**Plataformas de programación Web en Entorno Servidor**

# **1. Arquitectura Web**

El término Arquitectura Web se refiere a la estructura y organización de una aplicación web.

Determina cómo se diseñan y distribuyen los componentes y recursos de un sitio web para lograr objetivos específicos, como:

* la usabilidad,
* la eficiencia,
* la escalabilidad
* y la seguridad.

La Arquitectura Web va a incluir aspectos técnicos, como la infraestructura de servidores, base de datos, lenguajes de programación utilizados, y consideraciones de diseño de interfaz de usuario y experiencia del usuario.

## **1.1. Modelo Cliente-Servidor**

La arquitectura cliente-servidor tiene dos partes diferenciadas:

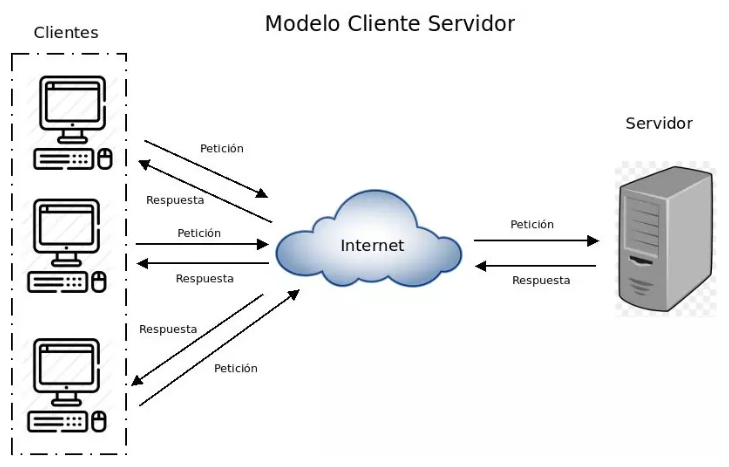
* la parte del **servidor**
* la parte del **cliente** o grupo de clientes.

En esta arquitectura el **cliente** suele ser una estación de trabajo o dispositivo móvil que solicita varios servicios al servidor. En el caso de las Aplicaciones Web, el cliente será el navegador web.

El **servidor** es una máquina que actúa como depósito de datos (mediante un sistema gestor de base de datos), y que se encarga de dar la respuesta demandada por el cliente.

La principal ventaja de esta arquitectura es: Usabilidad

|  |
| --- |
| **conectar a varios clientes a los servicios que provee un servidor** |



Servidor Apache => PHP / Servidor Tomcat => JAVA

SERVIDORES: WEB, SMTP, FTP

Otras ventajas que aporta también son:

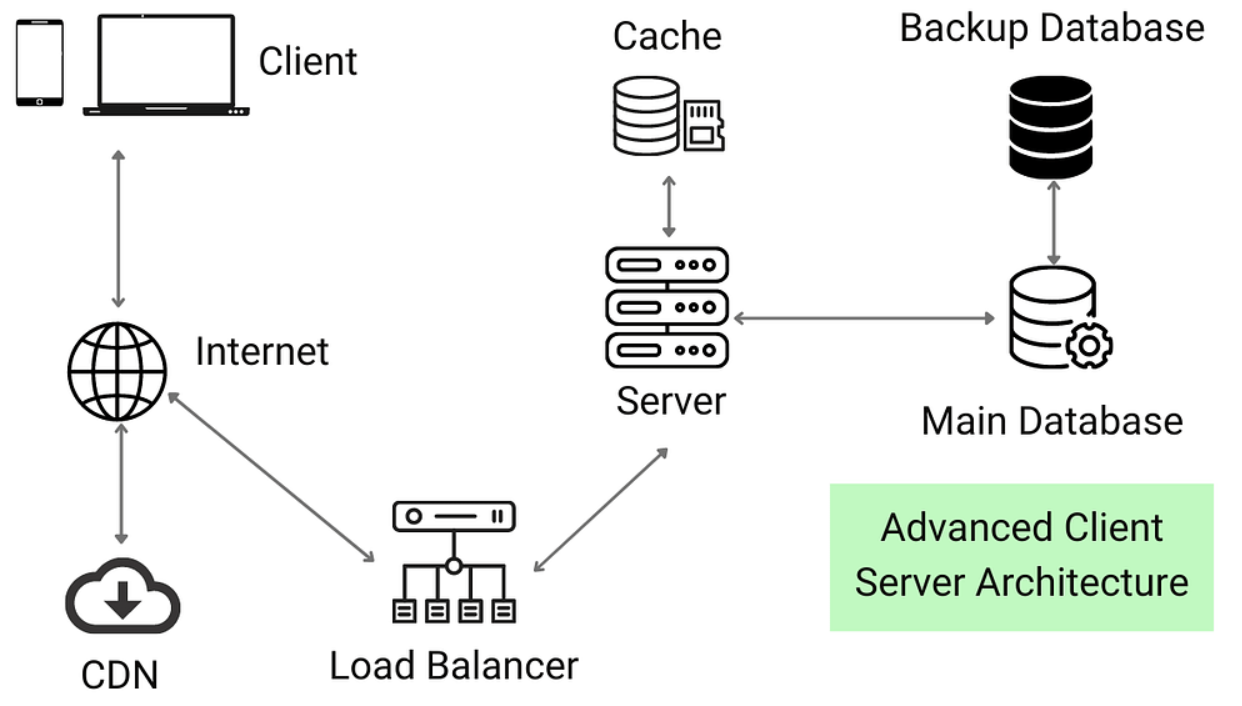
* **Escalabilidad**: El modelo cliente-servidor es altamente escalable, lo que significa que puedes agregar más clientes o servidores según sea necesario para adaptarse a la demanda. Esto facilita la expansión de sistemas y aplicaciones a medida que crecen.
* **Facilidad de mantenimiento**: En un modelo cliente-servidor, los componentes cliente y servidor son independientes. Esto facilita la actualización o el mantenimiento de uno de los componentes sin afectar al otro. Por ejemplo, puedes actualizar el servidor sin necesidad de modificar los clientes.
* **Seguridad**: Puedes implementar medidas de seguridad en el servidor para proteger los datos y los recursos de manera centralizada. Esto facilita la gestión de la seguridad y garantiza un control más estricto sobre quién tiene acceso a qué.
* **Rendimiento**: El modelo cliente-servidor puede distribuir la carga de trabajo entre múltiples servidores, lo que puede mejorar el rendimiento y la capacidad de respuesta del sistema. Esto es especialmente útil para aplicaciones que manejan una gran cantidad de solicitudes de clientes.
* **Acceso remoto**: Los clientes pueden acceder a recursos o servicios en el servidor desde ubicaciones remotas a través de redes, lo que facilita la colaboración y el acceso a datos y aplicaciones desde cualquier lugar con conexión a Internet.
* **Centralización de datos**: Los datos pueden mantenerse centralizados en el servidor, lo que facilita la gestión y la copia de seguridad de los datos. Esto reduce el riesgo de pérdida de datos en comparación con modelos descentralizados.
* **Compatibilidad multiplataforma**: Los clientes y servidores pueden estar en diferentes plataformas y sistemas operativos. Esto permite que aplicaciones cliente-servidor sean compatibles con una variedad de dispositivos y sistemas.
* **Balanceo de carga**: Puedes implementar equilibradores de carga para distribuir las solicitudes de los clientes de manera uniforme entre múltiples servidores, lo que mejora la eficiencia y la disponibilidad de la aplicación.
* **Economía de recursos**: Los clientes ligeros pueden ejecutarse en dispositivos con recursos limitados, ya que gran parte del procesamiento y almacenamiento pesado se realiza en el servidor. Esto puede reducir los requisitos de hardware en los clientes.
* **Administración centralizada**: La administración de usuarios, permisos y configuraciones se puede realizar de manera centralizada en el servidor, lo que simplifica la gestión de sistemas y aplicaciones.

**Centralizado vs Distribuido**

**P2P = distribuido**

**Clie-Serv = Centralizado**

A continuación, podemos ver un modelo avanzado de cliente-servidor multicapa:



Content Delivery Network = Servicios en la nube de clones de servidores para dar servicios mas rápidos por cercanía, Red de Servicios de Contenedores (Data Center)

## **1.1 Página Web estática WEB 1.0**

Se entiende por aquella página en la que el código fuente (HTML+CSS) que se envía al cliente (navegador web) se encuentra fijo, no se mueve ni cambia de ninguna manera (sin interactividad).

Las páginas web estáticas, son mayoritariamente informativas.

## **1.2 Página Web dinámica WEB 2.0**

Una página web dinámica incluye aspectos que se caracterizan por la interactividad y la funcionalidad. Los usuarios pueden interactuar con la información que se presenta en la página gracias a las instrucciones creadas a través de los lenguajes de programación y la base de datos sobre la que está construida.

Para generar una página dinámica, donde el contenido cambia, tenemos dos alternativas: <https://www.google.com/search?sca_esv=568036337&rlz=1C5CHFA_enES986ES986&sxsrf=AM9HkKmW0G_FbFyuTZLYH1OacnGuPnwIgQ:1695584290672&q=como+funciona+el+renderizado+en+el+lado+del+servidor&tbm=vid&source=lnms&sa=X&ved=2ahUKEwiq89Pa_8OBAxU4TKQEHUvlDHkQ0pQJegQICRAB&biw=1327&bih=811&dpr=1#fpstate=ive&vld=cid:08e15feb,vid:_oJUUtv8itU,st:0>

