UNIVERSIDAD SERGIO ARBOLEDA

TODOS A LA U
Fórmate digital

# Aplicación Web: Javascript y llamados Ajax

Desarrollo de Software

# Contenido

- Introducción
- Ventajas de Aplicación Web
- Aplicación MVC a Modelo Cliente-Servidor
- Conceptos: FrontEnd y BackEnd
- FrontEnd en una aplicación Web

# Presentación de Aplicación Web

- La difusión de equipos de cómputo, junto con la expansión de la tecnología móvil, así como la expansión de internet, ha cambiado el paradigma de entrega de soluciones informáticas.

- Antes el software era entregado al cliente a través de medios de almacenamiento de información.

- La aparición de las páginas web dio una pista a un nuevo modelo de aplicación.

# Las Aplicaciones Web

- Mediante la interpretación de código HTML y la ejecución de código JavaScript, los navegadores web se han convertido en un medio para distribuir una aplicación..

- La aplicación web tiene como base la generación dinámica de su contenido, lo que hace que sea fundamentalmente diferente de las páginas web, pues esas tienen su contenido estático y no cambia salvo por pequeñas interacciones del usuario. La página está concebida para mostrar contenido.

- La ejecución a través de un navegador web, puede implicar la conexión a internet, aunque no siempre. Es posible mediante la conexión a internet establecer comunicación con servidores de diferentes lugares y diferentes funcionalidades, protocolos y arquitecturas para robustecer la aplicación sin que esto genere complicaciones al usuario. Así por ejemplo un cliente web de correo electrónico como GMAIL generará en el usuario la experiencia de usuario de estar utilizando una aplicación web siendo transparente para él los protocolos y configuraciones relacionados con el correo electrónico.

# Ventajas de la aplicación WEB

- Los procesos de cambio y actualización, tienen una entrega mucho más rápida, debido a que el cliente no tiene que descargar ninguna actualización.

- Se tiene mayor control y monitoreo del uso que el cliente le da a la aplicación.

- El software puede ser accedido de manera rápida sin necesidad de procesos de instalación.
- Al ser ejecutado en el navegador web, el software suele no depender del tipo de plataforma que utilice el cliente, teniendo una cobertura más amplía.
- El usuario, salvo por pequeños archivos de configuración, no se verá forzado a ocupar su almacenamiento con programas instalados.

# MVC a Cliente Servidor

## Recordemos MVC

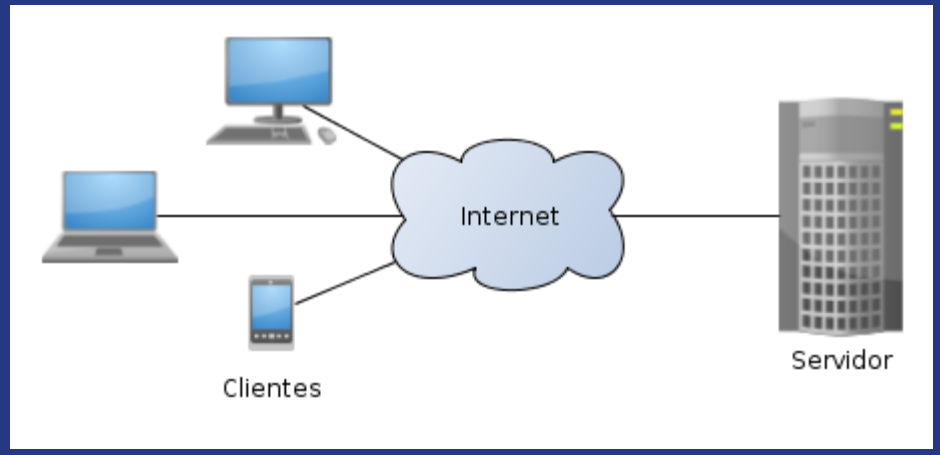
- La lógica del negocio y la manera como esta interactúa con los datos, se ubica en la capa de **Modelo**.
- En una aplicación StandAlone, el **Contolador** se encarga de gestionar la interacción del usuario sobre la vista (o de otras aplicaciones)
- La **Vista** suele ser la interfaz con la que interactúa el usuario (a veces, otras aplicaciones) y mediante la cual el computador se comunica con el usuario.

