PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Ingeniería Web

GONZALO TELLO VALENZUELA

Docente: Felipe García

2do Semestre, 2019

Índice

Lista de Figuras

Lista de Tablas

1. Arquitectura básica cliente/servidor y protocolo HTTP

1.1. Resumen historia de la web

- 1989: Propuesta de Tim Berners - Lee
- 1991: Protocolo HTTP¨, lenguaje HTML
- 1992: Navegador Mosaic
- 1994: Navegadores: Netscape, Opera, Internet Explorer
- 1995: Javascript
- 1996: XML (Extensible Markup Language)
- 1997: Flash
- 1999: AJAX
- 2002: Navegador Solari
- 2004: Navegador Firefox
- 2006: Navegador Chrome

1.2. ¿Por qué ha sido tan existosa la web?

Probablemente por una combinación de los siguientes factores:

- **Simplicidad:** Basta saber un poco de HTML y tener acceso a un servidor web.
- Tecnologias gratuitas y libre de patentes o impedimentos comerciales
- Sirvió para satisfacer un nicho: La publicación personal de contenido.
- **Accesibilidad:** Cualquier persona con acceso a internet puede, en principio, acceder a cualquier contenido público.
- Uso progresivo de la web en aplicaciones cotidianas: comercio, cuentas, trámites, etc.

1.3. Páginas vs Sitio vs Aplicación

1.3.1. Páginas web

- Contenido estático
- Sin coherencia estética, conceptual o de contenidos
- Contenidos ligeramente conectados
- Básicamente un puñado de páginas estáticas.

1.3.2. Sitio web

- Contenido (mayormente) estático.
- Consistencia temática, de contenidos y estéticas
- Estructura jerárquica o bien organizada de contenidos.
- Existe un diseño arquitectónico que apunta a la fácil mantenición del sitio.

1.3.3. Aplicación Web

- Una aplicación cliente/servidor
- Genera contenido de manera dinámica, en base a las peticiones y necesidades de sus usuarios.
- Diseño arquitectónico avanzado, considera seguimiento, seguridad, etc.

1.4. Web X.0

1.4.1. Web 1.0 - La web estática

- Principalmente páginas estáticas
- Mucho uso de framesets/iframes para incrustar contenido
- El aporte por parte de los usuarios (comentarios, etc) prácticamente inexistente.

1.4.2. Web 2.0 - La web social

- Comienza alrededor del año 2001, con una explosión en el desarrollo de tecnologías web.
- Generación dinámica de contenido y los sitios como aplicaciones.
- Los usuarios añaden valor: comentarios, redes sociales, crowdsourcing, etc.
- La web como plataforma de aplicaciones más allá de 1 dispositivo.
- Cambios en los paradigmas de desarrollo

1.4.3. Web 3.0 - La web de las máquinas

- La característica principal es la web como espacio universal de información.
- Esta información además debe ser 'facilmente' accesible y manipulable por software más que por humanos.
- Esta accesibilidad usualmente es dada por web services, que igual se usan en web 2.0
- Dado lo anterior, se facilita la aplicación de técnicas de inteligencia computacional.
  - Análisis de textos, twitter, faceboo, etc.
  - Análisis de red como grafo con intereses sociales, etc.
- Usualmente se asocia con la web semántica y sus tecnologías: RDF, SPARQL.
- También se asocia con 'arquitecturas orientadas a servicios'

1.5. ¿Qué entendemos por arquitectura?

Dentro de las muchas definiciones destacaremos lo siguiente:

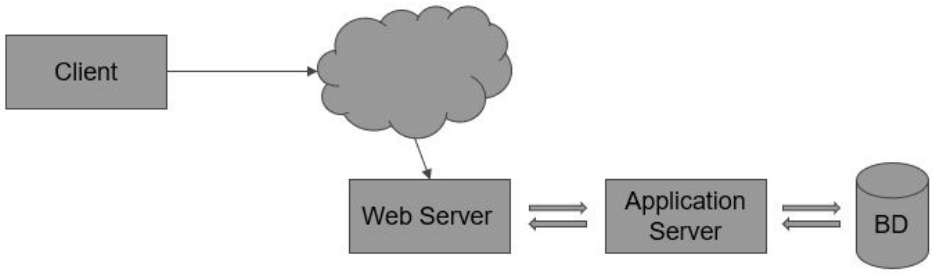
- La arquitectura describe la estructura de un sistema
- La arquitectura forma la transición desde el análisis hacia la implementación

- Una arquitectura puede verse desde distintos puntos de vista.
- La arquitectura hace que un sistema sea entendible.
- La arquitectura representa un marco de trabajo flexible y adaptable al cambio.

Es fundamental distinguir entre:

- La arquitectura de la infraestructura web: routers, servidores, firewalls, switches, balanceadores de carga, alta disponibilidad, etc.
- La arquitectura de las aplicaciones web: Patrones de diseño, separaciones de responsabilidad, paradigmas de programación.
- Usualmente se puede mostrar una visión mixta.

1.6. Componentes generales arquitectura física



- **Cliente:** Generalmente un navegador (user-agent) controlado por un usuario para acceder a recursos web o para operar una aplicación web.
- **Web services:** Servidor que recibe y procesa las peticiones HTTP de los clientes.
- **Aplication Server:** Expone la lógica de negocios a través de protocolos como HTTP u otros. Generan dinámicamente el contenido de las aplicaciones web.
- **Database Server:** Provee persistencia y administración de datos, generalmente se usan BD relacionales, pero también pueden ser 'NO SQL'.

1.7. Bloques esenciales de la web

En su mínima expresión, la web consiste en 3 bloques esenciales:

- Un lenguaje de marcado para formatear documentos de hipertexto
- Una notación uniforme para acceder a recursos web en las redes.
- Un protocolo para transportar mensajes a través de las redes.

1.8. Markup Language / Lenguaje de marcado

Permiten anotar un documento de manera que las notaciones sean sintácticamente diferenciables del contenido. El lenguaje HTML es esencialmente el estándar para los documentos de hipertexto usados en la web.

1.9. Direccionamiento uniforme de recursos

1.9.1. Universal resource identifie (URI)

String que se usa para identificar un nombre o recursos en internet.

- **Universal Resource Locator(URL)**: Especifica la ubicación y mecanismo para acceder a un recurso.
  - FTP: //Haxx.pirate.org/foo/series/twd/s03E01.mkv
  - ssh://jjh@github.com/the_matrix.git
  - HTTP://www.inf.pucv.cl
- **Universal Resource name (URN)**: Solamente identifica un recurso pero no dice como obtenerlo ni en donde está.
  - urn:ietf:rfc:2648
  - urn:isbn:0451450523
  - urn:isan:0000-0000-9E59-0000-O-0000-0000-2

1.10. Esquema general de sintaxis URL

scheme://host[:port]/path/.../[;url-params][?query-string][#anchor]

- **Scheme:** Indica el protocolo a utilizar, HTTP, FTP, SSH, etc.
- **Host:** Dirección IP o nombre de dominio del servidor al que se accede.

- **Path:** Ruta de acceso al recurso, relativo a la configuración del servidor.
- **URL-Params:** Manera poco usada para especificar parámetros.
- **Query-String:** Manera archiconocida de pasar parámetros mediante llave/valor
- **Anchor:** Marcador posicional dentro del documento.

1.11. Protocolo Hypertext Transfer Protocol - HTTP

'Es un protocolo a nivel de las aplicaciones que es ligero y veloz permitiendo la colaboración distribuida entre los distintos componentes de un sistema de hipertexto o hipermedios' (RFC 1945, 1996).

- Es el ingrediente básico esencial para construir lo que hoy se conoce como 'la web'
- Define un estándar para la petición de recursos en la web.
- Un recurso puede ser un documento HTML, un archivo, un script, la ejecución de un programa, etc.
- El corazón de la comunicación en la web es el protocolo HTTP:
  - Cliente/servidor o con paradigma request/response stateless
  - Formato simple, en texto plano, legible por humanos y máquinas.

1.12. HTTP como protocolo de comunicación

- Se encuentra en la capa de aplicaciones, el transporte depende de las capas inferiores. Ejemplo: TCP/IP
- Versiones estándarizadas por IETF: RFC2616, RFC7230

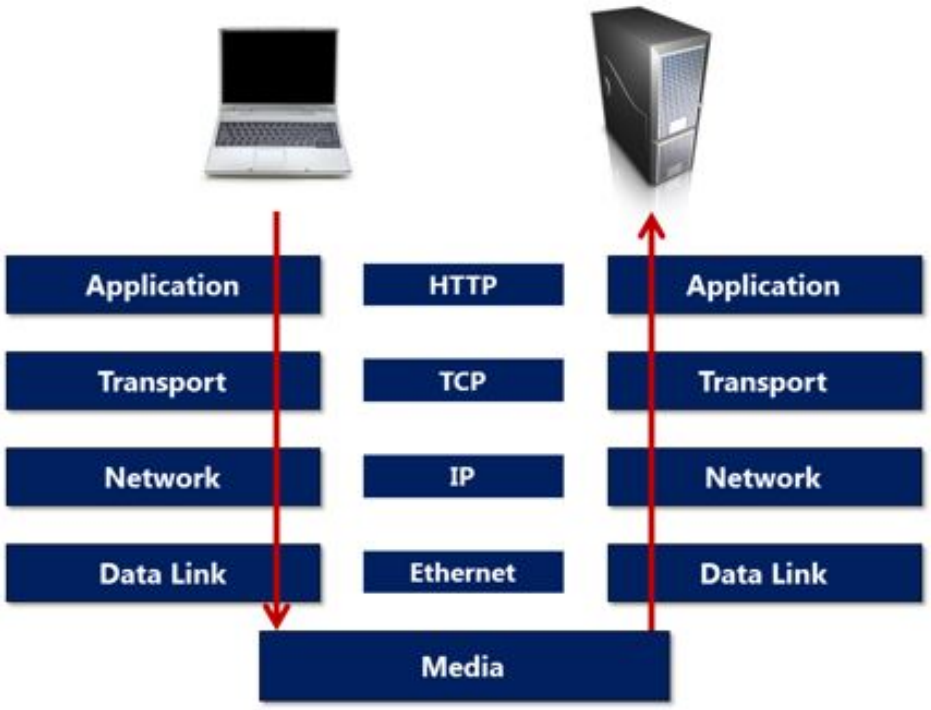


Figura 1: Modelo request/response: Mensaje con texto plano

1.13. Consideraciones extras

- HTTP como protocolo para el paso de mensajes.
  - No nos preocuparemos de los detalles de la conexión de red y protocolos subyacentes
  - La gracia de HTTP es que se usa como un protocolo estandarizado para la comunicación entre sistemas distribuidos.
  - Además, al usar JSON o XML como formato de intercambio, podemos enviar datos serializados. Esto es la base de los 'web services'
- Provee por defecto un modelo inherentemente descentralizado, lo que se aprovecha por ejemplo para:
  - Internet of things: Cada 'cosa es a la vez un cliente y un servidor, puede recibir y enviar mensajes.

- Bases de datos NoSQL y cualquier sistema que apunte a la interoperabilidad, especialmente sobre la web, provee acceso sobre HTTP.

1.14. Métodos HTTP para los request

Figura 2: Ejemplo de HTTP Request

Los métodos de request permiten diferenciar distintos tipos de mensajes, con restricciones o características particulares, para que el servidor web y los clientes puedan diferenciar lo que se busca hacer. Los métodos más comunes:

- **Get:** El método más simple y usado, solicita un recurso. En general no tiene 'cuerpo', y se parametriza mediante el *query string.*
- **Post:** También solicita un recurso, pero a diferencia de GET, tiene un 'cuerpo'. Generalmente se usa para someter formularios, donde los valores de los campos van en el cuerpo.

1.15. Códigos de estatus para los responses

El código de status en las respuestas indica al cliente qué pasó con la petición.

- 1xx: Informativo
- 2xx: Indican operaciones exitosas
- 3xx: Indican redirecciones o acciones adicional que el cliente deben hacer

- 4xx: Representan errores en el request, o sea, culpa del cliente
- 5xx: Representan errores en el servidor.

