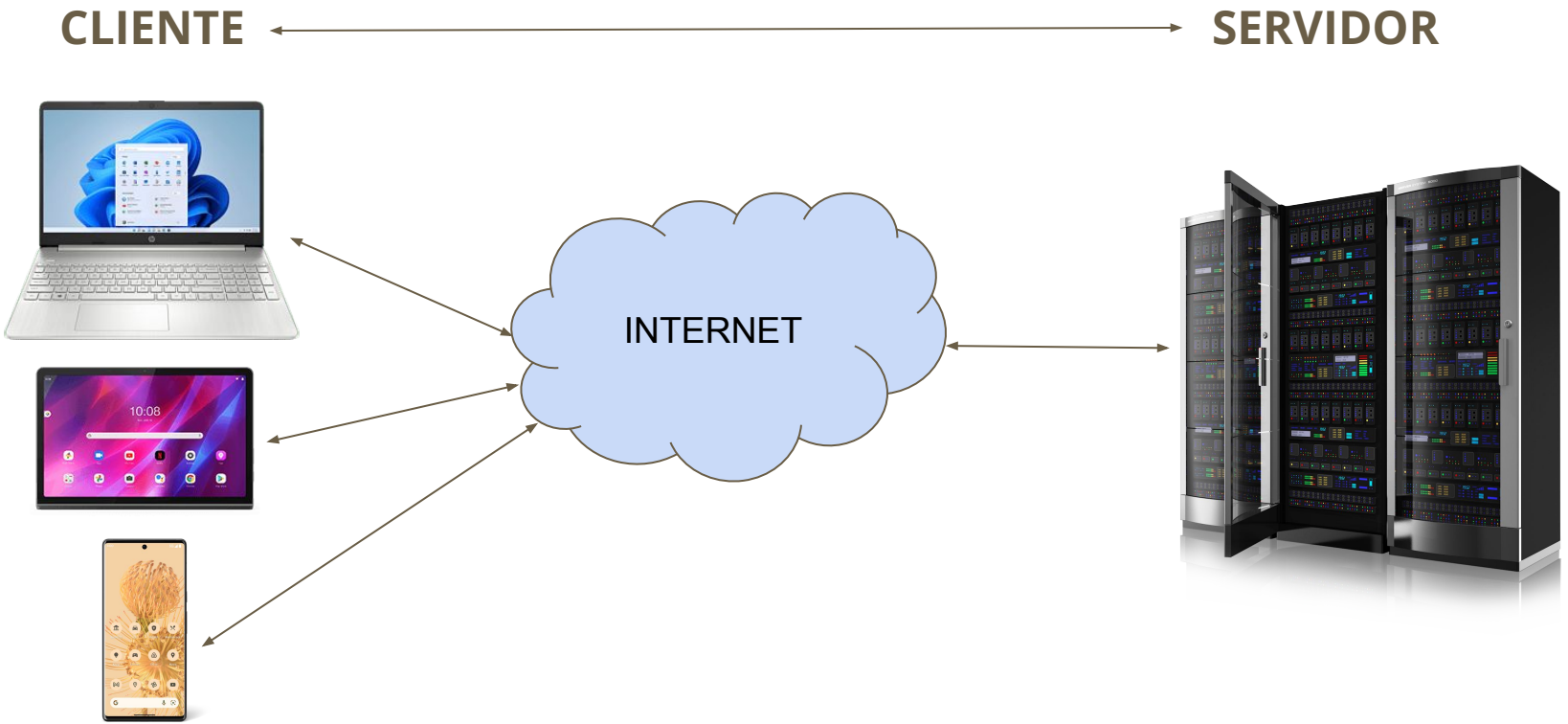
UT1: Introducción al Desarrollo Web en Entorno Servidor

Desarrollo Web en Entorno Servidor

ÍNDICE

1. Modelo Cliente-Servidor



1. Modelo Cliente-Servidor

CLIENTE

- Puede ser cualquier dispositivo con acceso a internet
- Utilizan un navegador web para acceder a las páginas o servicios web que proporciona el lado servidor
- Los navegadores interpretan los archivos que forman las páginas web (HTML, CSS o JS) y los datos que utilizan (JSON o XML)
- En el mundo del desarrollo se denomina Front-Side o Lado Cliente

