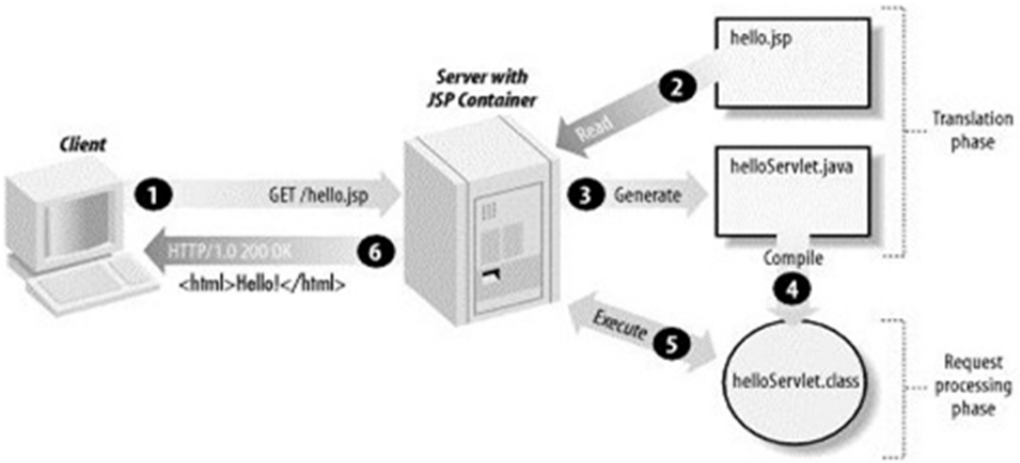


Ilustración 26: Proceso de resolución de una petición .jsp

Normalmente, el motor JSP almacena el servlet compilado mientras éste sea más reciente que el archivo JSP invocado, de forma que no tiene que recompilar cada vez que se realiza una llamada. Esto supone una ventaja frente a los scripts PHP que han de ser interpretados en cada invocación.

Un ejemplo de script JSP es el siguiente:

```
<html>
<head><title>Hello World</title></head>
<body>
Hello World!<br/>
<%
        out.println("Your IP address is " + request.getRemoteAddr());
%>
</body>
</html>
```

Otro con variables y sentencias intercaladas:

```
<%! int day = 3; %>
<html>
<head><title>IF...ELSE Example</title></head>
<body>
<% if (day == 1 | day == 7) { %>
 <p> Today is weekend</p>
<% } else { %>
 <p> Today is not weekend</p>
<% } %>
</body>
```

Un buen tutorial sobre JSP se puede encontrar en *www.tutorialspoint.com/jsp/jsp_tutorial.pdf*

NodeJS

NodeJS no es un lenguaje nuevo ni un framework de JavaScript. Es un entorno de ejecución en tiempo real para JavaScript utilizando el motor V8 de Google, es decir, nos proporciona un contexto para ejecutar JavaScript no solo en un documento HTML, sino en cualquier lugar donde se pueda instalar nodeJS.

Una característica de nodeJS es que, externamente, se ejecuta en una única hebra. Es decir, todas las peticiones que se reciben se ejecutan sobre el mismo espacio de memoria. Sin embargo, es un entorno **orientado a eventos** y **asíncrono**. Esto es lo que lo hace realmente potente, pues su asincronía permite ejecutar distintos trozos de código sin bloquear en ningún momento el servidor.

Para la gestión de los distintos recursos disponibles (librerías o módulos) se utiliza el gestor de paquetes **npm**. Instalar un nuevo módulo en la aplicación desarrollada en nodeJS es tan simple como hacer `npm install package`.

Un servidor sencillo en Node.js

Este código podríamos deducir que sustituye todo el "aparataje" que monta Apache en un servidor web:

```
var http = require("http");

function onRequest(request, response) {
  console.log("Peticion recibida");
  response.writeHead(200, {"Content-Type": "text/html"});
  response.write("Hola Mundo");
  response.end();
}

http.createServer(onRequest).listen(8080);
console.log("Servidor iniciado");
```

Obviamente faltan muchos elementos de seguridad, alias, etc. que ofrece Apache, pero podemos ver que tenemos lo más importante: alguien escuchando peticiones http por el puerto 8080.