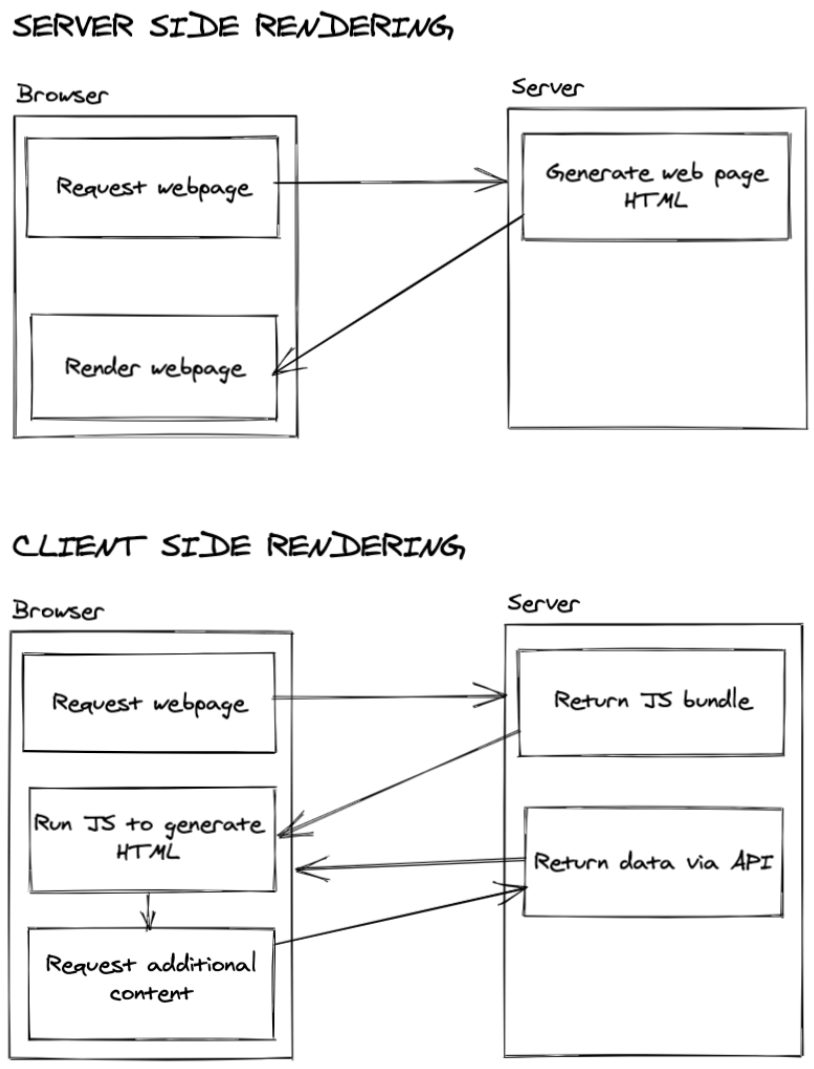
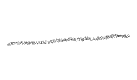
* Renderizado o Actualización y creación de la página (código HTML+CSS+JS) **en el lado del servidor:**

Mediante un lenguaje de programación o scripting en el lado del servidor se genera todo el contenido (HTML+CSS) accediendo a una BD u otros servicios. Al cliente se le envía la página web (el código fuente HMTL+CSS) completo. Lo tendremos en tecnologías de servidor de páginas clásicas PHP, JSP o ASP.

* Renderizado o Actualización y creación de la página (código HMTL+CSS+JS) **en el lado del cliente:**

Una página mediante el acceso a servicios de tipo API REST (backend), que se invocan desde cliente JS en el navegador, obtendrá la información necesaria para implementar cambios en el DOM (estructura de objetos de la página html manipulable desde JS) que actualicen la página. Lo tendremos en librerías AJAX JQuery y frameworks de cliente (frontend) como Angular, React o Vue.