1. Modelo Cliente-Servidor

SERVIDOR

- Escuchan y procesan las distintas peticiones HTTP que realizan los clientes (métodos GET, POST, etc)
- Contienen y generan los archivos de las páginas web que son enviados a los clientes
- Utilizan BBDD para crear/leer/almacenar/eliminar los datos que se utilizan para generar las páginas web. Esto permite la persistencia de la información.
- Sobre ellos recae la mayor parte de la responsabilidad en cuanto a la seguridad de las páginas y servicios web

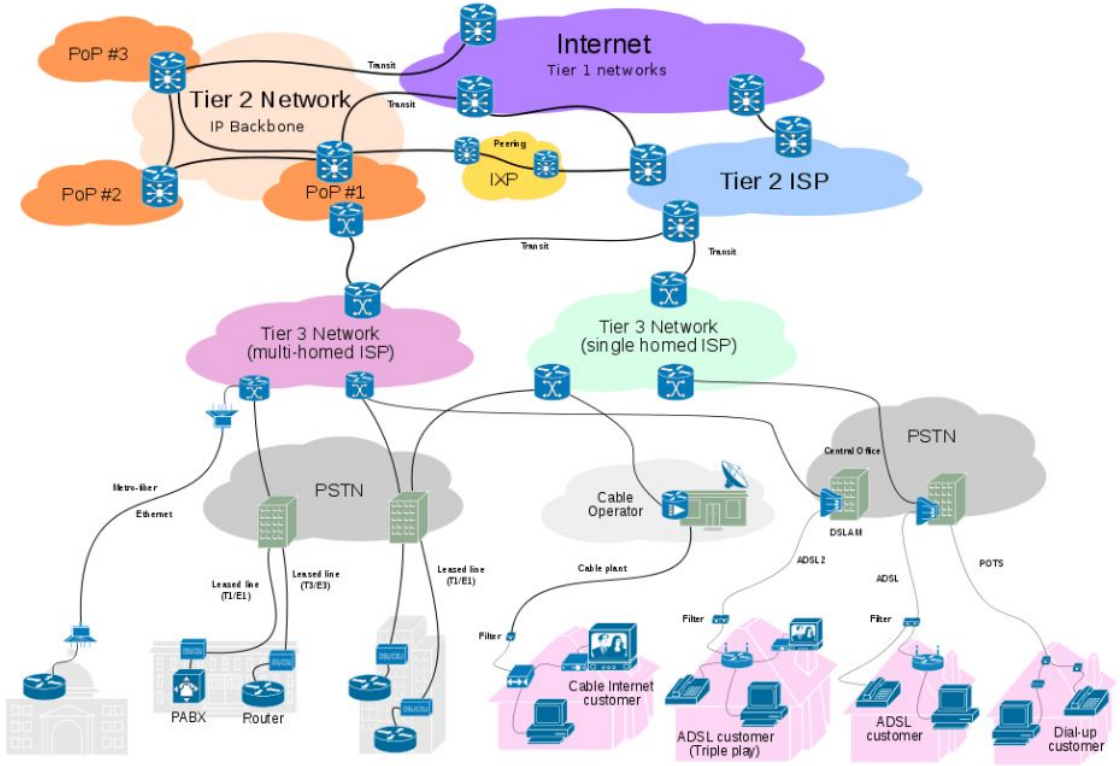
1. Modelo Cliente-Servidor

INTERNET

- Es la red de redes, ofreciendo un servicio global de computadoras interconectadas
- Utiliza el protocolo TCP/IP para conectar millones de dispositivos en todo el mundo
- Uno de sus servicios más utilizados es la World Wide Web (WWW o la Web). No hay que confundir Internet y WWW.
- La Web es un sistema de distribución de documentos de hipertexto interconectados y accesibles vía Internet, a través del protocolo HTTP