Precisamente por su ligereza, Node.js está pensado para aplicaciones con gran número de conexiones concurrentes que requieran poco cálculo. Para profundizar en la discusión, he aquí una hebra de StackOverflow donde se discute: http://stackoverflow.com/questions/5062614/how-to-decide-when-to-use-node-js. A modo de resumen, podemos pensar que Node.js es adecuado para:

- Juegos online.
- Gestores de correo online
- Herramientas colaborativas (foros, canvas, etc.)
- Chats.
- Redes sociales

Una buena referencia para comenzar con Node.js es http://www.nodebeginner.org/

Gestión de una petición con Node.js

En clase comentaremos el siguiente código, que muestra de forma incipiente la gestión de las peticiones GET y POST:

```
var http = require("http");
var dispatcher = require('httpdispatcher');

var server = http.createServer(handleRequest);


function handleRequest(request, response){
   try {
      console.log(request.url);
      dispatcher.dispatch(request, response);
   } catch(err) {
      console.log(err);
   }
}

//ruta donde se están los elementos estáticos (imágenes, js, css)
dispatcher.setStatic('resources');

// Petición GET
dispatcher.onGet("/page1", function(req, res) {
   res.writeHead(200, {'Content-Type': 'text/plain'});
   res.end('Pagina 1');
});

// Petición POST
dispatcher.onPost("/post1", function(req, res) {
   res.writeHead(200, {'Content-Type': 'text/plain'});
   res.end('Recibidos datos 2');
});

// Arrancamos el servidor
server.listen(8080, function(){
   console.log("Server listening on: http://localhost:%s", PORT);
});
```

ARQUITECTURAS ORIENTADA A SERVICIOS

Uno de los retos de las aplicaciones Web es la interoperabilidad necesaria en el funcionamiento de grandes sistemas informáticos, y éstos puede que no se ejecuten en la misma plataforma e incluso estén desarrollados con tecnologías y marcos de trabajo diferentes. En el año 2000, se presentó al W3C la arquitectura orientada a servicios (SOA, *Service Oriented Architecture*), que brinda entre otras cosas una forma estándar de publicar y utilizar servicios, conocidos comúnmente como servicios web (*web services*).

Un servicio web puede hacer cualquier cosa que el servidor desee poner a disposición del resto del mundo: Web Services científicos, geográficos, de negocios, validación, clima, compras, etc. Una empresa / organización / individuo puede tener muchas razones para publicar un Web Service, desde comerciales hasta altruistas.

De esta forma, una aplicación es vista como un conjunto de servicios, que intercambian mensajes en formato XML sobre el protocolo de transporte HTTP. Esta arquitectura permite construir sistemas distribuidos heterogéneos de forma que una organización expone sus competencias (funcionalidades o capacidades) para que sean utilizados por la misma organización o por otras organizaciones. Estos servicios, que pueden verse como una llamada a una función local o remota, independiente del lenguaje de programación y plataforma que se utilice, consta de una interfaz bien definida.

SOA busca el mínimo acoplamiento entre los componentes, favoreciendo la identificación de un conjunto de *servicios* en la red, emulando el comportamiento de los negocios en el mundo real:

- El negocio dirige los servicios y los servicios dirigen la tecnología (lo servicios son una capa de abstracción entre el negocio y la tecnología que lo implementa)
- La agilidad del negocio es un requisito fundamental del propio negocio (la habilidad para responder los cambios en los requisitos es un requisito en sí mismo)