1.16. Headers

- Los encabezados o headers son un sistema de metadatos para comunicación entre el cliente y el servidor.
- Se usan tanto en los request como en las responses, aunque hay algunos que sólo tienen sentido al ser enviados por el cliente, o en una respuesta del servidor. Están clásificados en:
  - Generales
  - Client-Request
  - Server-Response
  - Entity

1.16.1. Formato

Si bien los nombres y valores específicos de cada parámetro varían según cada header, las síntaxis general para los headers es:

```
<nombre-header>: valor-param1; valor valor-param2; ...; valor-param3
```

Por ejemplo:

- *'Expires: Mon, 05 Mar 2018 06:00:00 GMT'*
- *'Accept: text/HTML; Q=0.8, text/plain; Q=0.5, text/javascript'*
- *'Accept-Enconding: Gzip, deflate'*

1.16.2. Client-Request

- **User-Agent**.

  Identifica el software, usualmente el navegador, usado para generar el request.

- **Referer**

  Indica 'de donde provino el acceso' a la URL.

- **Authorization**

  Se utiliza para solicitar recursos protegidos.

1.16.3. Server-response

- **Location**

  En caso de redirecciones, indica la nueva URL a la que debiera dirigirse el cliente.

- **WWW-Authenticate**

  Ante un acceso no autorizado a un recurso, este header especifica que es necesario autentificarse y ahí se podrá acceder.

- **Server**

  Identifica el software utilizado en el servidor.

1.16.4. Content-Types

- **Content-enconding**

  Especifica el formato comprimido en el que se envía un recurso, por ejemplo: GZIP, COMPRESS, DEFLATE.

- **Content-type**

  Especifica el contenido del documento en función de una jerarquía de tipos/subtipos, por ejemplo: text/html, application/pdf, etc.

1.16.5. Otros

- **Control de cache:** Indica si el recurso puede ser puesto en cache, su duración/expiración, etc.

- **Autentificación:** El servidor puede establecer un desafío de autentificación, típicamente un password. Si no usa HTTP's, el password se envía codificado pero no encriptado (muy inseguro).

- **Manejo de cookies:** Indica si el cliente debe setear cookies, o bien, el cliente envía cookies a un servidor.

- **Cookies:** El header '*set-cookie*' indica al cliente que debe recordar y reenviar cierta información con cada request, usando el header cookie. De esta forma, se puede simular de manera muy frágil, la ilusión de tener un valor o variable compartidos.

1.17. Sesiones

Una sesión es cuando el ervidor almacena información, asociandola a una llave única de acceso. Si el cliente envía la llave de acceso, el servidor podrá 'recordar' y utilizar los datos almacenados. La 'metáfora del casillero' es bastante acertada:

- Es una forma de almacenamiento temporal
- Si perderemos la llave, perderemos las cosas guardadas.
- Luego, de un tiempo sin uso, se limpiarán los casilleros.
- Si alguien roba la llave, tendrá acceso a lo que está guardado.

Una sesión permite guardar valores almacenados como variables en el programa del servidor. En general, pueden almacenar más datos que usan cookies. Se puede pasar la llave de la sesión al servidor por distintos mecanismos.

- Cookies – La famosa 'cookie de la sesion'
- Como parámetro de la URL o el request
- Usando algún otro tipo de header que sea entendido por el servidor.

2. Panorama tecnológico - Front End

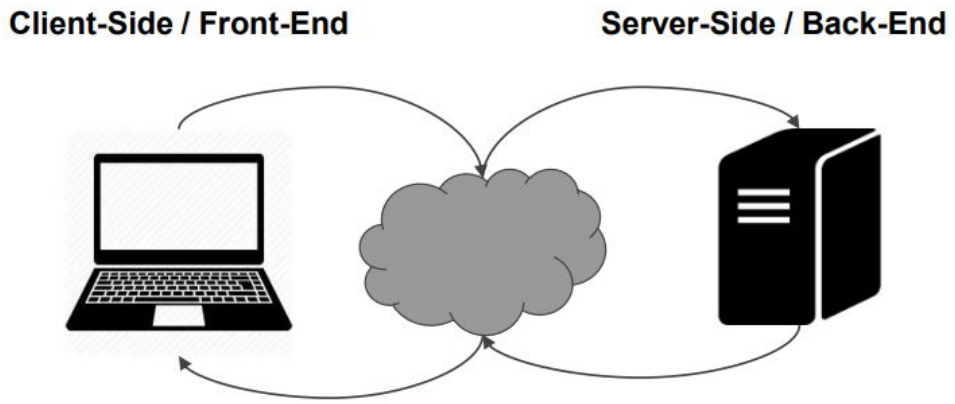


Figura 3: Stack de tecnologías web

2.1. Desarrollo en el Front-End

Cuando hablamos de front-end nos referimos a todo el código que se está ejecutando en el lado del cliente y, por contraste, no se ejecuta en el servidor. En la práctica, se refiere al código que está ejecutando el navegador web.

- Renderizado por HTML
- Aplicación de estilos CSS
- Ejecución de código Javascript

El código que se ejecuta en el front-end se obtiene al visitar un sitio web, es decir, es enviado inicialmente por un servidor web.

2.2. Tecnologías

- **Estructura**: HTML es un lenguaje declarativo que especifica la estructura del documento
- **Presentación**: CSS es un lenguaje declarativo que especifica la presentación de un documento HTML.
- **Interacción** Javascript o JS es un lenguaje multi-paradigma para modificación dinámica del documento a través del DOM.

2.3. Hypertext Markup Language (HTML)

Más allá del texto, el hipertexto que puede ser no lineal es un texto que contiene enlaces a otros textos.

Es un concepto que se va más allá de su actual implementación en productos de software, en donde los textos están enlazados para ir de un conceptro a otro, encontrando la información que uno quiere, formando una web. El proceso de ir de un texto a otro se llama navegación.

HTML es una encarnación del hipertexto, el cual tiene contenido:

- El texto
- Referencias a imágenes
- Sonidos
- Videos

Contiene además elementos que especifican la estructura y el significado de un documento, y que pueden estar parametrizados por atributos. El navegador web cumple la función de renderizar el documento HTML.

```html
<html>
    <head>
        <title>Todo Listo!</title>
    </head>
    <body>
        <h1>Todo Listo! Mi lista de tareas</h1>
        <form>
            <label for="titulo">Titulo</label>
            <input type="text" name="titulo"/>
            <label for="descripcion">Descripcion</label>
            <input type="text" name="descripcion"/>
            <input type="submit" value="Agregar Tarea">
        </form>
        <hr/>
        <h2>#1: Comprar en el supermercado</h2>
        <p>Yogurt, jugos, shampoo.</p>
        <hr/>
        <h2>#2: Preparar Talleres</h2>
        <p>Preparar taller de desarrollo front-end para Ingenieria Web</p>

    </body>
</html>
```

Figura 4: Ejemplo básico HTML

Figura 5: Vista desde el navegador para el ejemplo anterior

2.3.1. Elementos HTML

Un elemento consiste en una etiqueta de inicio y otra de de cierre, con el contenido insertado entre las dos etiquetas. Por ejemplo:

<etiqueta>Aqui va el contenido ... <\etiqueta>

El elemento HTML consiste en todo lo que hay entre las etiquetas de inicio y cierre. Esto puede incluir elementos anidados.

Algunas pueden cerrarse inmediatamente sin contenido, como por ejemplo <\input>.

Los 10 elementos HTML indispensables son:

<H1>hasta <H6>	Encabezados
<p>	Párrafo
<i>	Cursiva
<b>	Negrita
<A>	Enlace \Anchor
<OL>, <CU>, <LI>	Lista ordenada, no ordenada, items de la lista
<Blockquote>	Bloque de texto identado, para hacer citas textuales.
<HR>	Linea horizontal
<IMG>	Imágen
<DIV>	Crea una nueva agrupación en un elemento de bloque.

Tabla 1: 10 elementos indispensables en HTML

Todos los elementos pueden tener atributos, los cuales proveen información adicional sobre el elementop. Los atributos siempre se especifican en la etiqueta de inicio. Los atributos se especifican como pares nombre = 'valor'. Por ejemplo:

<input type='text' name='título'>

Algunos elementos globales sólo tienen atributos, y no tienen contenido, por ejemplo, <IMG>.

Ahora bien, algunos atributos globales de HTML son:

Id	Especifica un identificador único para el elemento
Class	Especifica uno o más nombres de clase que etiquetan el elemento
Hidden	Indica que el elemento aún no existe, o ya no es relevante. El navegador no debería mostrar el elemento al usuario.
style	Especifica el estilo visual del elemento, usando CSS. Sobreescribe cualquier valor global aplicado al documento.
Title	Especifica información adicional sobre el elemento. La infomación se muestra generalmente como un *tooltip*.

Tabla 2: Atributos globales HTML

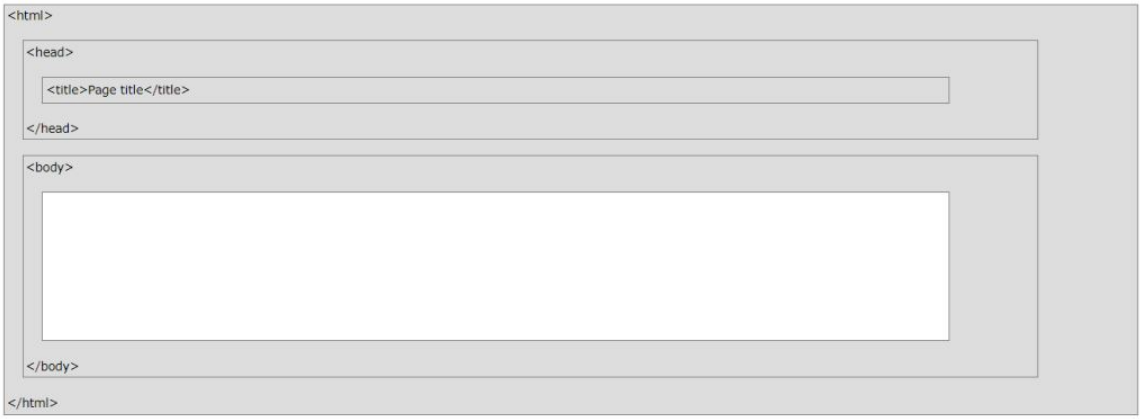


Figura 6: Estructura de un documento HTML

2.4. Cascading style Sheets (CSS)

CSS es un lenguaje declarativo que especifica la presentación de un documento HTML. Este es un lenguaje que describe el estilo de un documento HTML, el cual se puede aplicar de manera muy específica a elementos particulares, o muy general, a conjuntos de elementos. Además, describe cómo los elementos HTML deben ser mostrados por el navegador.

2.4.1. Selectores

Un selector es un patrón que se usa para seleccionar los elementos a los cuales se les desea aplicar un estilo. El manejo de selectores también es clave para la manipulación con Javascript y el web scraping.

Figura 7: Ejemplo de selectores

- **Sector ID:** El selector ID selecciona el elemento que tiene asignado, que se supone debería ser único en el documento. La siguiente línea selecciona el elemento con ID = 'HEADER1'

  #Header{...}

  Se espera a lo más 1 elemento asociado.

- **Selector Class:** El selector class selecciona todos los elementos que pertenecen a una clase dad. La siguiente linea selecciona todos los elemnetos de la clase tarea:

  .Tarea{...}

  Se espera una cantidad arbitraria de 0 o más elementos. Un elemento puede pertenecer a varias clases.