1. Modelo Cliente-Servidor

INTERNET

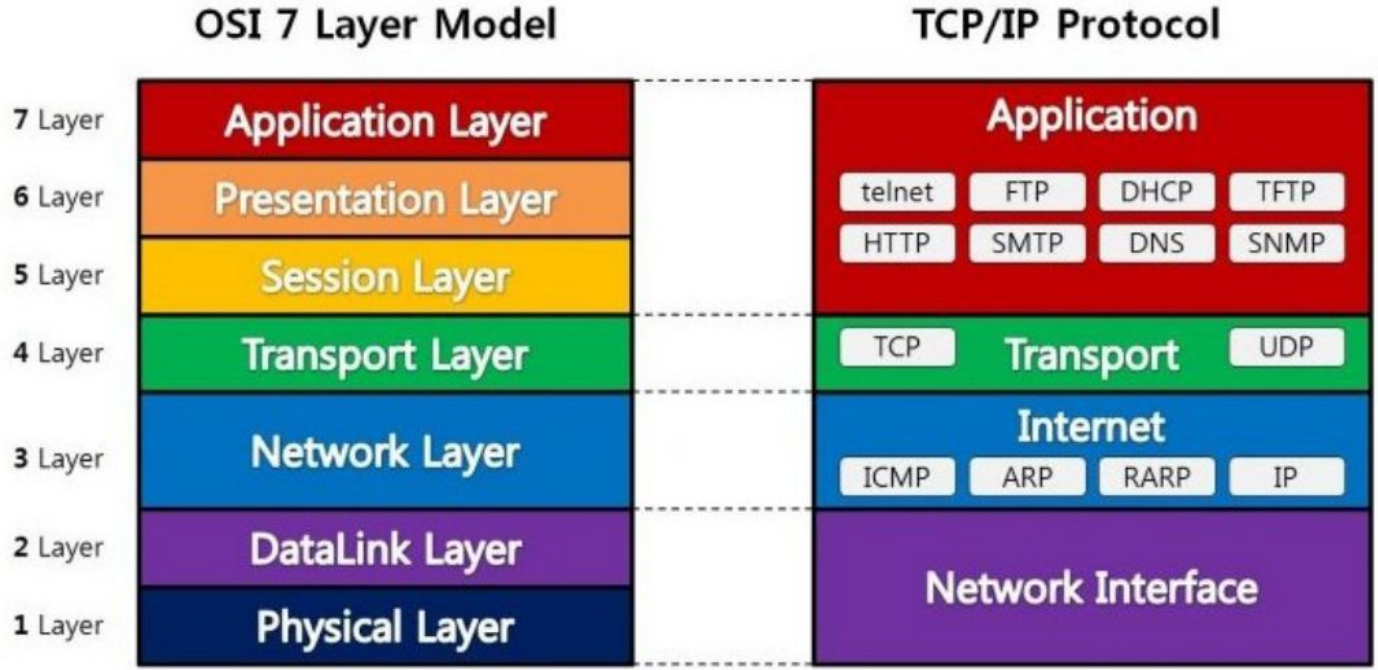


2. Protocolo HTTP

- Hypertext Transfer Protocol o HTTP (protocolo de transferencia de hipertexto) es el protocolo de comunicación que permite las transferencias de información en la World Wide Web.
- HTTP fue desarrollado en 1999 por el World Wide Web Consortium (W3C) junto con la Internet Engineering Task Force (IETF)
- HTTP sigue un modelo de comunicación de tipo petición-respuesta (**request-response**) entre un cliente y un servidor
- HTTP es un protocolo **sin estado**, lo que quiere decir que las peticiones son independientes unas de otras, no se guarda información (estado) entre peticiones
- Las aplicaciones Web sí necesitan, por contra, vincular las diferentes peticiones de un mismo usuario. Para ello se ha creado el concepto de **sesión** en el servidor. Para ello se utilizan **cookies** o técnicas como URL rewriting