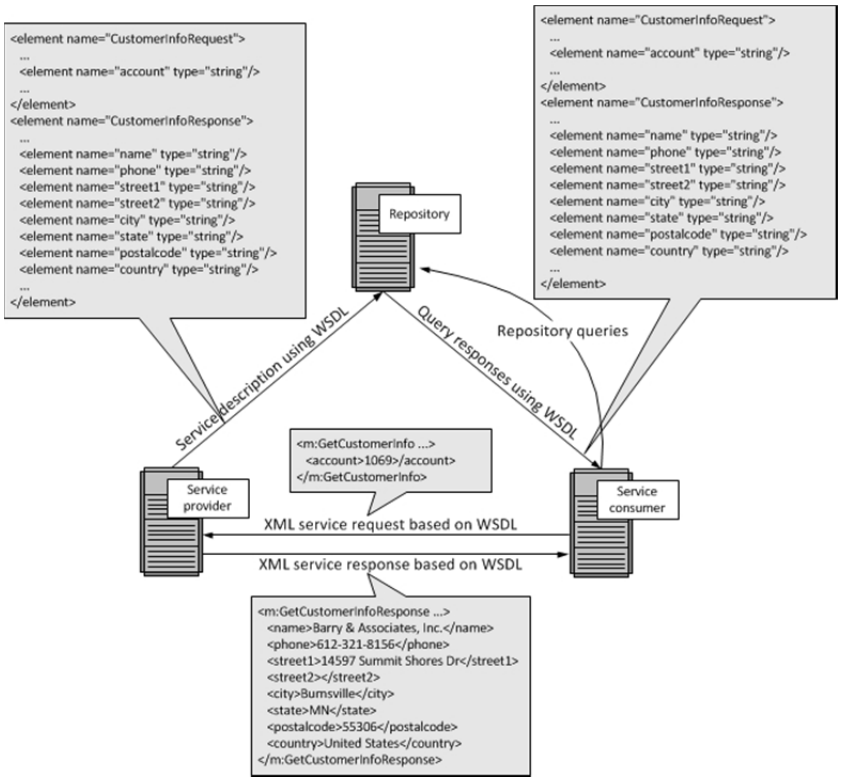


Ilustración 27: Paso de mensajes SOA

SOAP

En una arquitectura SOAP (Simple Object Access Protocol) intervienen tres tipos de actores*:* el *proveedor de servicios* es un elemento que proporciona un servicio en respuesta a una llamada o petición desde un *consumidor de servicios*, que ha utilizado el *publicador de servicios* para obtener información sobre los servicios disponibles y las interfaces que hay que utilizar para invocarlos. Las interfaces y los servicios se pueden describir en un entorno SOAP mediante lenguajes basados en XML: las interfaces en el lenguaje WSDL (*Web Service Description Language)*, y los listados de servicios en UDDI (*Universal Description, Discovery and Integration)*.

En la ilustración 24 vemos un ejemplo. Se ve cómo el *proveedor* envía al *publicador de servicios* la descripción de su servicio usando WSDL (y esto mismo se responde a los consumidores):

- Se describe la petición como "CustomerInfo" y un parámetro denominado account
- Se describe la respuesta *"CustomerInfoResponse"*

También se ve cómo el *consumidor* solicita con XML los datos de una cuenta:

```
<m:GetCustomerInfo>
<account>1069</account>
</m:GetCustomerInfo>
```

Y cómo el *proveedor* remite la respuesta con el formato descrito en el WSDL publicado:

```
<m:GetcustomerInfoResponse>
<name>Barry & Associates, Inc. </name>
<phone>623-432-8156</phone>
…
</m:GetCustomerInfoResponse>
```

REST (Representational State Transfer)

REST es una alternativa a SOAP y JSON, y es un tipo de arquitectura de software más que un conjunto de estándares. Las aplicaciones desarrolladas con esta arquitectura se denominan *RESTful*. Como detalle de su extensión, los Amazon Web Services proporcionan este tipo de arquitectura, que se basa exclusivamente en el uso del protocolo HTTP.

Un servicio REST es:

- Independiente de la plataforma
- Independiente del leguaje
- Basado en estándares
- Fácilmente usable aun existiendo cortafuegos.

REST ignora los detalles de la implementación de los componentes y sintaxis de los protocolos y se centra en el rol de cada componente, sus restricciones en cuanto a la interacción y la interpretación de los elementos de datos. REST se basa en cinco principios:

- *Todo recurso tiene un ID*. De esta forma, las URIs de una petición podrían ser:

  ```
  http://example.com/orders/2007/10/776654
  http://example.com/processes/salary-increase-234
  http://example.com/products?color=green
  ```

- *Las cosas se enlazan juntas*, o lo que es lo mismo, en la transmisión de los recursos se incluyen los enlaces para obtener más información de los detalles de dicho

recurso, de forma que el cliente puede pasar de un estado a otro de la aplicación con un enlace:

```
<order self='http://example.com/customers/1234' >
  <amount>23</amount>
  <product ref='http://example.com/products/4554' />
  <customer ref='http://example.com/customers/1234' />
</order>
```