Ahora bien, a modo de ejemplo, ¿Cómo podemos definir que los encabezados de nivel 2 tendrán texto azul y fondo rojo?, además de que los elementos de la clase 'tarea' tendrán efecto amarillo, los parrafos tendrán el texto alineado a la derecha y el elemento con el ID 'header 1' tendrá su texto en negritas. En efecto;

```css
h2 {
    color: red;
    background-color: blue;
}

.tarea {
    visibility: hidden;
}

p {
    text-align: center;
}

#header1 {
    font-weight: bold;
}
```

2.4.2. Como incluir/aplicar CSS: inline, interno, externo

```
<i style="color: red;">#1: Comprar en el supermercado</i>
```

Figura 8: Estilo inline

```html
<html>
    <head>
        <title>Todo Listo!</title>
        <style>
            h2
            {
                color: blue;
            }
        </style>
    </head>
```

Figura 9: hoja de estilo interno

```html
<link rel="stylesheet"
      type="text/css"
      href="estilo1.css" />
```

estilo1.css

Figura 10: Hoja de estilo externo

2.4.3. Cascading

¿Qué estilo aplicar cuando hay más de uno que puede aplicar a un elemento? Los estilos 'hacen cascada' generando hojas de estilos virtuales, con la siguiente prioridad:

- Estilos inline
- Hojas de estilo externas e internas
- Los valores por defecto del navegador

2.4.4. Sintaxis

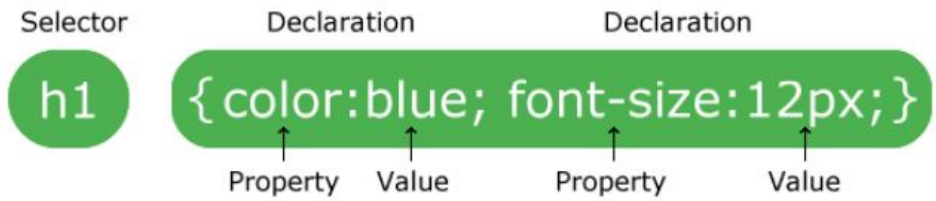


Figura 11: Síntaxis CSS

Al usar llaves, se pueden usar espacios y saltos de linea de manera arbitraria. Eso si, todas las declaraciones terminan con un punto y coma ';'.

2.4.5. Selectores con nombre de elemento HTML

Todos los nombres de elementos HTML son selectores.

- p{...} selecciona todos los parrafos
- h_1{...} selecciona todos los encabezados de nivel 1.
- Etc.

Se espera una cantidad arbitraria de 0 o más elementos que serán selccionados.

2.4.6. Pseudo-clases

Las pseudo-clases se usan para definir/seleccionar un estado especial de los elementos. Por ejemplo:

- Enlaces que han visitado, o que aún no han sido visitados
- Dar un estilo diferente para los elementos cuando tienen el foco activado.
- Dar un estilo diferente cuando se pasa el mouse por encima del elemento.
- Seleccionar el primer/último elemento.
- Seleccionar elementos dependiendo de si están habilitados, vacíos, etc.

2.4.7. Combinar selectores

Los combinadores permiten seleccionar elementos en base a la relación que estos tienen en el árbol n-ario que forma el documento HTML.

- Selector de descendientes (espacio en blanco)
- Selector de hijos >
- Selector de hermano adyacente +
- Selector de hermano (no necesariamente adyacente) -

2.4.8. Selector de descendientes

Selecciona todos los elementos que están en la cadena de jerarquía especificada. Por ejemplo, si queremos especificar que todos los párrafos dentro de los elementos con clase 'tarea' van a ser de texto azul:

.tarea p{color:blue}

Si queremos los textos <i>en los párrafos dentro de el elemento con id 'HEADER'.

2.5. Javascript (JS)

2.5.1. JS en el lado del cliente

- Manipulación dinámica del documento HTML a través del DOM.
- Acceso muy límitado a los recursos del sistema operativo: Archivos, etc.
- Se ejecuta en el contexto del navegador web.

2.5.2. ¿JS o ECMA script?

- Hay muchas versiones de Javascript. En la práctica cada navegador soporta distintas características.
- ECMA script es una especificación estándar de un lenguaje 'tipo JS'
- El Javascript moderno se basa en las características de ECMAScript 6(ES6).

- Existen herramientas para 'transpilar' código en ES6 a versiones más primitivas.

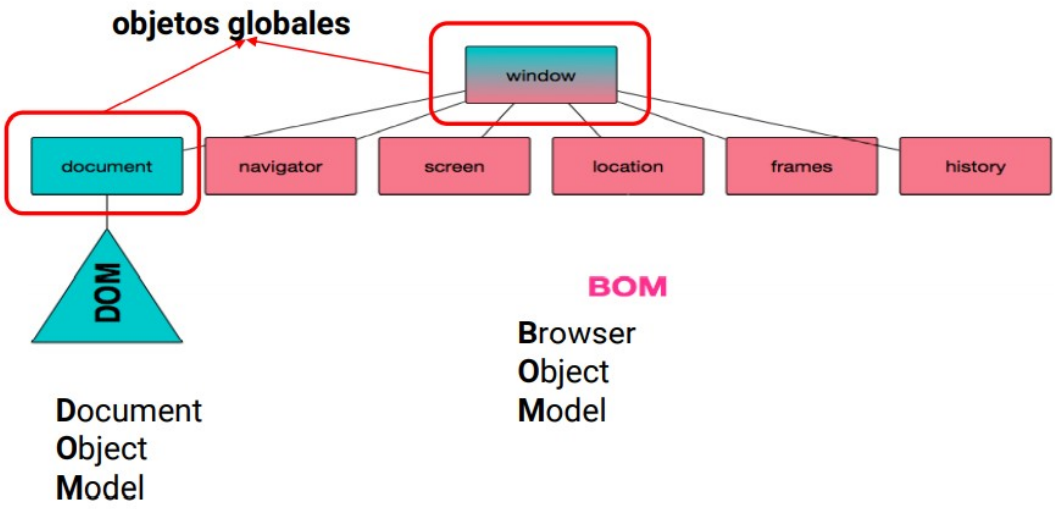


Figura 12: DOM & BOM: Manipulación programática

2.5.3. ¿Qué se puede manipular con JS?

La manipulación dinámica del documento con Javascript permite:

- Cambiar el contenido de los elementos HTML
- Cambiar los atributos de los elementos HTML
- Cambiar el estilo de los elementos (ya que el estilo es un atributo)
- Esconder y mostrar elementos HTML, manipulando el estilo display
- Una larga lista de etc, combinando lo anterior.

Figura 13: Cambiar dinámicamente hojas de estilo

2.5.4. Javascript y eventos del DOM

La manipulación programática del DOM está basada en eventos del DOM. La abstracción de su ejecución se puede leer como:

'**Cuando** ocurre un evento, **entonces** se ejecuta una función conocida como **callback** que recibe informaciónl de evento ocurrido'.

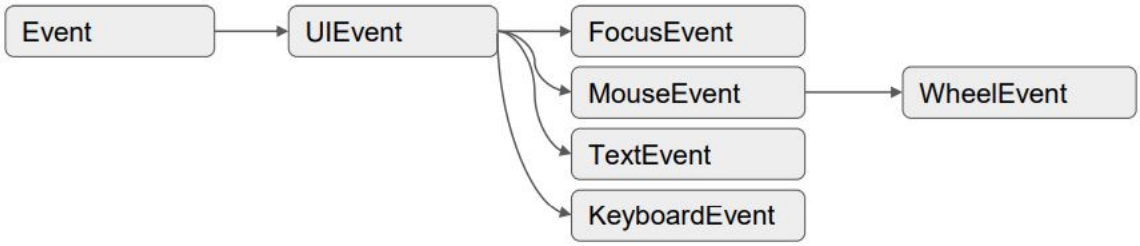


Figura 14: Jerarquía de eventos del DOM/UI

Esta es jerarquía simplificado de algunos tipos de eventos organizados por el DOM durante la ejecución de un documento HTML. Existen muchos más eventos relacionados con el navegador, formularios, objetos, etc.

Evento	Descripción
onclick / ondbclick	Se hace click en el elemento / Se hace doble click en el elemento
onmouseup / onmousedown	Se presiona un botón en el elemento / Se suelta un botón que estaba presionado en el elemento.
onmouseover / onmouseout	El puntero se está moviendo mientras está sobre un elemento / el puntero se mueve fuera del elemento, o fuera de uno de sus hijos.
onmousecenter / onmouseleave	El puntero se mueve dentro de un elemento / El puntero se mueve fuera de un elemento
oncontextmenu	Se hace click derecho en un elemento para abrir un menú contextual
onkeydown	Ocurre cuando el usuario está presionando una tecla.
onkeypress	Ocurre cuando el usuario presiona una tecla. Solo se disipa cuando la tecla es un caracter imprimible.
onkeyup	El puntero se está moviendo mientras está sobre un elemento / el puntero se mueve fuera del elemento, o fuera de uno de sus hijos.

Tabla 3: Eventos más comunes/Mouse Event

La tabla anterior tiene un orden de activación, el cual va de la siguiente forma:

- Onkeydown
- Onkeypress, si es un caracter imprimible
- Onkeyup, cuando se suelta la tecla.

Si la tecla se deja presionada, se dispara muchas veces onkeydown/onkeypress. Para ejecutar un segmento de código JS en un documento, se utiliza el tag <script>.

2.5.5. Manipulación programática de la página

Trata de obtener un objeto que representa la parte de documento con 'id' contenido

$$\text{c} = \text{Document.getElementById('contenido')}$$

En donde 'getElementById' se conoce como un selecto, se puede seleccionar por id, class, nombre de tag, etc. Por otro lado, tenemos:

$$\text{c.innerHTML} = \text{'<p>Hola contenido dinamico <\p>';}$$

Lo anterior, cambia el contenido HTML de C, mediante su atributo innerHTML modificando visualmente la página.

2.5.6. Eventos del ciclo de vida del DOM

window.onLoad

Se dispara cuando la página ya ha terminado de ser cargada. Esto es importante porque significa que todos los elementos del DOM ya existen, y pueden ser manipualdos.

```
<html>
  <head>
    <title>Javascript en el navegador</title>
    <script>alert("Hola Header!");</script>
  </head>
  <body>
    <script>
      alert("Hola Body!");
            window.onload = function() {
        c = document.getElementById("contenido");
        c.innerHTML = "<p>Hola contenido dinamico</p>";
        }
    </script>
    <div id="contenido"></div>
  </body>
</html>
```

2.5.7. Agregando scripts como recursos externos

En general no es deseable que un documento HTML contenga mucho código Javascript 'incrustado', porque eso juega en contra de la independencia de la modularidad de la aplicación. Lo usual es agregar detalles pequeños, y por ejemplo asignar los eventos *onclick* u otros a elementos HTML que deseen manipular . Pero, en vez de escribir el código directamente, se hace referencia a una función que está en un archivo externo.

Para incluir un archivo externo se usa el elemento 'SRC'

index.html

```
<html>
  <head>
    <title>Javascript en el navegador</title>
    <script type="text/javascript" src="my_script.js">
    </script>
  </head>
  <body>
    <div id="contenido"></div>
  </body>
</html>
```

my_script.j
s

```
window.onload = function() {
        c = document.getElementById("contenido");
        c.innerHTML = "<p>Hola contenido dinamico</p>";
}
```

2.5.8. Selectores del DOM

Como vimos en el ejemplo anterior (figura de la sección 2.5.7), gran parte del trabajo consiste en:

- Seleccionar el o los elementos que se desean modificar o consultar
- Cambiar los atributos de los elementos seleccionados en base a cálculos sencillos, o a llamadas a funciones definidas anteriormente.

Para seleccionar los elementos se trabaja de manera similar a CSS, utilizando métodos de selectores. Usando Javascript se pueden seleccionar elementos de distintas formas:

- Seleccionar por ID:

  document.getElementById('el id')

- Seleccionar por nombre:

  document.getElementByTagName('p')

- Seleccionar por clase:

  document.getElementByClass('tarea')

- Seleccionar por selector CSS:

  document.querySelectorAll('div a p')

2.5.9. Selectores DOM usando Jquery

JQuery es una librería que facilita la manipulación HTML usando Javascript. Entre sus muchas funcionalidades, provee una síntaxis uniforme para seleccionar elementos HTML. Todas las funcionalidades de selección/manipulación tienen la forma:

$('...').XXX

Dónde:

- $ es el objeto global JQuery
- $('...') es la selección de elementos

- .XXXX es la ejecución del método XXXX en los elementos seleccionados

document.getElementById('el id')	$('#El-ID')
document.getElementByTagName('p')	$('p')
document.getElementByClass('tarea')	$('tarea')
document.querySelectorAll('div a p')	Funciones anteriores y método FIND

Tabla 4: Selector vs Selector

3. Modelo de negocio

3.1. ¿Qué es un modelo de negocios?

Es una representación abstracta de una organización (textual o gráfica), en donde se muestran todos los conceptos relacionales de un negocio en una organización, lo que incluye los acuerdos financieros.

Es el portafolio central de productos o servicios que la organización ofrece y ofrecerá con base en las acciones necesarias, por lo que es importante diseñar e innovar en nuestros modelos de negocios.

3.2. ¿Por qué debemos innovar en nuestros modelos de negocios?