2. Protocolo HTTP

HTTP es un protocolo de nivel de aplicación del modelo TCP/IP



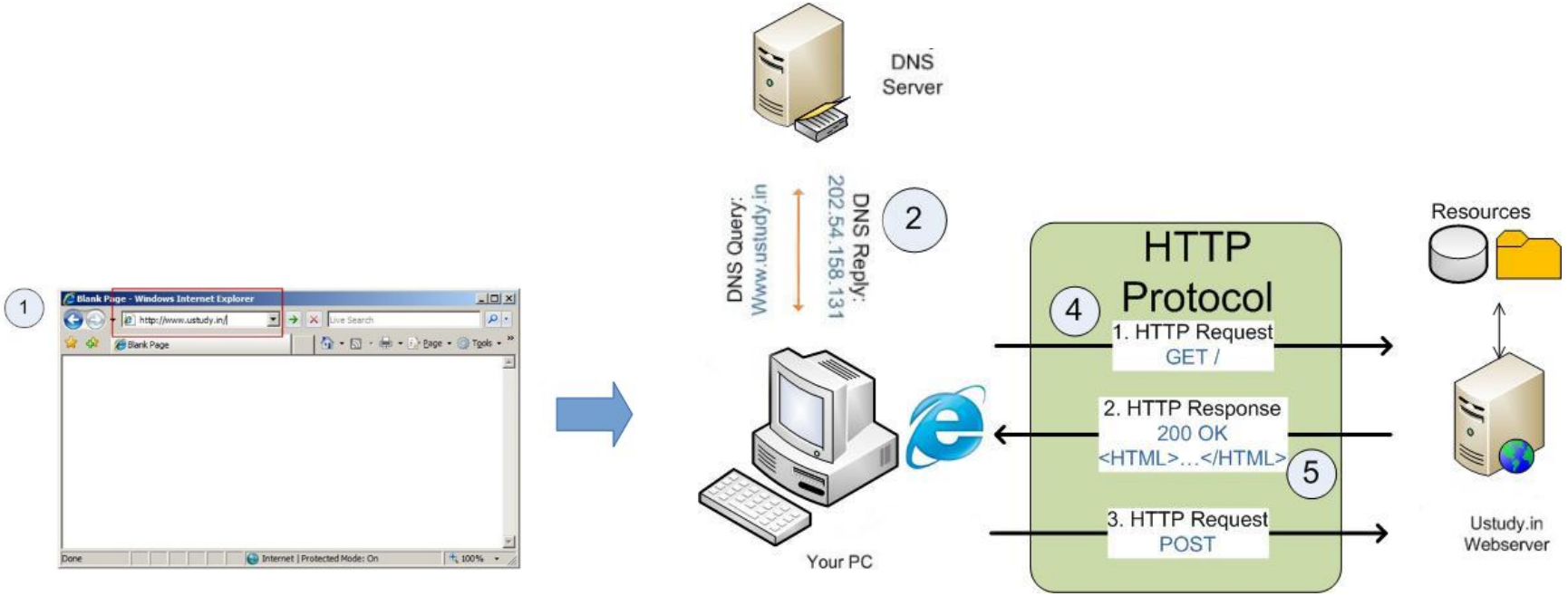
2. Protocolo HTTP

ESQUEMA DE INTERACCIÓN

- El cliente (o user agent) envía una petición HTTP (request) a un servidor Web (el cliente lo hace a través de un client side socket)
- El servidor está escuchando (server side socket) por peticiones en un puerto TCP, que usualmente es el 80. Cuando recibe la petición busca el recurso solicitado y lo devuelve al cliente
- Los recursos se identifican por URIs (Uniform Resource Identifiers) (o, más específicamente, Uniform Resource Locators (URLs)) usando los esquemas http: o https:

2. Protocolo HTTP

ESQUEMA DE INTERACCIÓN



2. Protocolo HTTP

MÉTODOS HTTP

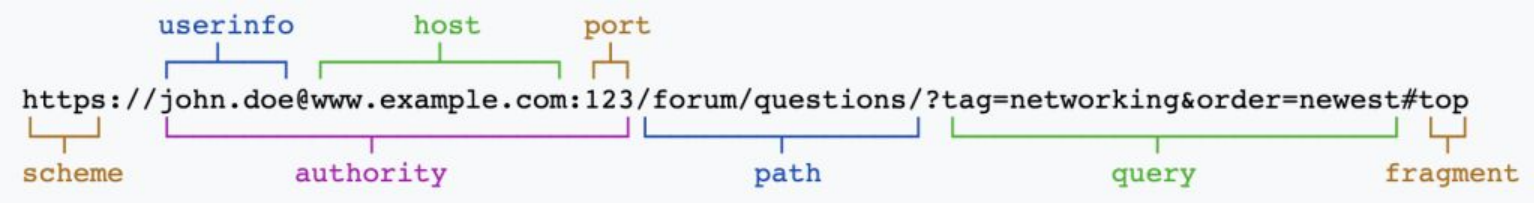
- **OPTIONS**. Solicita opciones de comunicación disponibles
- **GET**. Solicita un recurso al servidor identificándolo por su URI
- **HEAD**. Idéntico a GET pero solo devuelve cabeceras de respuesta
- **POST**. Envío de datos al servidor (campos de formularios, etc.)
- **PUT**. Almacenar un documento en la URI especificada
- **DELETE**. Borrar el documento indicado por la URI
- **TRACE**. (ECO) Obtener del servidor copia de la petición enviada
- **CONNECT**. Reservado

2. Protocolo HTTP

URI (URL, URN)

