Introducción a la Programación Web

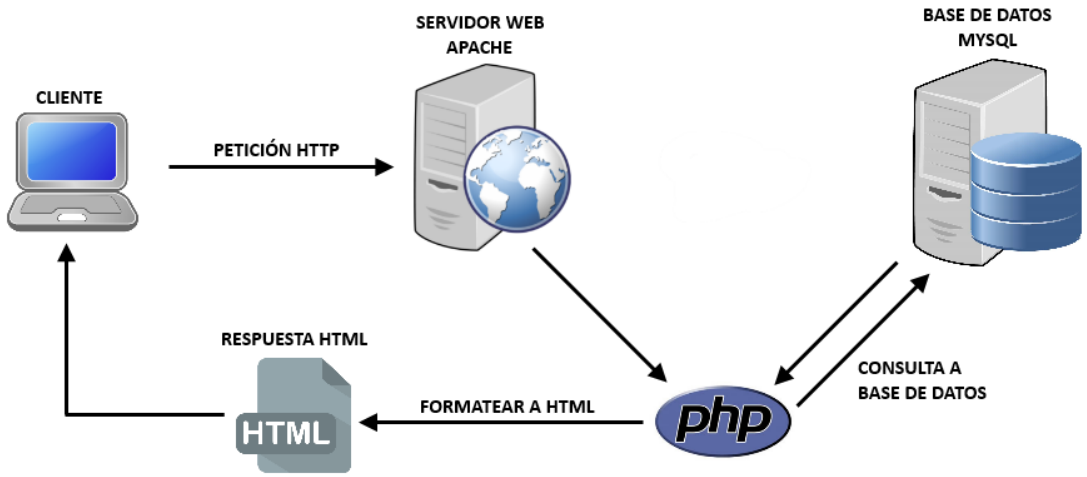
❖ Arquitectura Cliente-Servidor

La **arquitectura cliente-servidor** es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores , y los demandantes, llamados clientes.

En nuestro caso el cliente seria por ejemplo el navegador que esté usando el usuario para ejecutar el software, a través de esta interfaz haría una petición al servidor que es quien procesa esa petición.

El servidor procesa esa información y dependiendo de la tarea ejecutará una acción en forma de cálculos, acceso a BD para consultar datos etc. Después esta información es devuelta al cliente.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, los servidores de BD etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.



❖ Servidor Web

Un **servidor web** es un software que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta. El código recibido por el cliente es renderizado por un navegador web.

Para la transmisión de todos estos datos suele utilizarse algún protocolo. Generalmente se usa el protocolo **HTTP** para estas comunicaciones, perteneciente a la capa de aplicación del modelo OSI.

Al protocolo HTTP se le asigna habitualmente el puerto TCP 80. Las peticiones al servidor suelen realizarse mediante HTTP utilizando el método de petición **GET** , en el que el recurso se solicita a través de la url al servidor Web.
GET /index.html HTTP/1.1 HOST: www.host.com

En la barra de URL de un navegador cualquiera, la petición anterior sería análoga a la siguiente dirección Web:

www.host.com/index.html

El segundo tipo de petición HTTP más utilizado es **POST**. Los datos a enviar al servidor se incluyen en el cuerpo de la misma petición con las cabeceras HTTP asignadas correspondientemente respecto al tipo de petición.

Generalmente se asocia con los formularios web en los que los datos suelen ser cifrados para enviarlos de manera segura al servidor.

A modo de ejemplo, al teclear una web cualquiera en nuestro navegador, éste realiza una petición HTTP al servidor de dicha dirección. El servidor responde al cliente enviando el código HTML de la página; el cliente, una vez recibido el código, lo interpreta y lo exhibe en pantalla.

Como vemos con este ejemplo, el cliente es el encargado de interpretar el código HTML, es decir, de mostrar las fuentes, los colores y la disposición de los textos y objetos de la página; el servidor tan sólo se limita a transferir el código de la página sin llevar a cabo ninguna interpretación de la misma.

Además de la transferencia de código HTML, los Servidores web pueden entregar aplicaciones web. Estas son porciones de código que se ejecutan cuando se realizan ciertas peticiones o respuestas HTTP. Hay que distinguir entre:

- ***Aplicaciones en el lado del cliente***

El cliente web es el encargado de ejecutarlas en la máquina del usuario. Por ejemplo aplicaciones **javascript**: el servidor proporciona el código de las aplicaciones al cliente y éste, mediante el navegador, las ejecuta.