Vivimos en una civilización que está en constante cambio, por lo que en pocos años nuestro modelo de negocio puede quedar obsoleto y finalmente dejarnos fuera del mercado, sin contar que también existe la posibilidad de tener un competidor con un modelo de negocio más innovador al nuestro.

3.3. Caso Nokia

Hace 10 años, Nokia contaba con el 30 % de a cuota del mercado con una facturación de 30.000 millones de euros nacionales. Ahora bien, su actualidad es totalmente diferente, debido a que ha perdido el 80 % de su capitalización desde el año 2000.

3.4. Modelo Canvas

'Construimos para pensar y no pensamos para construir'

Nos permite utilizar el proceso de pensamiento de diseño:

- Observar y comprender a nuestros usuarios
- Idear tantos modelos de negocios como sea posible
- Seleccionar los más adecuados
- Crear prototipos rápidamente de varios modelos, y evaluar en el mercado si se comportan como esperamos

Ahora bien, el modelo se compone por:

- 4 áreas

- ¿Cómo?
- ¿Qué?
- ¿A quién?
- ¿Cuánto?

- 9 bloques
  - Socios claves
  - Actividades clave
  - Propuesta de valor
  - Relaciones con clientes
  - Canales
  - Segmentos de mercado
  - Estructura de costos
  - Fuentes de ingreso

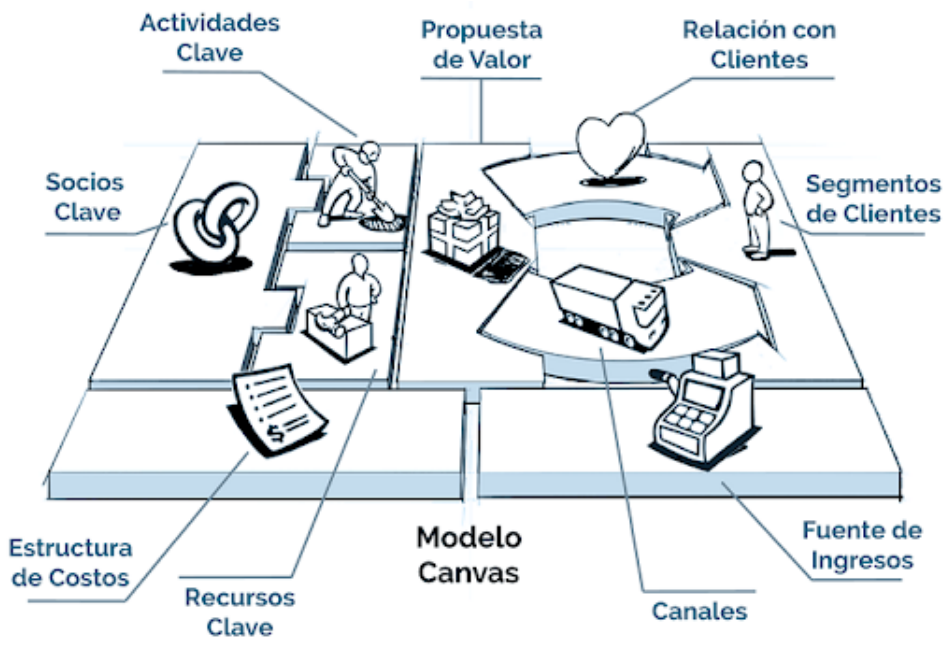


Figura 15: Modelo Canvas

4. Nociones Javascript, JSON y AJAX

4.1. Javascript

Javascript en el navegador:

- Posibilita la manipulación dinámica del DOM
- Tiene acceso muy limitado al hardware, los archivos, y en general los recursos del sistema operativo.
- El motor de ejecuciónde Javascript está dentro del navegador web.
- Se cuenta con una librería estándar.
- Se asume que el código que se muestra en este capítulo está siendo incluido y ejecutado en una página web.

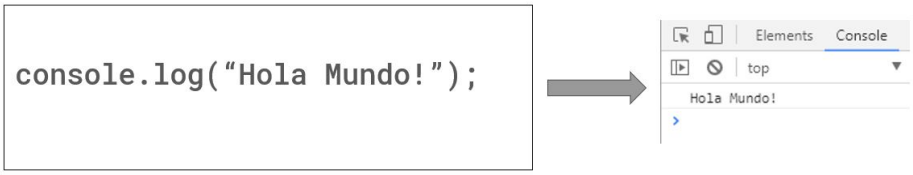
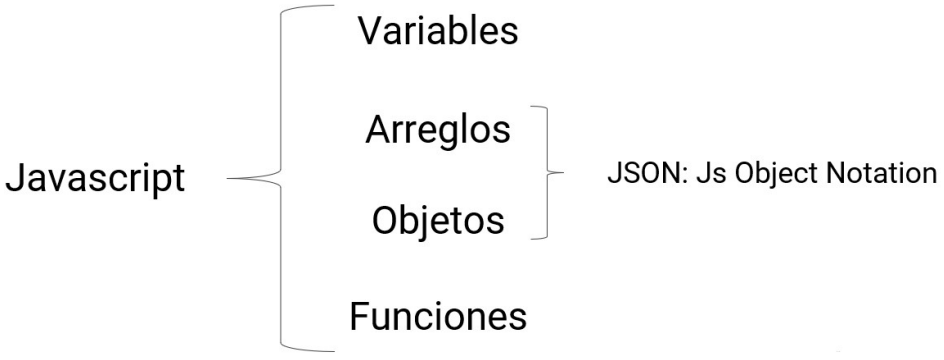


Figura 16: Hola Mundo JS!

Javascript es un lenguaje procedural, imperativo y orientado a objetos, el cúal presenta además una síntaxis prácticamente idéntica a C o Java. Hay objetos predefinidos siempre disponibles, por ejemplo *window, document, console*. Toda la salida de consola se muetra en la consola de las herramientas de desarrollo. Los elementos básicos de programación son:



Por otro lado, la declaración y asignación de variables es de la siguiente forma:

```
var nombre = “Felipe”;
var apellido = “Garcia”;
var edad = 29;
var casado = false;

// Concatenación tradicional
console.log(nombre + “ “ + apellido);

// Template strings
var msg = `${nombre} ${apellido} (${edad}) ${casado}`
console.log(msg);
```

Puede no estar disponible en navegadores antiguos

4.1.1. Arreglos e iteraciones

- **Iteración clásica**

  La representación de arreglos y la clásica iteración no difiere a lo utilizado en los lenguajes ya mencionados (a excepción que aquí no existen los punteros).

```
var notas = [7, 6.2, 5.5];

var suma1 = 0;
for(var idx = 0; idx < notas.length; idx++) {
    suma1 += notas[idx];
}
console.log(`Suma 1: ${suma1}`);
```

  - Cómoda síntaxis para arreglos literales, usando corchetes cuadrados [ ].

es un arreglo vacío

- [A, B, C] es un arreglo con los elementos A, B y C
- Los arreglos pueden contener valores de distintos tipos (cómo las listas en python).

  [1, "A", [ ]]

- La propiedad *Length* índica el largo del arreglo.

- **Iteración For-Each**

  Para representar un for-each, lo podemos hacer análogo al siguiente ejemplo:

```
var notas = [7, 6.2, 5.5];


var suma2 = 0;
notas.forEach(
    function(x) {
       suma2 += x;
});
console.log(`Suma 2: ${suma2}`);
```

  - El método *For Each* es otra forma de iterar. Aplica una función a cada elemento x. Pero, no se puede romper la iteración con un *break* o *return*.
  - La función definida dentro de notas (*function(x)*) se le llama función anónima.

- **Iteración For-In**

```
var notas = [7, 6.2, 5.5];

var suma3 = 0;
for(var idx in notas) {
    // Indices como strings... :/
    // NO USAR CON ARREGLOS!!
    console.log(typeof(idx));
}
```

- La expresión *for-in* itera sobre las llaves de un objeto.
- Si se usa en un arreglo, los 'indices' son strings, tales como '1', '2', etc.
- No usar con arreglos!!!, ni siquiera se garantiza que salgan en orden...
- El operador *type of* indica el tipo de su argumento, como string.

- **Iteración For-Of**

```
var notas = [7, 6.2, 5.5];

var suma4 = 0;
for(var nota of notas) {
        suma4 += nota;
}
console.log(`Suma 4: ${suma4}`);
```

- La expresión *for-of* itera sobre cada elemento del arreglo, en orden de ízquierda a derecha.
- Es la maner preferida de iterar por los elementos, a menos que por alguna razón se necesite conocer el índice.
- Funciona con otras colecciones: *strings, map, set, etc.*

- **Otros métodos**

```
var notas = [7, 6.2, 5.5];

notas.push(100);
console.log(notas);

var n = notas.pop();
console.log(notas);
console.log(n);

notas.slice(0,1);
console.log(notas);

notas.splice(0,1);
console.log(notas);
```

- *push:* Agrega un elemento al final del arreglo.
- *pop:* Retorna el último elemento, y además lo retira del arreglo.
- *slice:* Obtiene un segmento del arreglo, sin modificar el arreglo original.
- *splice:* Remueve un segmento del arreglo, retornando los elementos eliminados.

4.1.2. Objetos

- **Definición de objeto**

```
var persona = {
    nombre: “Felipe",
    apellido: “Garcia",
    edad: 29,
    casado: false
};

console.log(persona);
```

- En Javascript los objetos son diccionarios multiples.
- Cómoda síntaxis para objetos literales, usando llaves ({ })
- { A:1 } es el objeto con propiedad A cuyo valor es 1.
- Los nombres de las propiedades siempre son strings.
- Los valores pueden se cualquier cosa, menos ***undefined***.

- **Acceso a propiedades**

```
var persona = {
    nombre: “Felipe",
    apellido: “Garcia",
    edad: 29,
    casado: false
};

console.log(persona.nombre);
console.log(persona[‘apellido’]);
console.log(persona['e' + 'dad']);

console.log(persona.sueldo);
```

- La manera más sencilla de acceder a una propiedad es usando el operador punto.
- También se puede usar corchetes cuadrados, con el nombre de la propiedad como string.
- Usar corchetes es útil si el nombre de la propiedad se genera dinámicamente.
- Si no existe la propiedad, se retorna ***undefined***.

```
var persona = {
    nombre: “Felipe",
    apellido: “Garcia",
    edad: 29,
    casado: false
};

console.log(persona.sueldo || “zero”);

console.log(persona.sueldo.moneda); // TypeError

console.log(persona.sueldo && persona.sueldo.moneda)
```

- El operador || permite especificar un valor por defecto.
- También se puede usar corchetes cuadrados, con el nombre de la propiedad como string.
- Si intento acceder a una propiedad de algo ***undefined***, se genera un ***TypeError***.
- El operador && permite resguardar que algo esté definido antes de usarlo.

- **Actualizando propiedades**

```
var persona = {
    nombre: “Felipe",
    apellido: “Garcia",
    edad: 29,
    casado: false
};

persona.edad = 29;
console.log(persona);

persona.sueldo = 9999999999;
console.log(persona.sueldo);
```

- Los valores de las propiedades se actualizan usando asignación.
- Si la propiedad existe, el valor es reemplazado.
- Si la propiedad no existe, el objeto es extendido con dicha propiedad.

- **Borrando propiedades**

```
var persona = {
    nombre: “Felipe",
    apellido: “Garcia",
    edad: 29,
    casado: true
};

console.log(persona);

delete persona.edad;
console.log(persona);
```

  - Las propiedades se remueven utilizando el operador ***Delete***.

4.1.3. Funciones

- **Definición clásica**

```
function sumaNotas(notas) {
    var sum = 0;
    for(x of notas) {
        sum += x;
    }
    return sum;
}

var notas = [7, 6.2, 5.5];

console.log(sumaNotas(notas));
```

  - Usando ***Function*** se define una función con su nombre y el nombre de los argumentos.
  - Se retorna el valor utilizando ***return***
  - Se invoca como cualquier función.

- **Valores de primera clase**

```
var f = function(notas) {
    var sum = 0;
    for(x of notas) {
        sum += x;
    }
    return sum;
}

var notas = [7, 6.2, 5.5];

console.log(f(notas));
```

- Se puede usar ***Function*** para definir funciones anónimas.
- Las variables pueden contener funciones, se pueden pasar como argumentos, almacenar, etc.
- Esto se conoce como funciones de primera clase.

```
console.log( function(a,b) { return a+b; }(1,2) );
```

- Las funciones anónimas se pueden aplicar directamente
- No es necesario que una función tenga nombre.
- Aunque muchas veces si es conveniente que tenga nombre.