Renderizado del lado Servidor vs Renderizado del lado Cliente

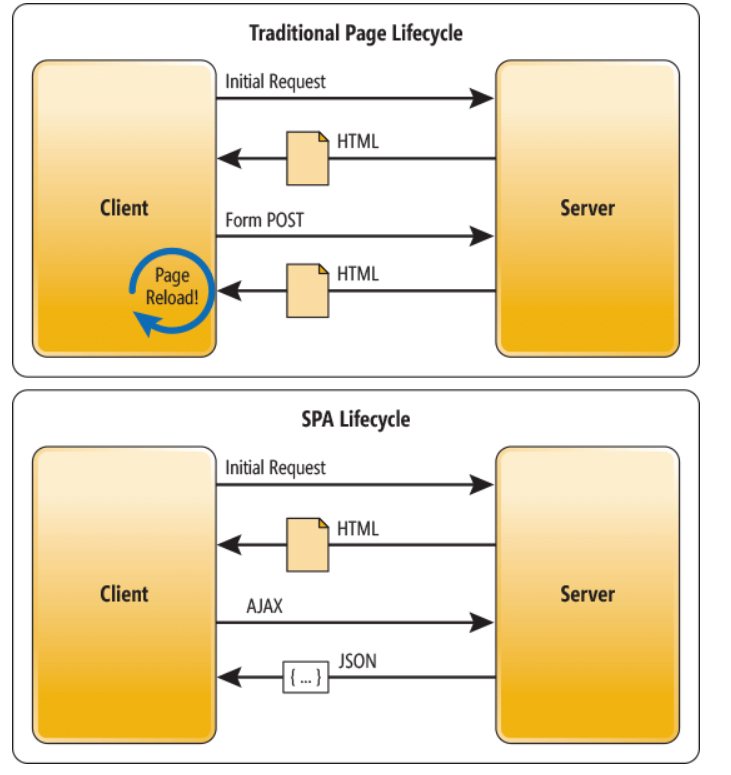




### **1.2.1 Aplicación de Página Única (Single Page Application). blog**

Es un tipo de aplicación web o sitio web que se caracteriza por cargar y mostrar todo su contenido en una sola página web, en lugar de cargar páginas web individuales cada vez que el usuario interactúa con la aplicación. Se basan en librerías y frameworks de JS para conseguir actualizar el contenido de la página a medida que el usuario interactúa con ella.

Algunos ejemplos populares de frameworks de JavaScript utilizados para desarrollar SPAs incluyen Angular, React y Vue.js. Las SPAs son adecuadas para aplicaciones web interactivas y ricas en contenido, como aplicaciones de una sola página, paneles de administración y aplicaciones de redes sociales, donde la carga rápida y la interacción fluida son esenciales para una experiencia de usuario óptima.



## 1.3 Arquitectura de 3 niveles (tiers)

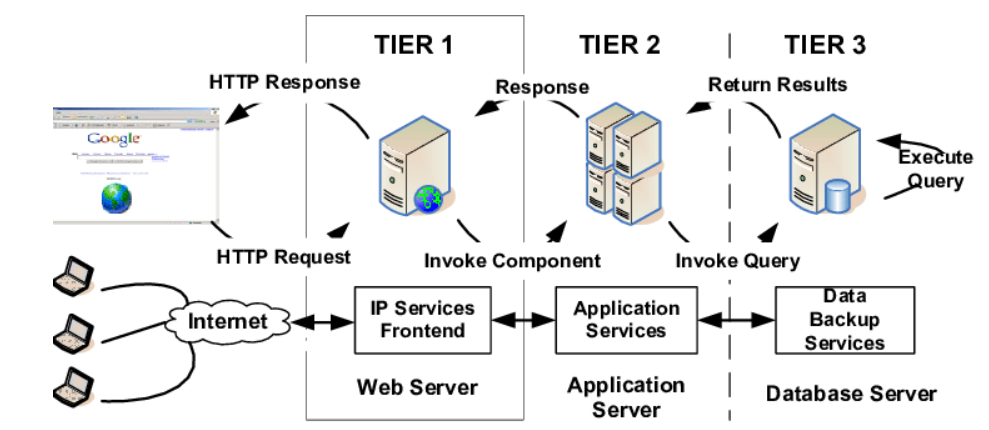
Hay que distinguir entre **capas físicas** (tier o niveles) y **capas lógicas** (layer o capas).

### 1.3.1 **Tier**

Capa física de una arquitectura. Supone un nuevo elemento hardware separado físicamente. Las capas físicas más **alejadas del cliente** están más protegidas, tanto por firewalls como por VPN.

Ejemplo de arquitectura en tres capas físicas (3 tier): van por servicios

1. Servidor Web (HTML) **Presentación**
2. Servidor de Aplicaciones (Cloud, DNS, FTP, Mail) **Procesos**
3. Servidor de base de datos (DDBB) **Datos**

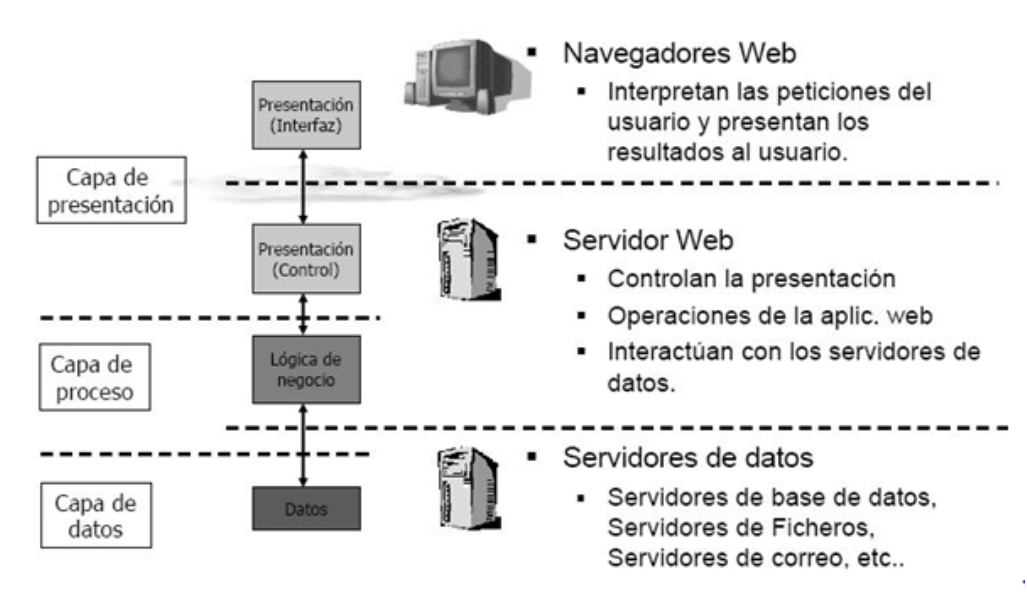


### 1.3.2 **Layer**

En cambio, las capas lógicas (**layers**) organizan el código respecto a su funcionalidad:

* 1. Presentación / Visualización
  2. Negocio / Aplicación / Proceso
  3. Datos / Persistencia

Como se observa, cada una de las capas se puede implementar con diferentes lenguajes de programación y/o herramientas. Nuestro proyecto estará dividido en estas tres capas lógicas, que llevará a un modelo MVC (Modelo Vista Controlador)



## Actividades de Desarrollo

1. Realiza un informe sobre las características de la arquitectura Peer-to-Peer (P2P) comparándolas con la arquitectura Cliente-Servidor. Indica aplicaciones que utilicen esta arquitectura.

1. Realiza un informe sobre las Web3 de blockchain y qué relaciones tiene con las arquitecturas Cliente-Servidor y Peer-to-Peer.

#### Informe sobre las Web3 de blockchain y su relación con las arquitecturas Cliente-Servidor y Peer-to-Peer

## Introducción

En los últimos años, la tecnología blockchain ha ganado popularidad debido a su capacidad para proporcionar un sistema descentralizado, seguro y transparente. Junto con el desarrollo de blockchain, ha surgido el concepto de Web3, que se refiere a la evolución de la web hacia una arquitectura más descentralizada y orientada a la confianza.

## ¿Qué es Web3?

Web3 es un término utilizado para describir la próxima generación de aplicaciones y servicios en línea que utilizan tecnologías basadas en blockchain y contratos inteligentes. A diferencia de la web tradicional (Web2), en la que los datos y la lógica están centralizados en servidores controlados por empresas o instituciones, Web3 busca descentralizar estos aspectos y empoderar a los usuarios.

## Características de Web3

Las aplicaciones y servicios basados en Web3 tienen algunas características distintivas:

1. \*\*Descentralización\*\*: En lugar de depender de servidores centralizados, Web3 utiliza redes peer-to-peer (P2P) basadas en blockchain para almacenar datos y ejecutar contratos inteligentes. Esto elimina la necesidad de intermediarios y permite una mayor transparencia y seguridad.

2. \*\*Confianza\*\*: Gracias a la tecnología blockchain, las aplicaciones Web3 pueden garantizar la integridad de los datos y la inmutabilidad de las transacciones. Esto reduce la dependencia de terceros de confianza y fomenta la transparencia y la verificabilidad.

3. \*\*Propiedad de los datos\*\*: En Web3, los usuarios tienen un mayor control sobre sus datos personales. Pueden decidir qué información compartir y con quién, y tienen la capacidad de revocar el acceso en cualquier momento.

1. \*\*Interoperabilidad\*\*: Web3 permite la interoperabilidad entre diferentes aplicaciones y servicios basados en blockchain. Los usuarios pueden combinar y utilizar diferentes componentes de manera flexible para crear nuevas soluciones.

## Relación con las arquitecturas Cliente-Servidor y Peer-to-Peer

Tanto las arquitecturas Cliente-Servidor como Peer-to-Peer tienen una relación con Web3, pero de diferentes maneras:

1. \*\*Arquitectura Cliente-Servidor\*\*: Esta arquitectura es la más común en la web tradicional (Web2). En un sistema Cliente-Servidor, los clientes (navegadores web) solicitan recursos a un servidor centralizado, que responde proporcionando los datos solicitados. En este modelo, el servidor es responsable del almacenamiento y procesamiento de los datos. En Web3, aunque todavía existen elementos de arquitectura Cliente-Servidor, la naturaleza descentralizada de blockchain reduce la dependencia de un único servidor centralizado. La lógica y los datos se distribuyen en la red blockchain, lo que permite una mayor resistencia a la censura y a los fallos.

2. \*\*Arquitectura Peer-to-Peer\*\*: En una arquitectura Peer-to-Peer, los nodos de la red interactúan directamente entre sí, sin la necesidad de un servidor centralizado. Cada nodo puede actuar tanto como cliente como servidor, compartiendo y consumiendo recursos de otros nodos. En Web3, las redes blockchain son inherentemente Peer-to-Peer, ya que todos los nodos participantes tienen una copia completa de la cadena de bloques y pueden interactuar directamente entre sí para validar transacciones y ejecutar contratos inteligentes.

En resumen, Web3 utiliza elementos de ambas arquitecturas Cliente-Servidor y Peer-to-Peer. Si bien todavía hay una dependencia en cierta medida de la arquitectura Cliente-Servidor para interactuar con la web tradicional, la tecnología blockchain subyacente permite una mayor descentralización y confianza a través de una arquitectura Peer-to-Peer. Esto proporciona a los usuarios un mayor control sobre sus datos y fomenta la transparencia y la seguridad en las aplicaciones y servicios en línea.