- El controlador toma la información que deja el usuario y la entrega al modelo para que este ejecute la lógica del negocio con estos datos como entrada. Hay ocasiones en las que el controlador responde a la vista sin necesidad de avisar al modelo, suele suceder por ejemplo en las validaciones.
- El modelo al recibir la instrucción del controlador, entrega los resultados a la vista, es decir que el modelo actualiza la vista. En muchas implementaciones esto puede no ser posible, por lo que el resultado se entrega de vuelta al controlador en forma de respuesta.

# MVC a Cliente Servidor

## Ahora veamos Cliente-Servidor

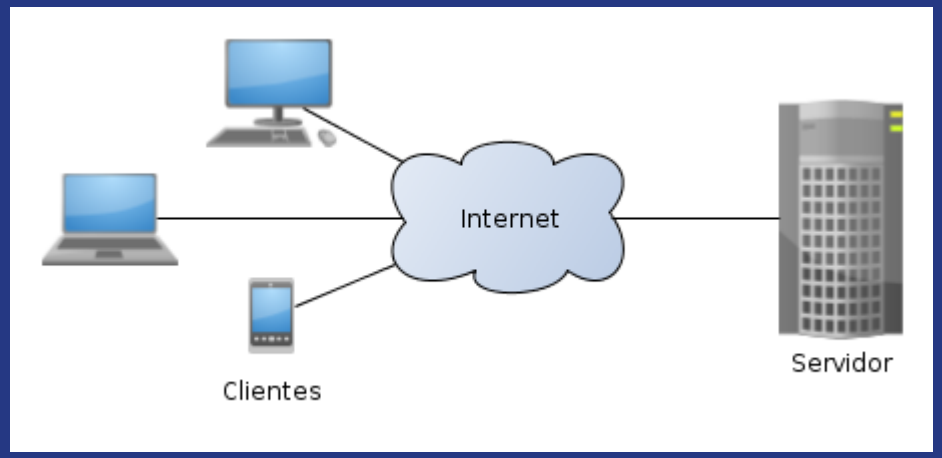
- La arquitectura cliente servidor está establecida como la manera en la que dos máquinas, conectadas entre sí a través de una red (que puede ser internet) realizan un trabajo gracias a una mutua colaboración.
- El servidor suele encargarse de la lógica de negocio, así mismo como el almacenamiento y persistencia de datos. No siempre el servidor es una sola máquina, es posible (y muchas veces necesario) tener una máquina dedicada un servicio específico, así los distintos servidores tendrán que comunicarse entre sí para lograr una tarea. Esta comunicación también se logra a través de una red.



# MVC a Cliente Servidor

## Ahora veamos Cliente-Servidor

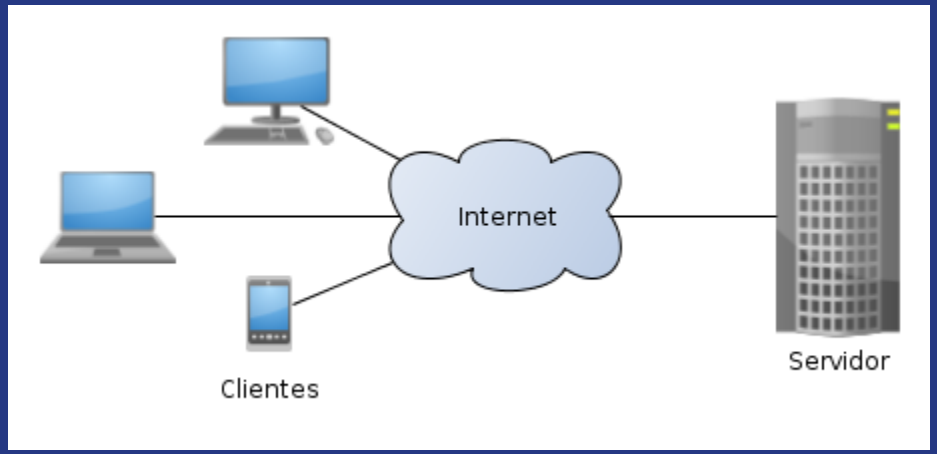
- El cliente funciona como un terminal encargado de hacer peticiones al servidor, cuando este le contesta, su contenido o visualización se actualiza.
- El servidor es capaz de responder a varios clientes.
- Esta arquitectura se diferencia del antiguo modelo de MainFrame en que el cliente no es un terminal bruto, sino que es una máquina con capacidad de procesamiento propio.