- **Métodos de objetos**

```
var persona = {
   nombre: “Felipe",
   saludar: function(n) {
    console.log(`Hola ${n} me llamo ${this.nombre}`);
   }
}

persona.saludar("Juanito");
```

- Los métodos de un objeto son las propiedades que son funciones.

- Los métodos pueden usar el nombre de 'this' para acceder a las propiedades del mismo objeto.
- Javascript tiene objetos basados en prototipos y no en clases.

- **Paso por valor y por referencia**

```
function actualizarSueldo(p,s) {
    p.sueldo = s;
    s = 10;
}

var persona = {};
var sueldo = 1000000;

console.log(persona);
console.log(sueldo);

actualizarSueldo(persona, sueldo);

console.log(persona);
console.log(sueldo);
```

- Los tipos básicos siempre se pasan por valor.
- Los objetos, arreglos y funciones siempre se pasan por referencia.

4.1.4. Alcance de las variables

El alcance de una variable es la región de un programa en el cuál es válida la asociación entre el nombre de la variables y su valor. En inglés, el concepto de alcance se llama *'scope'*.

- **Variables globales**

```
nombre = “Felipe”;

function saludar() {
   console.log(`Hola ${nombre}`);
}

saludar();
console.log(`Nombre: ${nombre}`);
```

- Se definen en el TOP-LEVEL del archivo sin usar *var*.
- Una vez definidas, pueden usarse en cualquier otra sección de código, bloque, función, etc. del mismo archivo.
- Es poco deseable utilizar variables globales.

- **Variables de módulo**

```
var nombre = “Felipe”;

function saludar() {
   console.log(`Hola ${nombre}`);
}

console.log(`Nombre: ${nombre}`);
```

- Se definen en el TOP-LEVEL del archivo sin usar *var*.
- Una vez definidas, pueden usarse en cualquier otra sección de código, bloque, función, etc. del mismo archivo.
- Es aceptable pero poco ideal utilizar variables de módulo.

- **Variables de función**

```
function saludar(msg) {
   var nombre = “Felipe”;
   console.log(`${msg} ${nombre}`);
}

saludar(“Hola”);

console.log(`Nombre: ${nombre}`);
```

- Se definen dentro de una función, como parámetro o cómo variables.
- No son visibles fuera de la función en la que son definidas.
- Puede resultar confuso, porque uno espera que tenga alcance de bloques.

```
var nombre = “Felipe”;

function saludar(n) {
   if (n > 3) {
      var nombre = “Juanito”;
   }
   console.log(`Hola ${nombre}`);
}

saludar(1);
saludar(4);
```

- Cuando usamos *var*, la declaración de la variable está disponible en toda la función, y no sólo en el bloque que la contiene
- Esta declaración está vigente hasta que se acaba el cuerpo de la función.
- La asignación de valores se mantiene según sea el flujo de control del programa.

```
function varLoop() {
  for(var i = 0; i < 10; i++) {
     console.log(i);
  }
  console.log(`i: ${i}`);
}

varLoop();
```

- Lo mismo pasa con las variables de una iteración.
- Su declaración está disponible en toda la función.

- **Problemas Global + Local**

```
var x = 5;

(function () {
  console.log(x);
  var x = 10;
  console.log(x);
})();
```

- Las variables declaradas usando *var* son sometidas a un proceso de hoisting.
- Un proceso de hoisting consiste en que con independencia de dónde esté la declaración de una variable, esta es movida al inicio del ámbito al que pertenece.
- Las definiciones se ponen al comienzo de la función con valor ***undefined***.
- Luego, las asignaciones funcionan sobre variables ya declaradas.

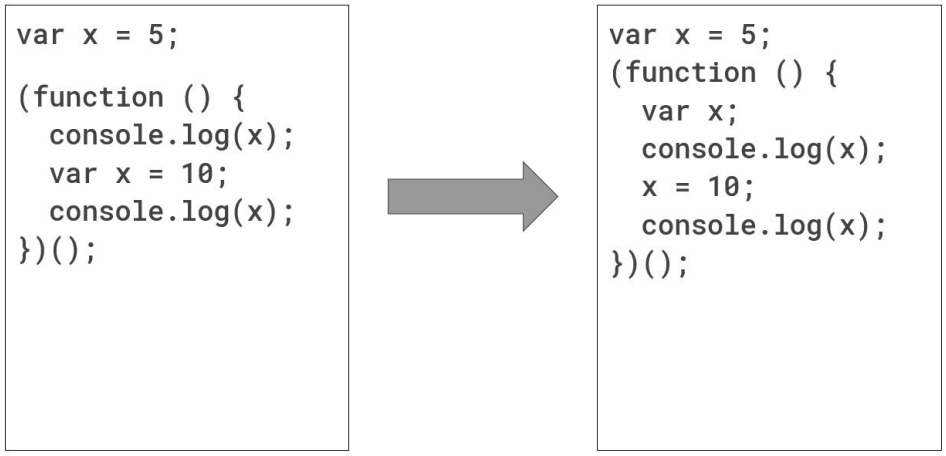


Figura 17: Proceso de Hoisting

- **Clausuras o closures**

```
var x = 5;
var f = function() {
   return x+1;
}

console.log(f());

{
   var x = 10;
   console.log(f());
}

x = 4;
console.log(f());
```

4.2. JSON (Javascript Object Notation)

Es un formato ligero para intercambio de datos el cual es fácil de leer y escribir para los humanos, como a su vez, es fácil de pensar y generar para

las máquinas. Se basa en un subconjunto de la síntaxis de javascript.

- Valores Literales:
  - Strings: 'Todo string literal es un valor string en Json'
  - Números:
    - Enteros (1,2,3,...)
    - Float (0.1, 0.02, ...)
    - Solo se usa formato decimal, es decir, no se usa ni octal ni hexadecimal
  - Booleanos
    - True
    - False
- Arreglos

  Se una la misma notación con los corchetes cuadrados y comas [].

  $$[1, 2, [], 'hola']$$

  En forma genérica

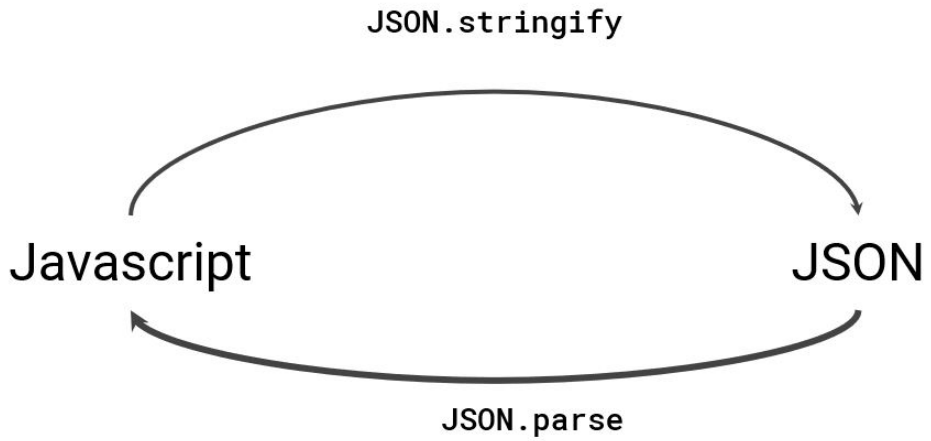
  $$\text{Arreglo Json} = \text{"[ ]"} | \text{"[}elementosJson\text{]"}$$

- Objetos

  Un objeto JSON se entiende como una colección no ordenada de pares nombre-valor, usando las mismas llaves que para objetos en JS {

  - { 'nombre': 'felipe', 'edad':29 }
  - { 'notas': $[7, 6, 5, 42]$ }
  - 'datos': { 'nombre': 'Juanito', 'edad':20 }

  Un objeto JSON es semánticamente distinto a un objeto Javascript, debido a que un objeto JSON no puede tener métodos ni tiene noción de *this*.



Ahora con todo lo mencionado anteriormente, ¿Cómo codificar valores javascript a JSON?

```
var holamundo = 'hola mundo!'
console.log(holamundo);
console.log(json.stringify(holamundo));
```

El objeto global JSON se encuentra disponible en la mayoría de los navegadores modernos. El método *json.stringify* transforma un valor javascript a su representación como un string en formato JSON. Además, la asignación de valores se mantiene según sea el flujo de control del programa.

```
var funstr = json.stringify(
    function(){
        return 1;
});
console.log(funstr);
```

Antes valore que no son transformables a JSON, el método retorna ***undefined***.

```
var persona = {
    nombre: 'Felipe',
    saludar: function(n){
        console.log('hola ${n}$ me llamo ${this.nombre}" );$
    }
}
```

Si un objeto tiene métodos y es transformado en JSON, el resultado sólo contendrá los atributos que sí son transformables.

Por otro lado, ¿Cómo decodificamos un objeto JSON en variables Javascript?.

```
var arregloJson = '[1, 2, 'hola', 'cuatro']';
var arreglojs = json.parse(arregloJson);
console.log(arregloJson);
console.log(arreglojs);
```

El método JSON.parse trata de traducir un string JSON en los valores javascript correspondientes. Si el string no es JSON válido se lanza una excepción.

4.3. AJAX (Asynchronous Javascript and XML)

4.3.1. ¿Por qué existe AJAX?

Sabemos que con Javascript podemos manipular dinamicamente el contenido de un documento HTML, pero lamentablemente estamos limitados a los valores constantes que podamos haber introducido de antemano, o al resultado de la evaluación de funciones locales.

4.3.2. ¿Qué se puede hacer con AJAX?

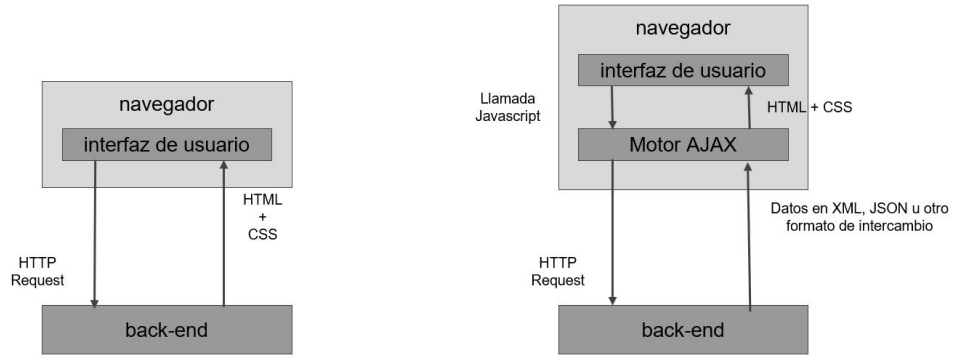
En estricto rigor, AJAX nos permite:

- Realizar un request HTTP arbitrario, con URL, parámetros, etc.
- Esperar la respuesta HTTP de manera asíncrona.
- Reaccionar a la respuesta HTTP mediante la ejecución de un callback

En base a lo anterior, podemos:

- Actualizar la interfaz de usuario sin recargar toda la página.
- Pedir/recibir datos al servidor después de la carga inicial de la página.

- Enviar datos al servidor de manera asíncrona y no invasiva.



AJAX funciona en base a la manipulación del objeto *XMLHTTPREQUEST*, el cual está disponible en todos los navegadores modernos.

```
var xhttp = new xmlhttprequest();
```

Desde lo anterior, tenemos los siguientes métodos para dicho objeto:

- Enviando request con AJAX

```
xhttp.open('GET', 'ajax_info.txt', True);
xhttp.send();
```

  - open(method, url, async) - Especifica el request que se hará, indicando su método, URL y si debe ser asíncrono o no.
  - send() - Envía el request al servidor, se usa para GET.
  - send(string) - Envía el request al servidor, con parámetros para el método POST

- Esperando la respuesta

  Una vez que se envía el request, el objeto *xmlhttprequest* emitirá e evento *onreadystatechange* según vata cambiando el estado del procesamiento del request. Se dispara para los estados 1 hasta el 4.