### El tamaño de una cadena de bloques puede variar dependiendo de varios factores, como el número de transacciones registradas, el tamaño de los datos almacenados en cada bloque y el algoritmo de consenso utilizado.

En general, una cadena de bloques comienza con un bloque génesis, que suele ser de tamaño fijo y relativamente pequeño. A medida que se agregan nuevos bloques a la cadena, el tamaño total aumenta.

La cantidad de espacio ocupado por una cadena de bloques también puede depender del tipo de blockchain. Por ejemplo, las cadenas de bloques de Bitcoin y Ethereum son públicas y tienen muchos participantes, lo que resulta en un tamaño más grande debido al gran número de transacciones y contratos inteligentes registrados.

Además, es importante tener en cuenta que las cadenas de bloques pueden crecer continuamente a medida que se agregan nuevos bloques. Sin embargo, algunos mecanismos como la compresión de datos o la eliminación de transacciones antiguas pueden ayudar a reducir el tamaño de la cadena de bloques.

En resumen, el tamaño de una cadena de bloques puede variar significativamente y depende de varios factores, incluyendo el número de transacciones, el tamaño de los bloques y el algoritmo de consenso utilizado.

### La tecnología de red peer-to-peer (P2P) se utiliza en una variedad de aplicaciones. Aquí hay algunos ejemplos:

1. Compartir archivos: Aplicaciones como BitTorrent utilizan redes P2P para permitir a los usuarios compartir archivos de manera descentralizada, sin depender de un servidor central.

2. Mensajería y comunicación: Aplicaciones de mensajería instantánea como Skype y WhatsApp utilizan la tecnología P2P para facilitar la comunicación entre usuarios, permitiendo que los mensajes se envíen directamente entre ellos sin pasar por un servidor centralizado.

3. Streaming de contenido multimedia: Plataformas de streaming de video como Popcorn Time o Ace Stream utilizan redes P2P para distribuir el contenido multimedia entre los usuarios, lo que permite una mejor escalabilidad y reducción de costos de ancho de banda.

4. VoIP (Voz sobre IP): Aplicaciones como Discord o TeamSpeak utilizan tecnología P2P para permitir llamadas de voz y conferencias en línea, evitando la necesidad de servidores centrales para gestionar las conexiones.

5. Juegos en línea: Algunos juegos en línea utilizan tecnología P2P para permitir partidas multijugador, donde los jugadores se conectan directamente entre sí en lugar de pasar por servidores centrales.

6. Sistemas de almacenamiento distribuido: Plataformas como IPFS (InterPlanetary File System) utilizan redes P2P para crear sistemas de almacenamiento distribuido, donde los archivos se almacenan y replican en varios nodos de la red, lo que mejora la disponibilidad y resistencia a fallos.

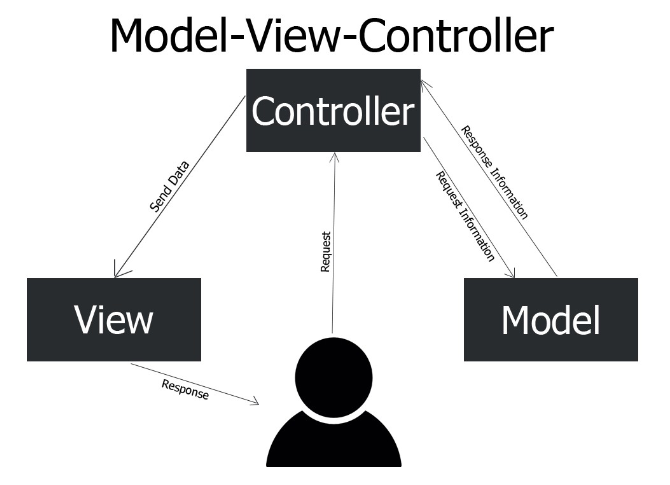
Estos son solo algunos ejemplos de cómo se utiliza la tecnología de red P2P en diversas aplicaciones. La naturaleza descentralizada y la capacidad de compartir recursos entre los usuarios hacen que esta tecnología sea muy versátil y ampliamente utilizada en diferentes campos.

# 1.4 MVC

*Model-View-Controller* o Modelo-Vista-Controlador es un modelo de arquitectura que separa **los datos y la lógica de negocio** respecto a **la interfaz de usuario y el componente encargado de gestionar los eventos y las comunicaciones.**

Al separar los componentes en elementos conceptuales permite reutilizar el código y mejorar su organización y mantenimiento. Sus elementos son:

* **Modelo**: representa la información y gestiona todos los accesos a ésta, tanto consultas como actualizaciones provenientes, normalmente, de una base de datos. Se accede vía el controlador. Con pojos que mapean los datos de ddbb al mundo ojetos.
* **Controlador**: Responde a las acciones del usuario, y realiza peticiones al modelo para solicitar información. Tras recibir la respuesta del modelo, le envía los datos a la vista. LOGICA CON END POINTS PROCESA CON LOS DATOS DE MODELO
* **Vista**: Presenta al usuario de forma visual el modelo y los datos preparados por el controlador. El usuario interactúa con la vista y realiza nuevas peticiones al controlador.