Es necesario, por tanto, que el cliente disponga de un navegador con capacidad para ejecutar aplicaciones (también llamados **scripts**).

- ***Aplicaciones en el lado del servidor***

El servidor web ejecuta la aplicación; ésta, una vez ejecutada, genera cierto código HTML; el servidor toma este código recién creado y lo envía al cliente por medio del protocolo HTTP.

Las aplicaciones de servidor suelen ser la mejor opción para realizar aplicaciones web. La razón es que, al ejecutarse ésta en el servidor y no en la máquina del cliente, éste no necesita ninguna capacidad añadida, como sí ocurre en el caso de querer ejecutar aplicaciones en el lado del cliente. Así pues, cualquier cliente dotado de un navegador web básico puede utilizar este tipo de aplicaciones.

El 75% de las aplicaciones del lado del servidor están escritas en **PHP**, seguido de **ASP** y las demás opciones usadas de forma alternativa y muy casual.

❖ Programación orientada a objetos (POO)

La programación orientada a objetos (**POO**) es un paradigma de programación que viene a innovar la forma de obtener resultados. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial.

Los objetos son entidades que tienen un determinado "estado", "comportamiento (método)" e "identidad (concepto análogo a variable o constante)"

La **programación orientada a objetos** difiere de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida.

Los programadores que emplean POO, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

La **POO** es una forma de programar que introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:

- ***Clase***

Una clase se puede definir de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ella.

- ***Herencia***

Por ejemplo, herencia de la clase C a la clase D, es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D.

Por lo tanto, puede usar los mismos métodos y variables registrados como "públicos" (*public*) en C. Los componentes registrados como "privados" (*private*) también se heredan pero se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos.

- ***Objeto***

Instancia de una clase. Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos.

- ***Método***

Algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

❖ Patrón de arquitectura MVC

Modelo-Vista-Controlador (**MVC**) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

El objetivo principal de este patrón o cualquier otro similar es de hacer los proyectos más escalables, permitiendo la reutilización de código y una mejor estructuración. Todo ello para facilitar la futura inclusión de nuevas funcionalidades así como para modificar las ya existentes reduciendo la deuda tecnológica que podamos adquirir con malas prácticas.

De manera genérica, los componentes de este patrón se podrían definir como sigue:

- ***Modelo***

Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación lógica de negocio.

Envía a la '**vista**' aquella parte de la información que en cada momento se le solicita para que sea mostrada. Las peticiones de acceso o manipulación de información llegan al '**modelo**' a través del '**controlador**'

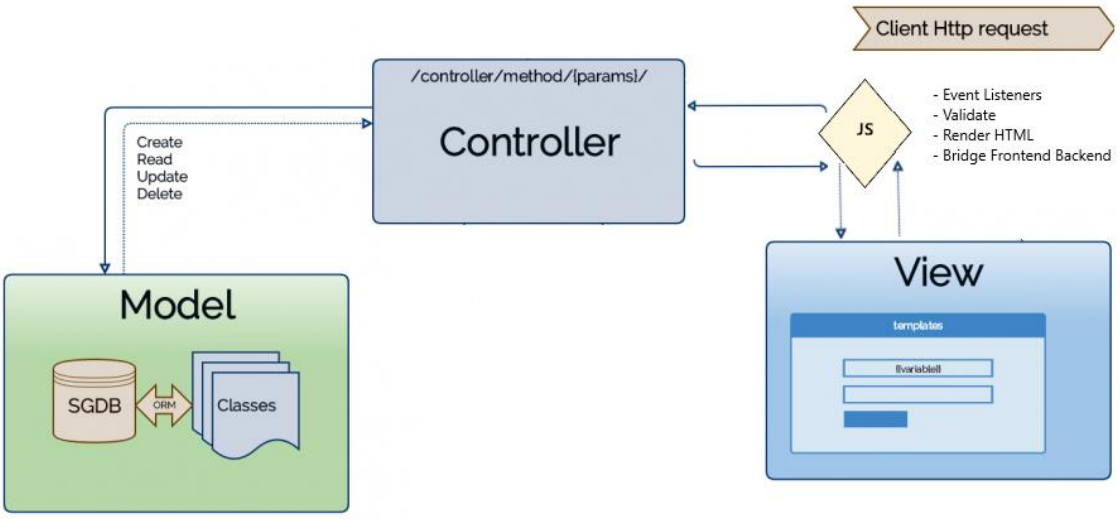
- ***Controlador***

Responde a eventos (usualmente acciones del usuario) e invoca peticiones al '**modelo**' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos).