  El request pasa por cuatro estados:

- 0: request aún no inicializado
- 1: Se estableció conexión con el servidor
- 2: El request fue recibido
- 3: El request está siendo procesado
- 4: El request finalizó y la respuesta fue recibida.

- Invocación completa por llamada AJAX

```
var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function () {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("demo").innerHTML = this.responseText;
    }
}

xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

- Usando AJAX con JQuery

```
$.ajax("ajax_info.txt", { /* por defecto método GET */

   success: function(data) {
         /* callback caso exitoso */
    …
   },

   error: function() {
         /* callback caso no exitoso */
         ...
   }
});
```

AJAX sólo puede hacer llamadas a URL's que se encuentren en el mismo dominio que el sitio actual en el que se está ejecutando el código.

5. Nociones de PHP y programación Back-end

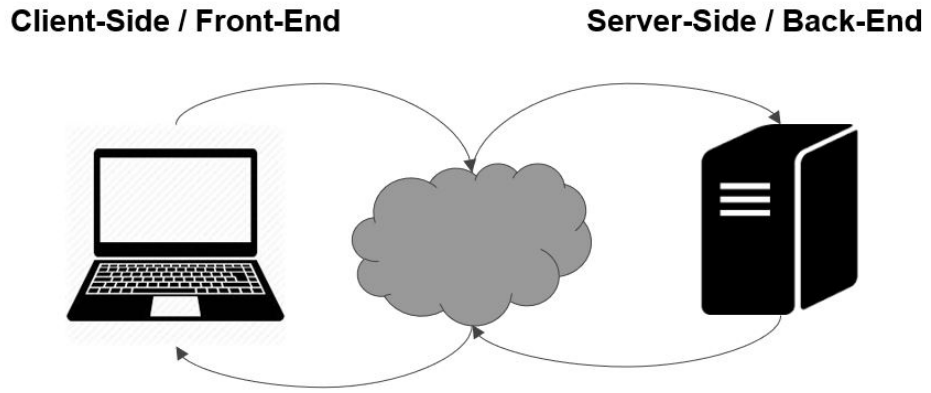


Figura 18: Stack de Tecnología web

5.1. Desarrollo en el Back-end

Cuando hablamos de back-end nos referimos a todo el código que se está ejecutando en el lado del servidor y, por contraste, no se ejecuta en el cliente. En la práctica se refiere a código que:

- Se conecta a bases de datos para segurar la persistencia de los datos.
- Puede ser implementado en practicamente cualquier lenguaje de programación
- No tiene ningún tipo de restricción en cuanto a las operaciones que puede realizar.

En general, el código del back-end es creado, instalado y configurado por los responsables finales de la aplicación web.

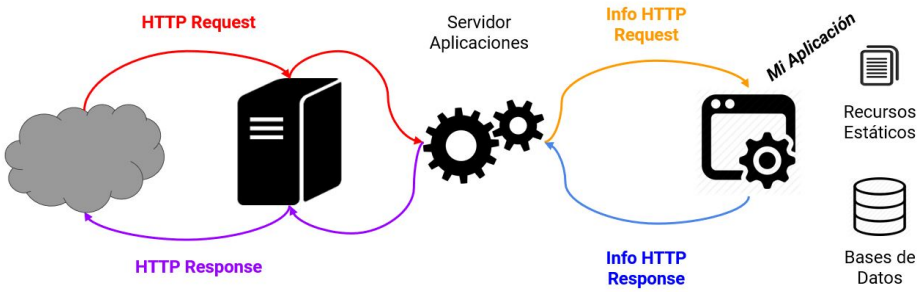


Figura 19: Ejecución de aplicaciones back-end

El servidor de aplicaciones

- Provee el contexto del request en un formato accesible para *mi-aplicacion*, típicamente como una estructura de datos.
- Traduce la respuesta de la aplicación de un *htppResponse*

Figura 20: Tecnologías esenciales para el back-end

5.2. Hypertext Pre Processor (PHP) cómo lenguaje de programación enfocado al back-end

5.2.1. PHP en el servidor

Así como en Javascript fue creado originalmente para trabajar en el front-end, PHP fue concebido para el desarrollo simple de aplicaciones en el back-end.

- PHP es fácil de instalar y de ejecutar (por ejemplo, con XAMPP),
- PHP permite una implementación fácil y rápida, de sesiones y cookies.
- PHP permite acceso directo y rápido a información sobre el request y el servidor en el que se ejecuta a través de las variables superglobales.
- La libería estándar de PHP es muy amplia y abarca gran cantidad de escenarios típicos de desarrollo web.

5.2.2. Instalación XAMPP y carpeta $DocumentRoot

Tras la instalación de XAMPP y una vez que se está ejecutando, las aplicaciones están disponibles en una carpeta que se conoce de manera abstracta como $DocumentRoot. En la práctica en una instalación típica:

$DocumentRoot = C:/XAMPP/HTDOCS

Al acceder a Http://Localhost/ estamos accediendo a la raíz de $DocumentRoot. Los sub-elementos de la URL corresponden a sub-carpetas de $DocumentRoot, por ejemplo, Http://Localhost/app corresponde a $DocumentRoot/app .

5.2.3. PHP y el contenido mixto

El código de un programa PHP está delimitado por las etiquetas <?php y ? >. La función echo envía un string a la salida/respuesta. Todo el texto que no esté entre estas etiqueas.

- Es ignorado por el intérprete php
- Es agregado inmediatamente a la salida de texto que es la respuesta que se le envía al navegador.

Esto es conveniente para generar HTML dinámicamente.

```php
<html>
    <head>
        <title>Hola PHP!</title>
    </head>
    <body>
    Esto es un archivo PHP
    Hoy estamos a:
    <?php
        echo date('d/m/Y', time());
        echo " Y 2+2 = ";
        echo 2+2;
    ?>
    </body>
</html>
```

Figura 21: Hola mundo - PHP

5.2.4. Nociones básicas - Como lenguaje de programación de propósito general

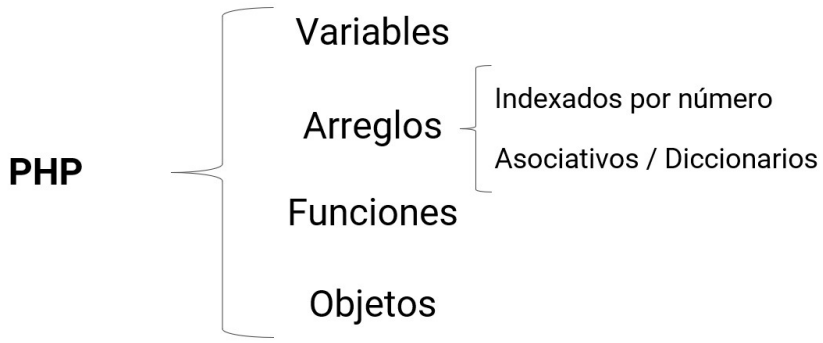


Figura 22: Elementos básicos de programación

- **Variables**

$DocumentRoot/hola-php/vars.php

```
<?php
$nombre = “Felipe”;
$apellido = “García”;
$edad = 32;
$casado = true;

// Concatenación tradicional
echo $nombre . $apellido;

// Interpolación de strings
$msg = “{$nombre} {$apellido} ({$edad}) {$casado}”;
echo “Interp: “ . $msg;
?>
```

- Se definen como nombres con un $ al comienzo.
- PHP tiene un sistema de tipos dinámicos.
- Los tipos básicos son:
  - String
  - Integer
  - Float
  - Boolean
  - Array
  - Object
  - Null
  - Resource
- Las operaciones ya están definidas (aritméticas, comparación, etc).

- **Arreglos For**

$DocumentRoot/hola-php/arrays1.php

```
<?php
$notas = array(7, 6.2, 5.5);

$suma1 = 0;
for($idx = 0; $idx < count($notas); $idx++) {
    $suma1 += $notas[$idx];
}

echo "Suma 1: {$suma1}";
?>
```

- Se usa la función array para construir arreglos
- Array() es el arreglo vacío.
- Array(A, B, C) es el arreglo con elementos A, B y C.
- Los arreglos pueden contener valores de distintos tipos:

  array(1, array('A', True))

- La función count() obtiene la cantidad de elementos del arreglo.

- **Arreglos for each**

`$DocumentRoot/hola-php/arrays2.php`

```
<?php
$notas = array(7, 6.2, 5.5);

$suma2 = 0;
foreach($notas as $nota) {
    $suma2 += $nota;
}

echo “Suma 2: {$suma2}”;
?>
```

- La expresión foreach itera sobre cada elemento del arreglo, ordenado de izquierda a derecha.
- Es la manera preferida de iterar por los elementos, a menos que por alguna razón se necesite conocer el índice.
- Cómo los diccionarios son arreglos asociativos, el uso de foreach también aplica sobre ellos.

- **Funciones arreglos**

$DocumentRoot/hola-php/arrays3.php

```
<?php
$notas = array(7, 6.2, 5.5);
print_r($notas); echo "<br/>";

array_push($notas, 100);
print_r($notas); echo "<br/>";

$notas[] = 200;
print_r($notas); echo "<br/>";

$n = array_pop($notas);
print_r($n); echo "<br/>";
print_r($notas); echo "<br/>";

print_r(array_slice($notas, 0, 2)); echo "<br/>";
$m = array_splice($notas, 0, 2);
print_r($m); echo "<br/>";
print_r($notas); echo "<br/>";
?>
```

- Array_push agrega u elemento al final de arreglo.
- Array_pop retorna el último elemento, y lo remueve del arreglo.
- Al asignar a $notas[ ] también se agrega al final del arreglo.
- Array_slice retorna un segmento del arreglo, pero sin modificarlo.
- Array_splice retorna un segmento del arreglo, y lo remueve.
- print_r imprime un valor de manera legible para humanos

- **Arreglos asociativos**

$DocumentRoot/hola-php/arrays4.php

```
<?php

$persona = array();
$persona[“nombre”] = “Felipe”;
$persona[“apellido”] = “García”;
$persona[“casado”] = true;
$persona[“edad”] = 33;

print_r($persona);
?>
```

- Los arreglos pueden ser indexados por números enteros.
- Los arreglos también pueden ir indexados por strings.
- Un arreglo asociativo es un arreglo indexado por strings, y que en la práctica se comporta como un diccionario.
- En estricto rigor, todos los arreglos son asociativos, unos con claves de tipo entero y otros con claves de tipo string.

- **Funciones clásicas**

$DocumentRoot/hola-php/fun1.php

```
<?php
function sumaNotas($notas) {
    $suma = 0;
    foreach($notas as $nota) {
        $suma += $nota;
    }
    return $suma;
}

$notas = array(7, 6.2, 5.5);
echo sumaNotas($notas);

?>
```

- Usando function se define una función con su nombre de los parámetros.
- Se retorna el valor usando *return*
- Si no se retorna nada, no se usa *return*
- Se invoca como cualquier función.

- **Función primera clase**

$DocumentRoot/hola-php/fun2.php

```
<?php

$f = function($notas) {
$suma = 0;
    foreach($notas as $nota) {
        $suma += $nota;
    }
    return $suma;
}

$notas = array(7, 6.2, 5.5);
echo $f($notas);
?>
```

  - En PHP también se puede usar funciones anónimas. Aúnque en la práctica no se usan tanto...
  - Las variables por tanto, pueden tener valores-funciones y se pueden psar como argumentos, etc.

- **Alcance de variables**
  - **Globales vs Locales**

```
<?php

$nombre = “Felipe”;
function saludar() {
  echo “Hola {$nombre}”;
}

saludar();

?>
```

- Este código produce un error.
- Esto es porque por defecto en php cuando se hace referencia a una variable dentro de una función, se asume que se refiere siempre a una variable local.
- En este caso, la variable local $*nombre* no existe.
- Y no se ocupa inmediatamente la variable global.