# 1.5 Servidor Web

Software que recibe peticiones HTTP (GET, POST, DELETE, ...). Devuelve el recurso solicitado (HTML, CSS, JS, JSON, imágenes, etc...)

Uno de los servidores más utilizados es Apache Web Server (https://httpd.apache.org/), creado en 1995.

Software libre y multiplataforma

Sistema de módulos dinámicos → PHP, Python, Perl

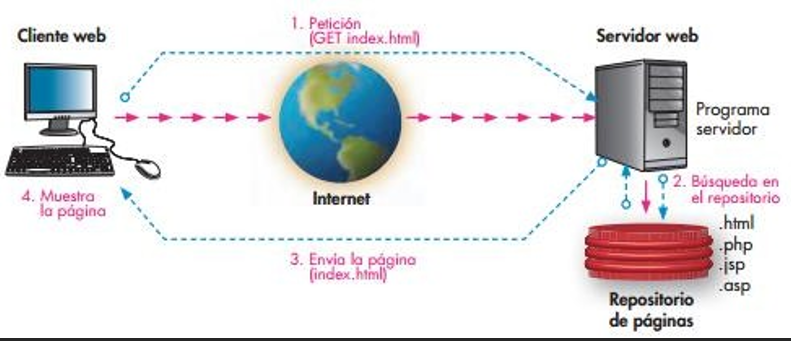
Una alternativa es Nginx (https://www.nginx.com). Se trata de un producto más moderno (2004) y que en determinados escenarios tiene mejor rendimiento que Apache.

Comparativa servidores web: <https://w3techs.com/technologies/history_overview/web_server/ms/q>

## Actividad de Desarrollo

1. Realiza una presentación comparando las características de los servidores web Apache y Nginx.

# Apéndice: notas sobre http



URL + Filtro ? = URI

[protocolo]://[host][:puerto]/[path]/[fichero]?[filtro]#[identificador]

Ej.

http://www.juntadeandalucia.es/averroes/centros- tic/11700470/moodle2/course/view.php?id=111#section-1

URL = localizador hasta fichero.

URI = localizador completo incluyendo filtro e identificador.

## Codificación de URL

Las URL sólo se pueden enviar a través de Internet mediante el conjunto de caracteres ASCII.

Dado que las URL contienen a menudo caracteres fuera del conjunto ASCII, la URL tiene que ser convertida en un formato ASCII válido.

La codificación de URL reemplaza caracteres ASCII inseguros con un "%" seguido de dos dígitos hexadecimales.

Las URL no pueden contener espacios. Se reemplaza un espacio con un signo más (+) o con %20.

Ej. enviar texto Hola, España!

http://www.w3schools.com/tags/html\_form\_submit.asp?text=Hola+Espa%C3%B1a%21

HTTP Es un Protocolo cliente – servidor.

Protocolo sobre capa de transporte TCP. Orientado a conexión.

Puerto http, por defecto: 80

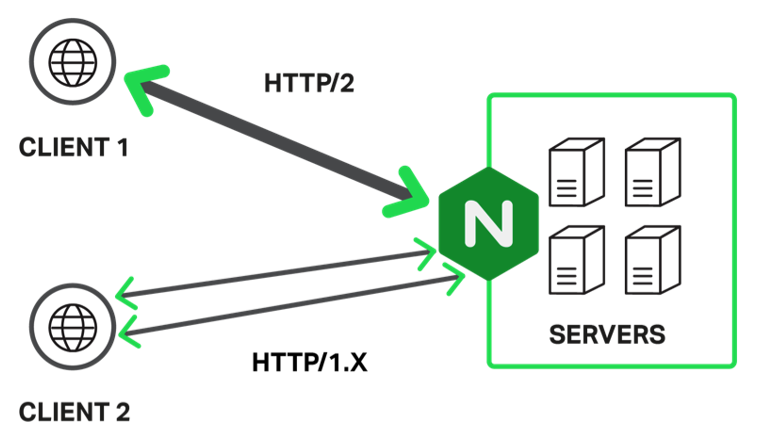
Puerto https (http + encriptación ssl), por defecto: 443