También puede enviar comandos a su '**vista**' asociada si se solicita un cambio en la forma en que se presenta el '**modelo**' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el '**controlador**' hace de intermediario entre la '**vista'** y el '**modelo**'.

- ***Vista***

Presenta el '**modelo**' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho '**modelo**' la información que debe representar como salida.



Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo de control que se sigue generalmente es el siguiente:

- ✓ El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)

- ✓ El **controlador** recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (**handler**). En nuestro caso serían los ficheros JavaScript que actúan de enlace entre el Frontend y el Backend .

- ✓ El **controlador** accede al **modelo**, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario).

- ✓ El **controlador** delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se reflejan los cambios en el modelo (por ejemplo, producir un listado del contenido del carro de la compra).

  Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

- ✓ La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente...

Guía Formación – Nivel 0

Para esta primera fase de la formación haremos uso de la plataforma :

https://www.w3schools.com/

Es una plataforma de enseñanza gratuita enfocada en la mayoría de su contenido a programación web. En un principio nos centraremos en las siguiente tecnologías:

[FRONTEND]

- ✓ HTML5
- ✓ CSS3
- ✓ BOOTSTRAP 4

[BACKEND - BD]

- ✓ PHP
- ✓ SQL
- ✓ JAVASCRIPT

[LIBRERÍAS]

- ✓ JQUERY

En esta primera fase es muy importante afianzar el contenido que indicaremos porque suponen la base de la programación web. Y si bien es cierto que en el mercado hay frameworks y tecnologías que nos ayudan mucho en el desarrollo de software es imprescindible tener estos conocimientos de base.

De cada tecnología indicaré a continuación los apartados que el alumno irá estudiando. En la división de arriba he decidido incluir Javascript como tecnología de BACKEND aunque se podría considerar como de FRONTEND , es un lenguaje muy versátil y la principal fuente que tenemos para conectar con la parte del cliente.

Aunque este lenguaje tiene también su uso en la parte de servidor gracias a tecnologías como NODE.

En cada apartado de cada tecnología aparte de la teoría encontraremos tipos de prácticas como las vistas en las siguientes capturas:

Este botón Try it Yourself >> nos abrirá una ventana con algún ejercicio que son muy útiles a la hora de ver código y practicar los distintos lenguajes.

Al final de cada sección y dependiendo de la tecnología hay también otros ejercicios en los que poner en práctica toda la teoría incluida en ese apartado.

Es muy recomendable aparte de la lectura de cada apartado que estos primeros ejercicios aunque no queden registrados en ningún sitio se hagan todos porque son un forma de iniciarse en el código muy útil.

Lo importante de todo esto es preparar bien las bases para pasar después a prácticas más complejas en entornos similares a un proyecto real lo que llevará después en un futuro a poder realizar tareas de proyectos tanto internos como de clientes externos.

Dentro de cada tecnología de los apartados que tienen se recomienda mínimo los siguientes, por supuesto queda a disposición del alumno ampliar con otras referencias de esta misma plataforma o búsqueda de información en otros medios.

De hecho esto es una parte muy importante a la hora de programar que es la búsqueda de recursos y saberse manejar entre documentación externa.

Este sería el orden a seguir en la formación.

[HTML5] https://www.w3schools.com/html/default.asp

- HTML5 TUTORIAL
- HTML Forms
- HTML Examples
- HTML References

[CSS3] https://www.w3schools.com/css/default.asp

- CSS TUTORIAL
- CSS Advanced
- CSS Examples

[BOOSTRAP 4] https://www.w3schools.com/bootstrap4/default.asp

- Boostrap 4 Tutorial
- Boostrap 4 Grid

[PHP] https://www.w3schools.com/php/default.asp

- PHP Tutorial
- PHP Forms
- PHP Advanced
- MySQL Database
- PHP Examples
- PHP Reference

[SQL] https://www.w3schools.com/sql/default.asp

- SQL Tutorial
- SQL Database
- SQL References
- SQL Examples

[JAVASCRIPT] https://www.w3schools.com/js/default.asp

- JS Tutorial
- JS Objects
- JS Functions
- JS HTML DOM
- JS vs JQuery
- JS Examples

[JQUERY] https://www.w3schools.com/jquery/default.asp

- JQuery Tutorial
- JQuery HTML
- JQuery Examples