# Modelo cliente servidor

* Cliente y servidor son PROCESOS, que pueden ejecutarse en máquinas diferentes o en la misma máquina.
* El servidor está a la espera de peticiones, nunca inicia la comunicación.
* La comunicación siempre la inicia el cliente, lanzando una petición al servidor.
* El servidor atiende la petición del cliente, o de varios clientes. Lo normal es que los servidores puedan atender a muchos clientes a la vez.

# Aplicaciones web

* Programa que usa un navegador (Firefox, Chrome, Edge, …) para interactuar con el usuario, y para realizar tareas en Internet.
* Siguen el modelo cliente-servidor. Cliente y servidor son PROCESOS, que pueden ejecutarse en máquinas diferentes o en la misma máquina.
  + El cliente envía una solicitud al servidor
  + En el servidor se procesa la petición, que puede hacer diferentes trabajos: acceso a BD, envío de mail, etc.
  + El servidor envía la respuesta al cliente

# Modelos de programación web

* Páginas estáticas.
  + Fundamentalmente HTML + CSS.
  + No requieren servidor, salvo si se quieren publicar en Internet.
* Páginas dinámicas en cliente
  + HTML + CSS + JavaScript
  + No requieren servidor, salvo si se quieren publicar en Internet.
* Páginas dinámicas en servidor
  + HTML + CSS + Lenguaje de servidor
  + Lenguajes de servidor: Java, C#, Python, PHP, Node.js (JavaScript)
  + Generan distintas versiones de la misma página en función de, por ejemplo:
    - Datos recibidos de la petición del cliente
    - Fecha / hora
    - Usuario conectado
    - Etc.

# Ventajas e inconvenientes de las aplicaciones web

* Ventajas:
  + Accesibles desde cualquier lugar con conexión a Internet.
  + Multiplataforma. cualquier sistema con navegador puede ejecutarlas.
  + Distribución simple. Los cambios se implementan en el servidor.
  + Actualización permanente. siempre disponible la versión más actualizada.
  + Clientes más ligeros. El servidor es el que debe soportar la mayor carga.
  + Escalabilidad. En general, son fácilmente escalables.
  + Más seguras: tanto en almacenamiento de datos como en exposición al hackeo. El recurso que proteger es fundamentalmente el servidor.
* Inconvenientes:
  + Dependencia de Internet. Sin conexión a Internet no funcionan.
  + Dependencia del servidor. Si el servidor falla, la aplicación falla.
  + Rendimiento limitado. Algunas cosas funcionan mejor en escritorio.
  + Latencia. Pueden afectar las condiciones de la red.
  + Funcionalidad limitada: cuando tenemos versión de escritorio y versión web de una aplicación, la versión web es más limitada (generalmente).
  + Consumo de datos: al usar Internet puede implicar un gasto adicional en datos, en conexiones que no sean de tarifa plana.
  + Expuestas a ataques DoS o DDoS, XSS, inyección SQL, etc.

# Generación de páginas en servidor

* Con tres grupos de tecnologías:
  + CGI
  + Lenguajes de script
  + Frameworks de desarrollo
* CGI:
  + Especificación para que el servidor llame a programas externos que se encargan de generar la salida HMTL.
  + Independiente del lenguaje. Cualquier lenguaje sirve para hacer un CGI.
  + Más lento que otras tecnologías, porque lanza procesos nuevos.
  + Versiones más modernas, como FastCGI mejoran reutilizando procesos.
* Lenguajes de script:
  + En general, funcionan más integrados con el servidor, con un sistema de módulos. Pero algunos siguen funcionando como CGI o FastCGI.
  + Menor rendimiento que lenguajes compilados.
* Frameworks de desarrollo
  + Conjunto de patrones, estructuras de aplicación, bibliotecas de clases y herramientas para desarrollar aplicaciones más robustas de forma más rápida.
  + Suelen seguir algún patrón de arquitectura. El más difundido es MVC: Modelo – Vista – Controlador.
  + Ventajas de los frameworks
    - Productividad. Permiten desarrollar aplicaciones rápidamente.
    - Fomentan el uso de buenas prácticas, al forzar el uso de patrones y estructuras específicas.
    - Facilitan la reutilización de código.
    - Seguridad: suelen incluir mecanismos para implementar la seguridad fácilmente.