- *Se usan métodos estándar*, además del GET y el POST, en estas arquitecturas se utilizan las otras acciones de HTTP (PUT, DELETE, HEAD y OPTIONS), de forma que cada recurso o servicio implementa la respuesta a estas acciones (ilustración 25).
- *Los recursos tienen múltiples representaciones*, por ejemplo, podemos solicitar los datos de un cliente en formato VCARD o en XML específico de la compañía:

```
GET /customers/1234 HTTP/1.1
Host: example.com
Accept: application/vnd.mycompany.customer+xml


GET /customers/1234 HTTP/1.1
Host: example.com
Accept: text/x-vcard
```

- *La comunicación es sin estado*. Esto no quiere decir que no se pueda guardar en el cliente el estado de la comunicación, pero el servidor debe mantenerse al margen de las sesiones creadas y su evolución (es decir, no se usan cookies). Esto hace al servidor más robusto frente a caídas y a aumentos de cargas.

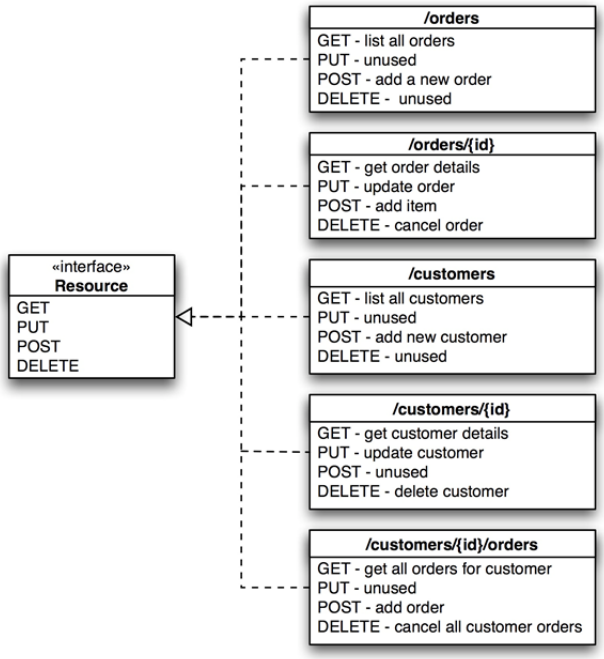


Ilustración 28: Respuestas HTTP en función del URI

Comparación con SOAP

Supongamos que queremos obtener los detalles de contacto de un usuario. Con SOAP, la consulta XML sería enviada en una variable POST al servidor, y tendría este código:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
```

```
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

Mientras que con REST la petición al servidor sería

```
http://www.acme.com/phonebook/UserDetails/12345
```

o con parámetros por GET

```
http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe
```

Nótese que la petición va en la misma URL, no en el cuerpo de la petición. Un detalle es que el método no se llama "getUserDetails", sino "UserDetails". Es porque en REST es una norma no escrita el usar nombres en lugar de verbos para denotar recursos.

La respuesta que ofrece el servidor a las peticiones REST suele ser en formato XML, aunque también podría ser en formato CSV o JSON. Sólo se acepta HTML como respuesta cuando está documentado que el servicio proporciona un documento legible por un humano.

Ejemplos de servicios REST reales:

- Google Glass API
- Twitter REST API
- Flickr
- Atom (alternativa RESTful a RSS)

Si SOAP es como enviar una carta, REST es como enviar una tarjeta postal.

Algunos puntos a tener en cuenta a la hora de implementar una arquitectura REST son:

- No usar URL físicas (p.ej. http://do.it/productos/doc1.xml) sino URL lógicas (http://do.it/productos/001)
- Las consultas no deben devolver un exceso de datos, por lo que hay que proporcionar si fuese necesario un mecanismo de paginación.
- Hay que tener bien documentado y actualizado el formato de respuesta.
- Si en la respuesta se permiten acciones adicionales (mediante URLs), hay que darlas explícitamente en la respuesta y no dejar "recetas" al cliente para que la construya.
- La llamada GET nunca causa un cambio de estado. Para cambiar cosas en el servidor se usará POST o DELETE.