- Un identificador de recursos uniforme o URI —del inglés uniform resource identifier— es una cadena de caracteres que identifica los recursos de una red de forma unívoca. La diferencia respecto a un localizador de recursos uniforme (URL) es que estos últimos hacen referencia a recursos que, de forma general, pueden variar en el tiempo.
- Ejemplo:

```
scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]
```



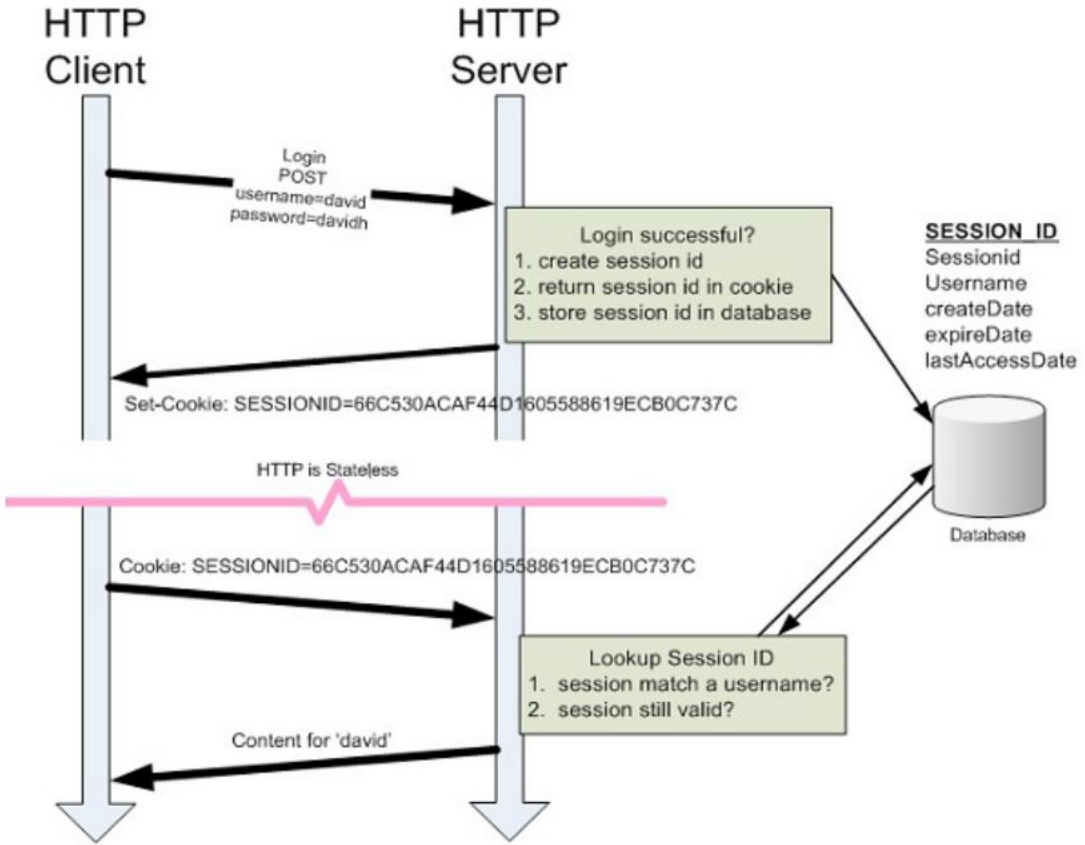
2. Protocolo HTTP

URI (URL, URN)

- Un URI consta de las siguientes partes:
  - **Esquema**: nombre que se refiere a una especificación para asignar los identificadores, e.g. urn:, tag:, cid:. En algunos casos también identifica el protocolo de acceso al recurso, por ejemplo http:, mailto:, ftp:, etc.
  - **Dominio**: elemento jerárquico que identifica el host (por ejemplo //www.example.com).
  - **Ruta**: Información usualmente organizada en forma jerárquica, que identifica al recurso en el ámbito del esquema URI y la autoridad de nombres (e.g. /domains/example).
  - **Consulta**: Información con estructura no jerárquica (usualmente pares "clave=valor") que identifica al recurso en el ámbito del esquema URI y la autoridad de nombres. El comienzo de este componente se indica mediante el carácter '?'.
  - **Fragmento**: Permite identificar una parte del recurso principal, o vista de una representación del mismo. El comienzo de este componente se indica mediante el carácter '#'.