# HTTP

* Protocolo utilizado por los navegadores y servidores web para comunicarse.
* Normalmente utiliza los puertos TCP 80 (HTTP) y TCP 443 (HTTPS), pero puede funcionar en otros puertos.
* Basado en texto. Los mensajes que se intercambian el cliente y el servidor son texto. Si hay que enviar binarios (ficheros, por ejemplo) se codifican para que puedan enviarse como texto.
* Es un protocolo sin estado. Cada petición de un cliente al servidor es independiente de las demás. El estado se puede implementar con sesiones (que a su vez usan cookies)

# URL

* Uniform Resource Locator (Localizador Uniforme de Recurso)
* Una URL es un tipo de URI (Uniform Resource Identifier – Identificador Uniforme de Recurso)
* Estructura de una URL:
* schema://[user[:password]@]host[:port][/path][?query][#fragment]
  + Schema: protocolo / forma de acceso al recurso.
  + User y password: Credenciales para acceder al recurso. Opcional.
  + Host: servidor en el que se ubica el recurso. Nombre DNS o IP.
  + Port: puerto TCP del servidor. Sólo si no es el estándar del protocolo utilizado.
  + Path: ubicación del recurso dentro del servidor
  + Query: para recursos dinámicos (ejecutables). Parámetros necesarios para su ejecución. Habitual en aplicaciones web.
  + Fragment: referencia interna dentro del recurso solicitado

# Mensajes HTTP

* Son mensajes basados en texto.
* Estructura de una petición:
  + Primera línea: Método (GET / POST / …), path (ubicación de recurso solicitado dentro del servidor) y versión de HTTP que se está utilizando
  + Líneas a partir de la 2: cabeceras de la petición. Una cabecera por línea. Cada cabecera es de la forma <nombre de cabecera>: <valor>
  + Después de las cabeceras, siempre hay una línea en blanco
  + Si la petición lleva asociado un cuerpo (es opcional), estará después de la línea en blanco. En el cuerpo de la petición van los datos enviados, como, por ejemplo, los datos de los campos de formularios, o los ficheros adjuntos.
* Estructura de una respuesta:
  + Primera línea: Versión de HTTP utilizada, código de estado y texto explicativo.
  + Líneas a partir de la 2: cabeceras de la respuesta. Igual que en la petición, una cabecera por línea. Cada una <nombre de cabecera>: <valor>
  + Después de las cabeceras, siempre hay una línea en blanco
  + Si la respuesta lleva asociado un cuerpo (es opcional), estará después de la línea en blanco. En el cuerpo de la respuesta van los datos enviados, como, por ejemplo, el HTML de una página, o el JSON de una respuesta de un servicio web.

# Métodos HTTP

* GET
  + Usado al escribir una URL en el navegador o hacer clic en un enlace.
  + Las peticiones no tienen cuerpo. Si tienen que pasar datos lo hacen en la URL, con la parte “query”.
  + Permite la recarga de una página sin problemas.
  + Las respuestas se pueden cachear en los proxies.
* POST
  + Usado en el envío de formularios.
  + Lo normal es que las peticiones tengan cuerpo, aunque no es obligatorio.
  + Aunque lleve datos en el cuerpo, también puede usarse la parte “query” de la URL para pasar información.
  + Los navegadores, al recargar la página, preguntan para confirmar el reenvío de los datos.
  + Las respuestas no se deben cachear en los proxies.
* Otros métodos: OPTIONS, HEAD, PUT, DELETE, PATCH, TRACE

# Cabeceras HTTP más interesantes

* Authorization: transmite las credenciales para autenticar y autorizar al usuario.
* Content-Type: tipo de contenido enviado. Puede ser el contenido enviado del cliente al servidor o del servidor al cliente.
* Content-Length: tamaño del cuerpo de la respuesta
* Cookie: cabecera exclusiva para peticiones. Envía las cookies del cliente al servidor.
* Set-Cookie: cabecera exclusiva para respuestas. Indica al cliente que debe guardar una cookie fijada por el servidor.

# Códigos de respuesta HTTP

* 100 – 199: Informativos. La petición se está procesando, pero ha terminado.
* 200 – 299: Éxito. Indican que una operación se ha completado con éxito. El más habitual es el 200 (Ok).
* 300 – 399: Redirección. Indican al cliente que debe repetir la petición, usando una dirección diferente.
* 400 – 499: Errores en la petición del cliente, que el servidor no ha podido procesar. El problema no está en el servidor. Está en cómo está haciendo la petición el cliente. El más conocido es el 404 (Not found).
* 500 – 599: Errores de proceso en el servidor. Aunque la petición era correcta (no se devuelve error 4xx), se ha producido un error interno.