```
<?php

$nombre = “Felipe”;
function saludar() {
  global $nombre;
  echo “Hola {$nombre}”;
}

saludar();

?>
```

- Debemos indicar explícitamente que $*nombre* hace referencia a una variable global.
- Para esto, se usa la palabra clave *global*.

- **Bloques y funciones**

```php
<?php

function saludar($n) {
  if ($n > 3) {
     $nombre = “Felipe”;
  }
  echo “Hola {$nombre}<br/>”;
}

saludar(1);
saludar(4);

?>
```

- En PHP no respeta tampoco el alcance de bloque.
- Las funciones tienen sóll un nivel de alcance, alcance de función.
- Por otro lado, en PHP no se puede 'usar' una variable sin haberla declarado antes. Evidentemente, no va a existir y habrá un error.

5.3. Variables superglobales (como acceder a la información sobre el request)

5.3.1. ¿Qué son las variables superglobales?

- Son variables que están siempre definidas durante la ejecución.
- Se puede acceder a ellas desde cualquier lugar del programa: métodos, clases, funciones, etc.
- Su propósito es dar información sobre el contexto de ejecución del programa actual.

5.3.2. ¿Cuáles son las variables superglobales?

- $GLOBALS
- $_SERVER

- $_REQUEST
- $_POST
- $_GET
- $_SESSION
- $_COOKIES
- $_FILES
- $_NEW

```php
<?php

$nombre = “Felipe”;
function saludar() {
  echo “Hola ” . $GLOBALS[“nombre”];
}

saludar();

?>
```

Figura 23: Supervariable $GLOBALS

- Arreglo asociativo que almacena todas las variables disponibles en el contexto actual de ejecución del programa
- Se puede usar como complemento o alternativa a la palabra clave GLOBAL.

```php
<?php

print_r($_SERVER);

?>
```

Figura 24: Supervariable $_SERVER

- Arreglo asociativo que almacena información sobre el contexto de ejecución del programa actual.
- En particular, datos sobre el servidor y los headers HTTP recibidos
- Los headers tienen nombre **'HTTP_XXXX'** en las llaves del arreglo.

```php
<?php

print_r($_REQUEST);

?>
```

Figura 25: Supervariable $_REQUEST

- Arreglo asociativo que almacene información sobre el request recibido y que se está procesando.
- Combina los valores de $_GET, $_POST, y $_COOKIES.
- Es mejor trabajar con esas otras supervariables por separado.

```
http://localhost/hola-php/get.php
http://localhost/hola-php/get.php?id=10&hola=saludo
http://localhost/hola-php/get.php?param=valor
```

```
<?php

print_r($_GET);

?>
```

Figura 26: Supervariable $_GET

- Arreglo asociativo que almacena los parámetros recibidos mediante el método GET
- Esto incluye parámetros del query string
- También campos de formularios enviados con GET.

```
http://localhost/hola-php/post.php
http://localhost/hola-php/post.php?id=10&hola=saludo
http://localhost/hola-php/post.php?param=valor
```

```
<?php

print_r($_POST);

?>
```

Figura 27: Supervariable $_POST

- Arreglo asociativo que almacena los parámetros recibidos durante el método POST.

- Se usa principalente para pasar valores con formularios mediante el método POST.
- No considera valores en el Query String

```php
<?php

if($_SERVER['REQUEST_METHOD'] === "GET") {

    if(isset($_COOKIE["nombre"])) {
        echo "Hola {$_COOKIE["nombre"]}!";
    } else { ?>
        <form method="POST" action="cookie.php">
            <input type="text" name="nombre"
                   placeholder="¿Cual es tu nombre?"/>
            <input type="submit" value="Enviar"/>
        </form>
    <?php
    }
} else if($_SERVER['REQUEST_METHOD'] === "POST") {
    setcookie("nombre", $_POST["nombre"]);
    header("Location: cookie.php");
}

?>
```

Figura 28: Manejo de cookies - setcookie $_COOKIE

- **setcookie** envía en la respuesta el header para que el cliente guarde el valor en la cookie
- El valor estará disponible la próxima vez que se reciba un request.
- Cuando la cookie es enviada por el cliente, aparece en la variable superglobal $_COOKIE.

5.4. Manejo de sesiones

Una sesión es cuando el servidor almacena información asociandola a una llave única de acceso. Si el cliente envía la llave de acceso, el servidor podrá 'recordar' y utilizar los datos almacenados.

La 'metáfora del casillero' es bastante acertada:

- Es una forma de almacenamiento temporal
- Si perdemos la llave, perdemos las cosas guardadas

- Luego de un tiempo sin uso, se limpiarán los casilleros
- Si alguien roba la llave, tendrá acceso a lo que está guardado

5.4.1. Iniciando sesión en PHP (*sesion_ start())*

```php
<?php

session_start()
print_r($_SESSION);

?>
```

- ***session_ start*** inicia o continúa la sesión establecida con el usuario.
- Internamente esto se realiza mediante la cookie **PHPSESSID**.
- Una vez que la sesión está iniciada, tenemos acceso a la variable superglobal $_SESSION.
- Toda la información de la sesión debemos almacenarla y recuperarla desde $_SESSION.
- En este sentido, las sesiones no involucran necesariamente un usuario y una contraseña.

5.4.2. Ejemplo de sesión

```php
<?php
// archivo sesion1.php

session_start();

// Se usa un valor especial solo
// disponible en la sesionn iniciada
if(isset($_SESSION["user"])) { ?>

   <h1>Hola <?php echo $_SESSION["user"]; ?></h1>
   <a href="logout.php">Cerrar Sesion</a>

<?php } else { ?>

   <form method="POST" action="login.php">
     <input type="text" name="user" placeholder="usuario"/>
     <input type="password" name="password" placeholder="password"/>
     <input type="submit"/>
   </form>

<?php } ?>
```

Figura 29: Ejemplo de sesión con autentificación.

- ***session_ start*** se usa para iniciar o continuar la sesión.
- Se considera que el usuario tiene sesión abierta si existe el valor $_SESSION['user']
- Este valor sólo debe ser posible de almacenar en la sesión desde la funcionalidad de login.
- Las páginas pueden mostrar contenido según el usuario haya ingresado o no al sistema.

```php
<?php

session_start();

// Se usa un valor especial solo
// disponible en la sesion iniciada
if(! isset($_SESSION["user"])) {
    $user = $_POST["user"];
    $pwd  = $_POST["password"];

    // Revisar pwd ....

    // Guardamos usuario en la sesion
    $_SESSION["user"] = $user;
}

// Hacemos redireccion
header("Location: session1.php");

?>
```

Figura 30: Ejemplo de sesión con autentificación - login.

- ***session_start*** se usa para iniciar o continuar la sesión.
- Para iniciar sesión verifico primero que no haya iniciado sesión
- Obtengo el usuario y el password entregados, y si se cumple la validación de usuario/password.
- Entonces, se agrega el valor 'mágico' a $_SESSION para considerar que el usuario ha ingresado al sistema.

```php
<?php

session_start();

if(isset($_SESSION["user"])) {
   session_destroy();
}

header("Location: session1.php");

?>
```

Figura 31: Ejemplo de sesión con autentificación - login.

- ***session_ start*** se usa para iniciar o continuar la sesión.
- Se verifica si el ususario efectivamente ha iniciado sesión.
- Se usa ***session_ destroy*** para eliminar la sesión PHP, la cookie asociada, y los datos almacenados.

6. Patrón de diseño Modelo-Vista-Controlador (MVC)

6.1. ¿Cómo se ejecuta una aplicación web?

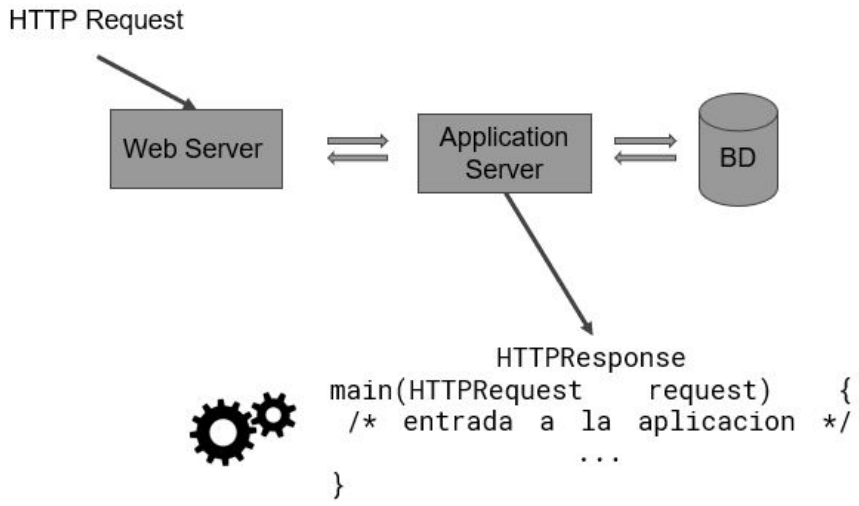


Figura 32: Representación ejecución de una aplicación web

Siempre es un request HTTP el que gatilla la ejecución por parte de la aplicación en el contexto del servidor de aplicaciones. En la versión más primitiva, el programa 'principal' tiene aceso a la información del request, en forma de strings disponibles en el contexto o ambiente de la ejecución.

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {

  char *queryString = NULL;
  queryString = getenv("QUERY_STRING");

  printf("Content-Type: text/plain;charset=us-ascii\n\n");
  printf("Hello world\n\n");
  if(queryString) {
     printf("Query String: %s\n\n", queryString);
  }

  return 0;
}
```

El protocolo CGI especifica una manera común de comunicar información entre el servidor web y una aplicación web.

Se utilizan las 'variables de entorno' del sistema operativo para traspasar informacion tal como el query string y otros.

Esto nos permite en la práctica poder desarrollar aplicaciones web en cualquier lenguaje, por ejemplo en C.

6.1.1. Problemas

Las aplicaciones web por lo general se usan para representar y manipular un **modelo o lógica de negocios** que forma el núcleo de la aplicación.

Las aplicaciones son utilizadas por **usuarios con distintos roles y preferencias**. Incluso, un mismo usuario puede necesitar distintas maneras de interactuar con la información.

Es crucial estructurar la aplicación para que pueda soportar con facilidad estas características.

6.2. Patrón de diseño MVC

6.2.1. Definición

Un patrón de diseño es:

'Una solución general y repetible a problemas que ocurren comúnmente en el diseño del software'.

El modelo vista controlador es un patrón de diseño para la implementación modular de interfaces de usuario. Fue creado/diseñado en la década de los 70 para el diseño de interfaces en la plataforma *Smalltalk*.

Fue adaptado posteriormente para su uso en aplicaciones web, aumentando la **escalabilidad y separación de responsabilidades** en este tipo de aplicaciones.

- **El modelo**
  - Se encarga de la lógica de negocios.
  - Contiene los datos relevantes de la aplicación.
  - Encapsula el estado interno de la aplicación.
  - Provee una interfaz que expone la funcionalidad de la aplicación.
  - En la práctica, consiste en un modelo de clases, interfaces y las relaciones que entre ellas se definan, y que forman el núcleo de la aplicación.
  - El modelo no sabe que se le está mostrando al usuario.
- **Las vistas**
  - Se encarga de renderizar la información obtenida desde el modelo.
  - Son (o debería ser) objetos que representan la interfaz del usuario.
  - Se encarga de recibir la entrada del usuario y dirigirla al controlador.
  - En general, cada vista tiene asociado un controlador específico.
  - Debe existir un mecanismo que propaga los cambios desde el modelo hacia los controladores e interfaz de usuario, para asegurar la consistencia.
  - En web, las vistas son HTML renderizado en el navegador.
- **Los controladores**

- Se encargarán de definir el comportamiento de la aplicación.
- Son como el 'pegamento' entre la interfaz de usuario y el modelo
- Traducen las acciones de la interfaz en operaciones conretas a realizar sobre el modelo
- Un controlador puede manejar una o muchas vistas, determinando dinámicamente qué vista presentar al usuario.
- El controlador ignora cómo se despliegan las vistas al usuario, y cuáles son los efectos de las acciones en el modelo.
- El programador debe conocer todo lo mencionado.