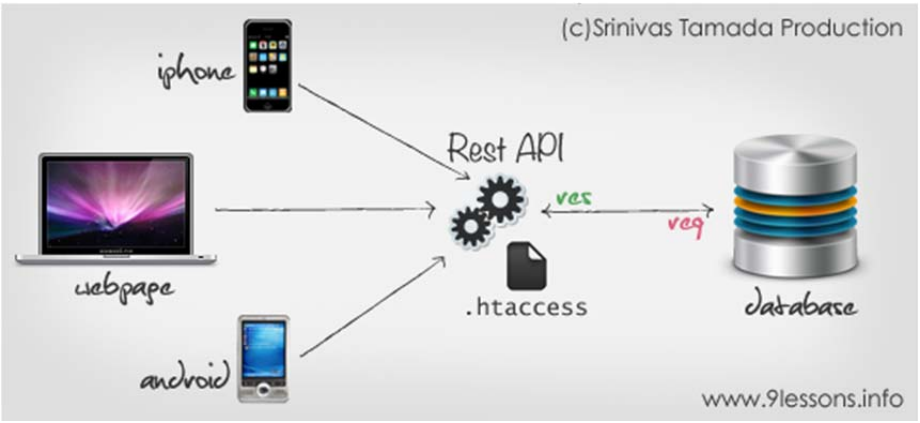


Ilustración 29: Esquema de una aplicación RESTful

¿Cómo se implementa una aplicación RESTful? Básicamente se trata de por un lado traducir las URL a llamadas a los scripts adecuados, lo que se hace con los archivos .htaccess de Apache:

```
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ api.php?rquest=$1 [QSA,NC,L]

RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^(.*)$ api.php [QSA,NC,L]

RewriteCond %{REQUEST_FILENAME} -f
RewriteRule ^(.*)$ api.php [QSA,NC,L]
</IfModule>
```

Todo lo que no sea directorio o un archivo existente será reescrito con una llamada al archivo api.php y como parámetro rqest de la petición la secuencia de caracteres introducida a partir del nombre del servidor:

P.ej. La llamada http://localhost/rest/deleteUser se convierte en

```
http://localhost/rest/api.php?rquest=deleteUser
```

Cuando se invoque un directorio o un archivo directamente se invoca a api.php sin parámetros.

En el archivo api.php tendríamos las funciones del servidor, que detectan qué tipo de operación HTTP estamos invocando. En el caso del ejemplo anterior, todo está en un único fichero api.php donde primero se determina qué acción se quiere realizar:

```
public function processApi() {
 $func = strtolower(trim(str_replace("/","",$_REQUEST['rquest'])));
 if((int)method_exists($this,$func) > 0)
       $this->$func();
 else
       $this->response('',404); // Page not Found
}
```

Y en el caso de llamar con *http://localhost/rest/deleteUser/2* se ejecutaría, tras extraer los parámetros, la siguiente función que borra un usuario de la base de datos:

```
private function deleteUser() {
   // Comprobamos que se ha usado el método DELETE en la llamada HTTP
   if($this->get_request_method() != "DELETE"){
         $this->response('',406);
   }
   $id = (int)$this->_request['id']; // _request es rellenado previamente
   if($id > 0) {
    mysql_query("DELETE FROM users WHERE user_id = $id");
    $success = array("status" => "Success",
                     "msg" => "Successfully one record deleted.");
    $this->response($this->json($success),200);
   }
   else {
    $this->response('',204); // If no records "No Content" status
   }
   }
```

Ejercicio 29. *Como se ha comentado, la mayoría de los servicios más famosos de la red proporcionan servicios mediante arquitecturas REST. Documente en sus apuntes las funciones, a su entender, más útiles de la API 1.1 de Twitter. ¿Cómo se podría obtener el Tweet más reciente de un usuario concreto?*

Ejercicio 30. *¿Cómo se podría programar una api RES con NodeJS? ¿Y con PHP?*

Ejercicio 31. *Disponemos de una base de datos centralizada a nivel nacional para la gestión de becas, donde se almacenan datos socio-económicos de los alumnos. Cada universidad debe desarrollar su propia aplicación para la gestión de las becas de su plan propio, pero los datos de los alumnos se extraerán y registrarán en esta base de datos central. En una reunión entre los ingenieros del MECD y de las universidades se discute sobre cómo acceder a esos datos y anotar las becas concedidas. Se plantean varias opciones:*

- *acceso directo con privilegios de consulta al servidor Oracle SGBD donde están los datos*
- *desarrollo de servicios web SOAP*
- *desarrollo de una API REST*

Evalúe, desde su conocimiento como experto en Sistemas de Información Basados en Web, las tres alternativas presentadas y justifique la elección de una u otra, explicando las principales características de la solución escogida.