# FrontEnd y Backend

## Códigos y lenguajes

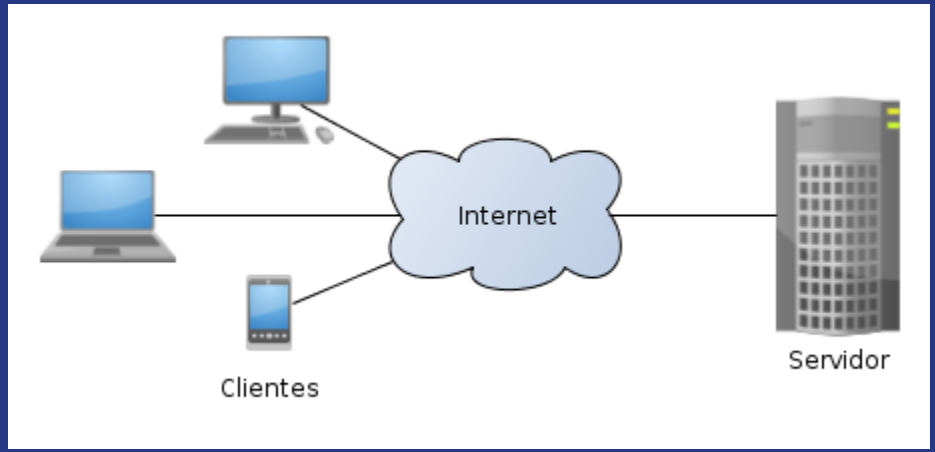
- En una aplicación web, el desarrollador debe escribir el código de las dos partes, tanto de cliente como de servidor.
- El cliente, será conocido como FrontEnd
- El servidor será conocido como BackEnd
- Teniendo en cuenta que el FrontEnd será utilizado a través de un navegador Web, el lenguaje utilizado será HTML para la diagramación, CSS para dar estilos y ajustar el *look and feel* y javascript como lenguaje de programación, el cual es ejecutado dentro del navegador.



# FrontEnd y Backend

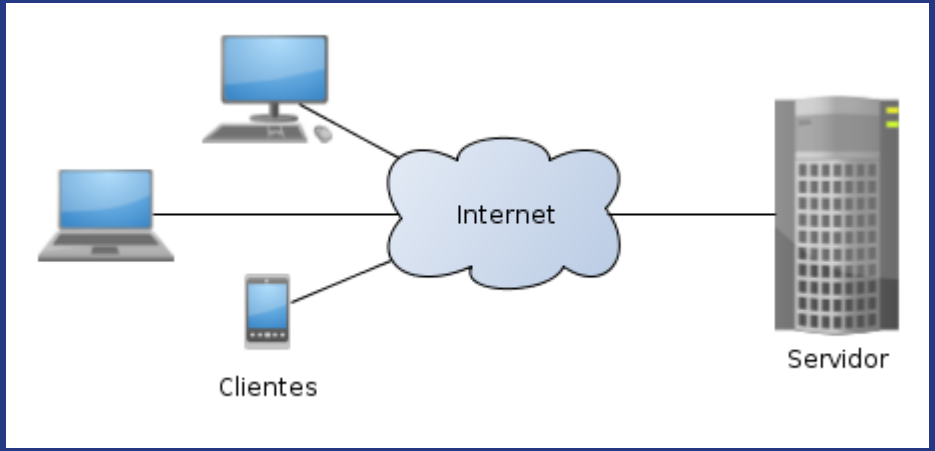
## Códigos y lenguajes

- Por su parte el backend, es un software escrito en una gran variedad de lenguajes, tales como JAVA, PHP, C#, PERL, RUBY, PYTHON y muchos más, incluyendo también Javascript.
- El backend, al ser una aplicación, necesita un Servidor de aplicaciones para que esta sea “expuesta” y pueda ser accedida por el código del frontend.