6.2.2. Beneficios y dependencias

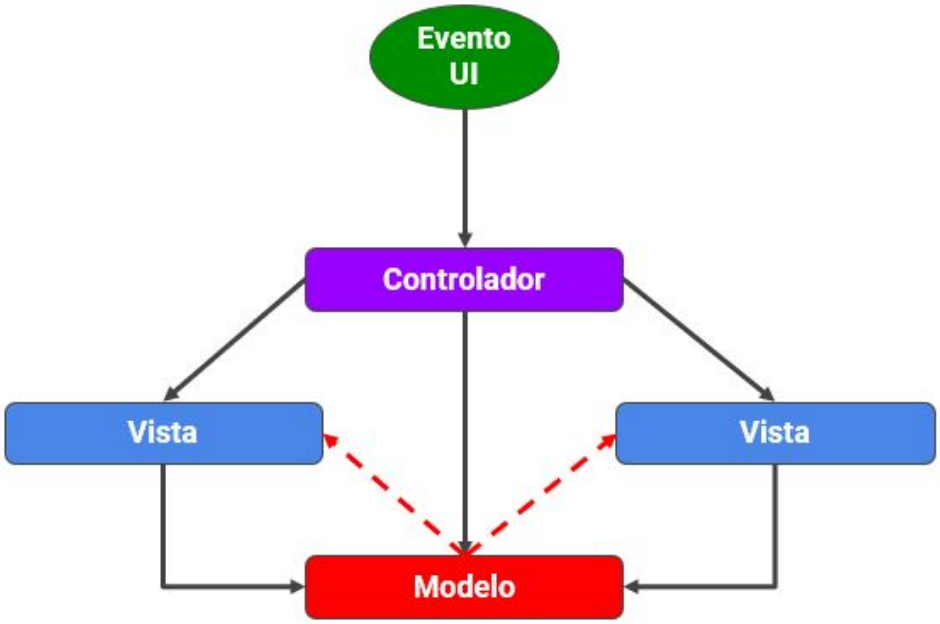


Figura 33: Flujo de ejecución en MVC

- De color verde está representada la generación de un evento en la UI, y se transmite con dirección al controlador.
- Después de la transmisión del evento, el controlador hace cambios en el modelo y/o en una(s) vista(s).

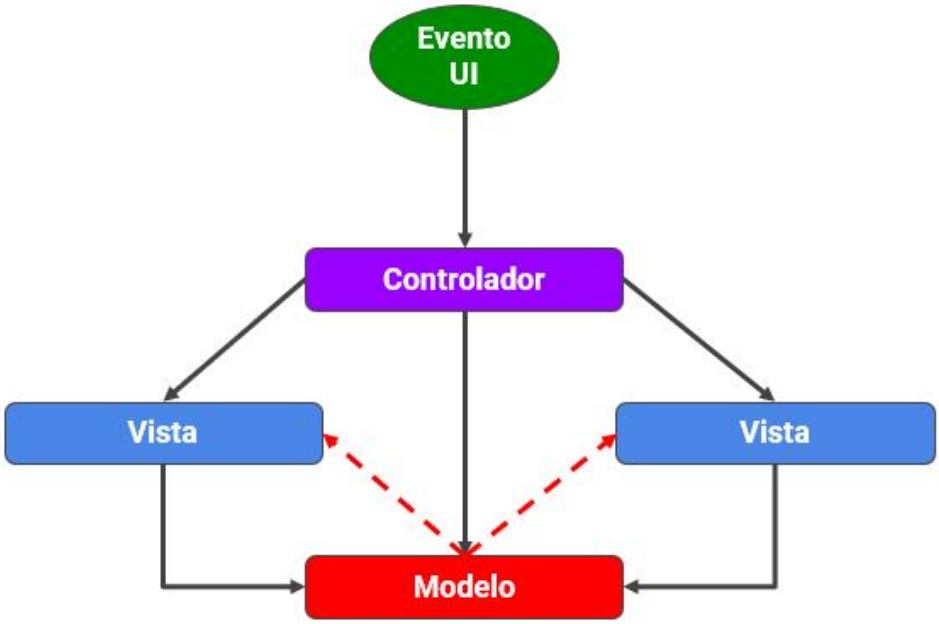
- A continuación, las vistas obtienen datos desde el modelo.
- Para así finalmente, el modelo actualiza las vistas cuando los datos cambian.

Las vistas y los controladores dependen del modelo, sin embargo, este mismo no depende ni de las vistas ni de los controladores. En algunas plataformas, la vista y el controlador van representados en un mismo objeto.

Los beneficios que este modelo conlleva incluye:

- **Soporte para múltiples vistas:** Gracias a que las vistas están separadas del modelo, se pueden crear diversas vistas diferentes sin afectar al modelo.
- **Adaptación al cambio:** Gracias a que el modelo no depende de las vistas, la rápida evolución de las interfaces de usuario no afecta al núcleo de la aplicación.
- **Costos de cambios muy frecuentes:** Si los cambios son muy grandes o muy frecuentes, finalmente se terminan modificando todos los elementos.
- **Complejidad:** El patrón introuce mayor indirección y un enfoque más orientado a eventos, lo que puede ser más complejo.

6.2.3. Ajustando la terminología Web



- En el evento UI, el usuario genera un **request HTTP**, sea por un link, formulario o invocación AJAX.
- El controlador recibe el request con sus datos y parámetros, e invoca métodos del modelo para instanciar una vista, la que retorna como response HTTP.
- Las vistas se generan dinámicamente usando **motores de plantillas** o templates.
- En general, el modelo no tiene cómo actualizar las vistas.

6.3. Todo Listo! Una aplicación web con MVC

La aplicación 'Todo listo!"maneja una lista de tareas donde un usuario puede desear:

- Ver el listado actual de tareas por realizar
- Visualizar un calendario con la cantidad de tareas creadas y finalizadas cada día y cada mes.

Además, un administrador podría.

- Visualizar cantidad de tareas según distintos filtros, mostrando solo la cantidad, pero no el contenido de las tareas.

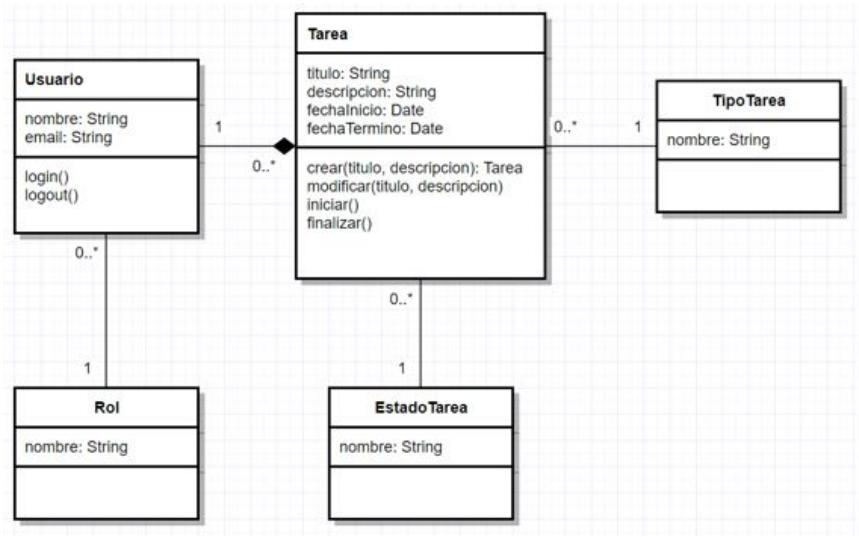


Figura 34: Todo Listo! - Modelo

Figura 35: Interfaz de usuario - Login

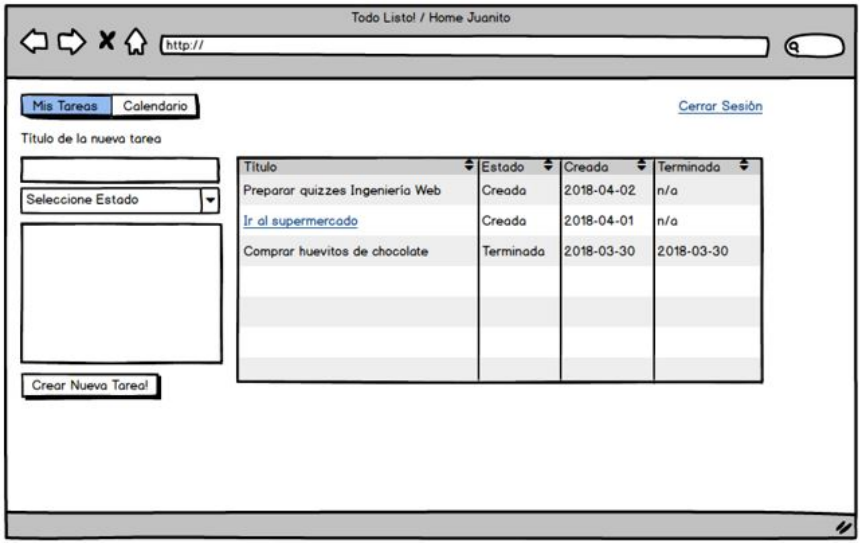


Figura 36: Interfaz de usuario - Listado de tareas

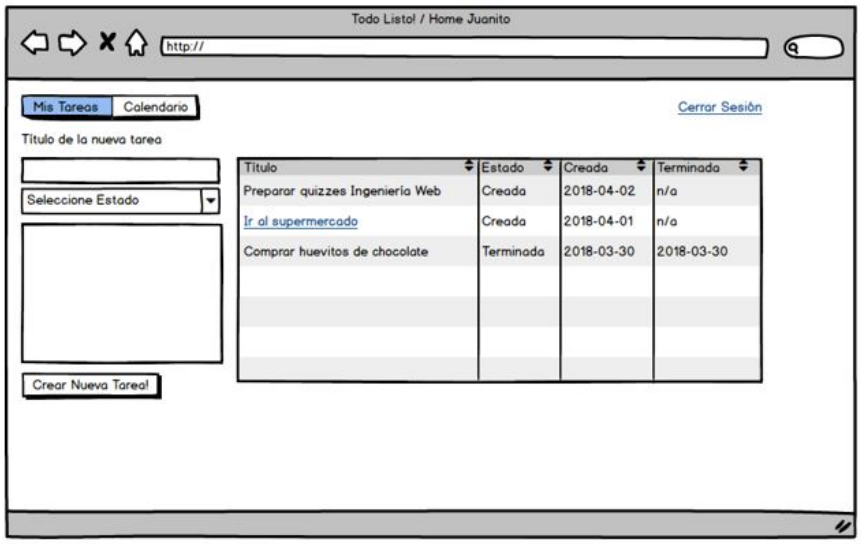


Figura 37: Interfaz de usuario - Editar tareas

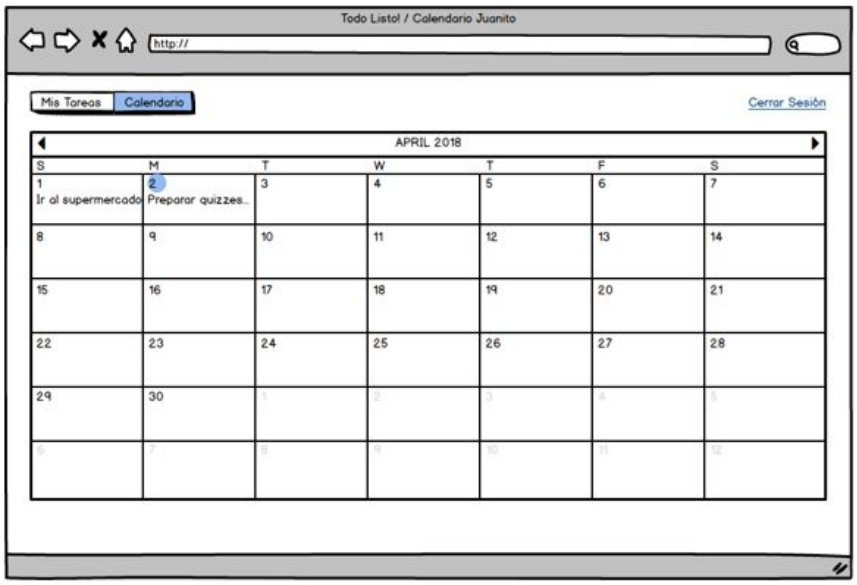


Figura 38: Interfaz de usuario - Vista Calendario

6.3.1. Diseño en base a MVC

Debe definir las rutas o URLS que vamos a implementar, y asociar cada URL a una vista específica, en dónde cada vista debe ser asociada a un controlador específico. La URL debe contener los parámetros necesarios para poder llenar la vista de manera satisfactoria, por ejemplo, id de la tarea.

Se debe considerar la información contextual que requiere la vista, por ejemplo, que un usuario específico haya iniciado sesión. Los formularios también deben ir mapeados a URL's que realizarán el procesamiento.