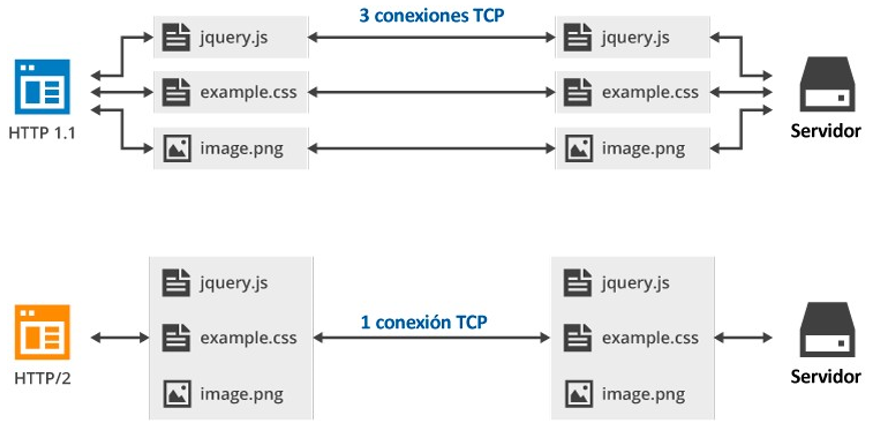
Diferentes versiones:

* HTTP 1.2
* HTTP 2

Mejoras de HTTP 2 sobre HTTP 1

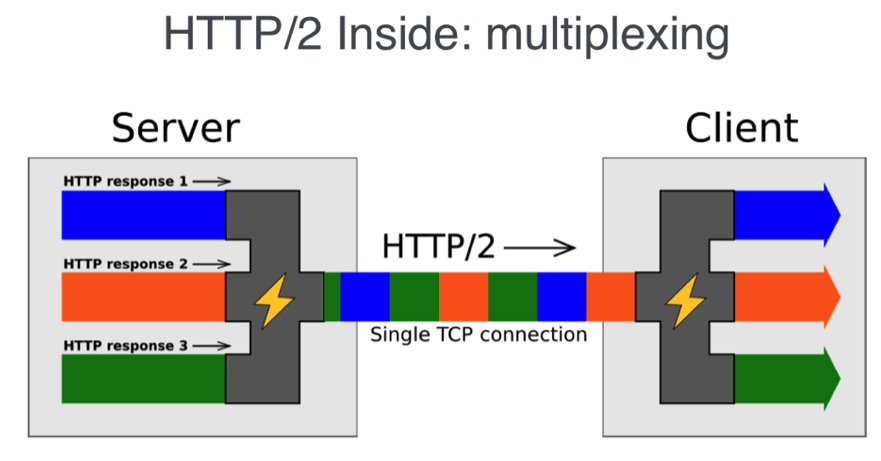
Una única conexión





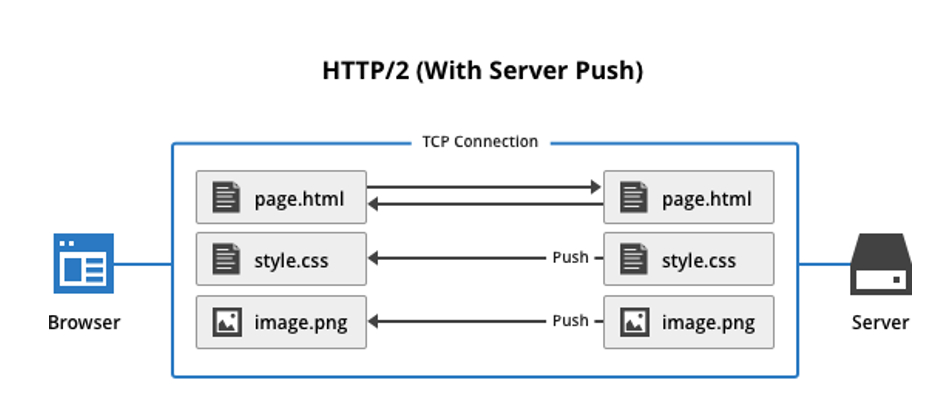
Eliminación de información redundante.

Multiplexación.



Protocolo binario frente a ineficiencias de protocolos texto.

Permite server push.



Compresión de cabeceras para transmitir menos información.

Priorización de flujos.

Métodos HTTP

o GET facturas/123456 lee la factura 123456

o POST facturas crea una nueva factura

o PUT facturas/123456 edita completamente una factura

o PATCH facturas/123456 edita parcialmente una factura

o DELETE facturas/123456 borra una factura

GET y POST aseguran soporte en todos los servidores.

· Métodos HTTP relacionados con las operaciones de un sitio CRUD:

o Create - POST

o Read - GET

o Update – PUT, PATCH

o Delete – DELETE

Aplicaciones sobrecargan la URL y utilizan POST y GET para operaciones Update y Delete.

o POST facturas/delete/123456

o POST facturas/update/123456 Lista completa de métodos:

HEAD – en la respuesta debe devolver sólo las cabeceras, echo headers HTTP.

GET

POST

PUT

DELETE

TRACE – devuelve en la respuesta la misma petición, echo completo para depurar.

OPTIONS – devuelve qué métodos puede atender una URL de petición determinada

CONNECT – para saber si un proxy da acceso a un servidor

Extensión métodos para WebDAV (Web Distributed Authoring and Versioning) control de versiones vía Web.

PATCH

SEARCH

COPY

LOCK

UNLOCK

MOVE

MKCOL

PROPFIND

PROPPATCH

MERGE

UPDATE

LABEL