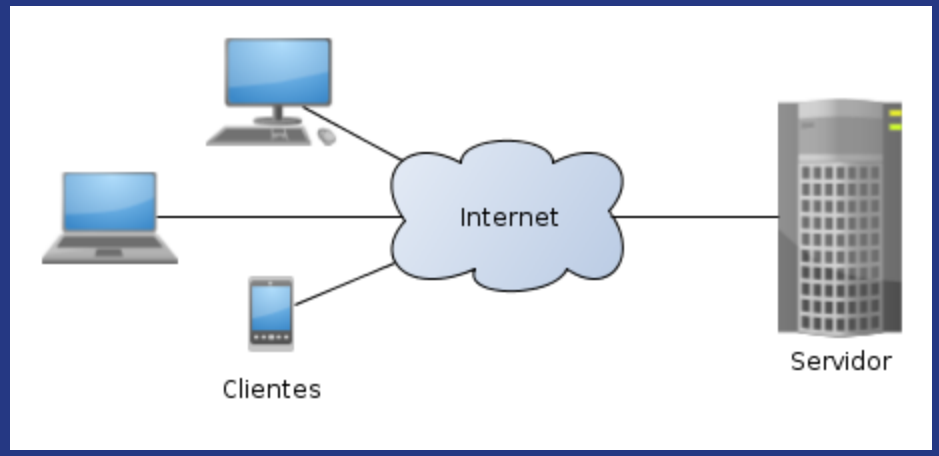
# Comunicación entre FrontEnd y BackEnd

- De forma común, en una arquitectura cliente-servidor, la comunicación suele ser iniciada siempre por el Cliente. Esta comunicación se hace a través de peticiones sobre el protocolo HTTP.
- El tipo de Petición indica la intención o la necesidad del cliente. Si bien se puede utilizar cualquier petición para cualquier fin, es **preferible** que se utilicen peticiones get para solicitar información, peticiones post para guardar, peticiones put para editar (también existen las peticiones patch) y las peticiones delete, para borrar.



# Tipos de Peticiones

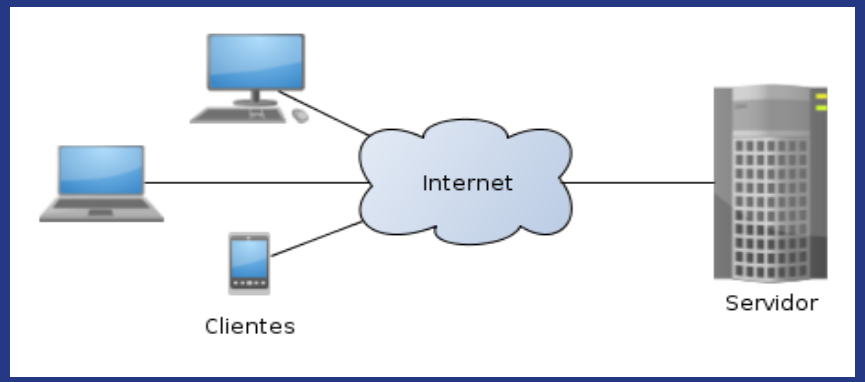
- Las primeras comunicaciones web entre cliente y servidor utilizaban peticiones **síncronas.**
- Una petición síncrona es aquella de que al ejecutarse no inicia una etapa sin haber terminado la anterior.
- Cuando la petición se hace a un servidor, puede tomarse un tiempo en dar respuesta. Hasta que no se tenga una respuesta, la ejecución no continúa.
- La experiencia de usuario se ve afectada por “congelamientos” mientras se tiene una respuesta.



# Peticiones AJAX

En 2005 Jesse James Garret acuña el término ***AJAX*** para identificar la tecnología de peticiones **asíncronas.**

- Al ejecutar una petición asíncrona, el flujo de instrucciones no se detiene a esperar la respuesta del servidor, razón por la cual no se congela la ejecución ante los ojos del usuario.
- Con el fin de mejorar la experiencia de usuario, se establecen las acciones por realizar antes de iniciar la petición y qué hacer según la respuesta obtenida: sea exitosa o no.



# Ejemplo petición AJAX

## Ejemplo con JQuery

```
$.ajax({
    url : 'api/servicio',
    data : { id : 123 },
    type : 'GET',
    dataType : 'json',

    success : function(json) {
        $('<h1/>').text(json.title).appendTo('body');
        $('<div class="content"/>')
            .html(json.html).appendTo('body');
    },
    error : function(xhr, status) {
        alert('ha sucedido un problema');
    },
    complete : function(xhr, status) {
        alert('Petición realizada');
    }
});
```