2. Protocolo HTTP

Sesiones HTTP y Cookies



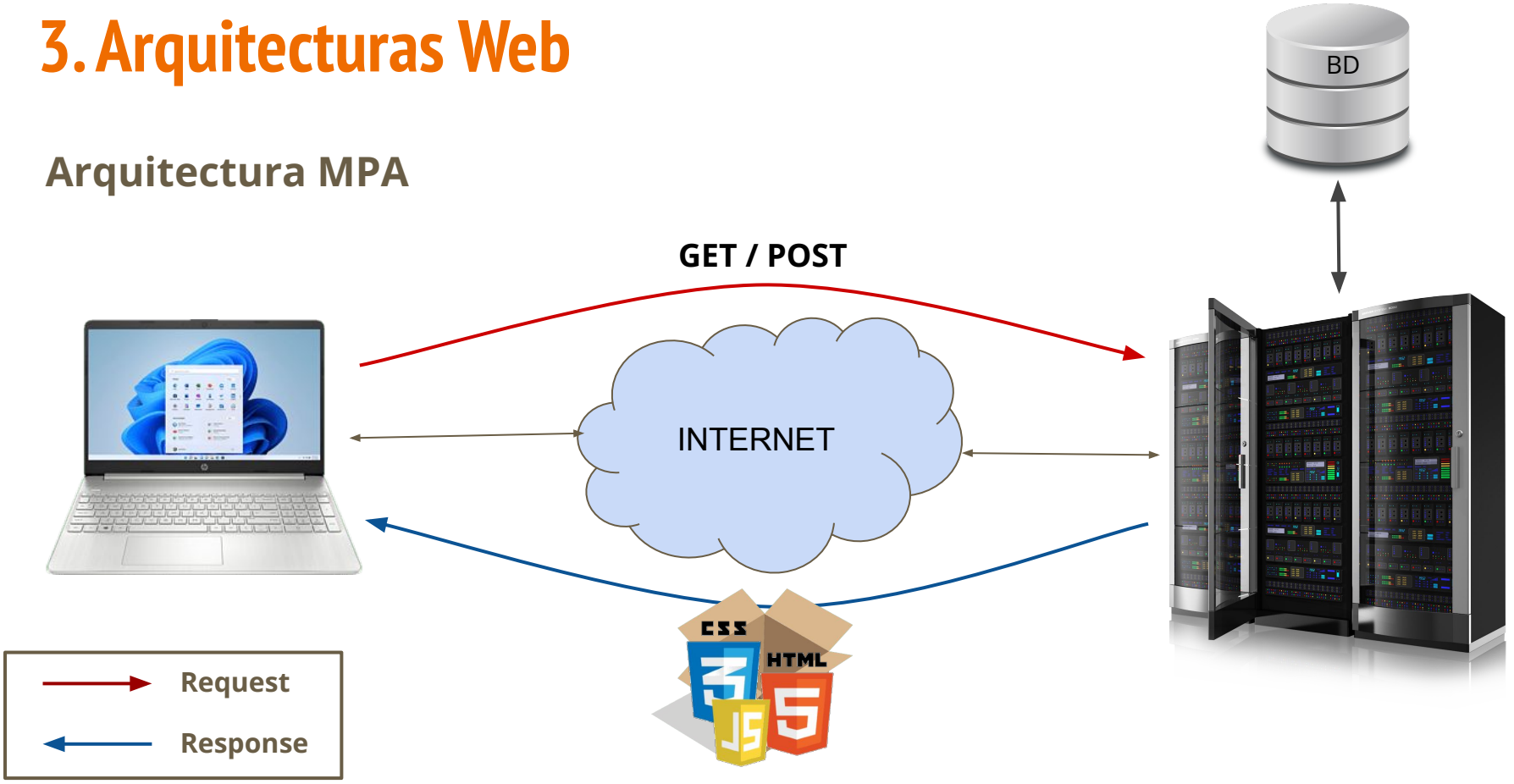
3. Arquitecturas Web

Trataremos dos tipos de arquitecturas de páginas web:

- **MPA** (Multi Page Application) → Arquitectura clásica
  - Cada petición al servidor devuelve una página web completa
  - Cada página muestra su contenido y se conecta con links con las demás
  - Todas las páginas son generadas en el servidor
- **SPA** (Single Page Application) → Arquitectura moderna
  - Sólamente la primera petición al servidor devuelve una página
  - El resto de peticiones al servidor solo devuelve datos
  - Es el navegador web del cliente el que construye las sucesivas páginas durante la navegación
  - A la aplicación servidor se suele llamar **API** (*Application Programming Interface*) en esta arquitectura

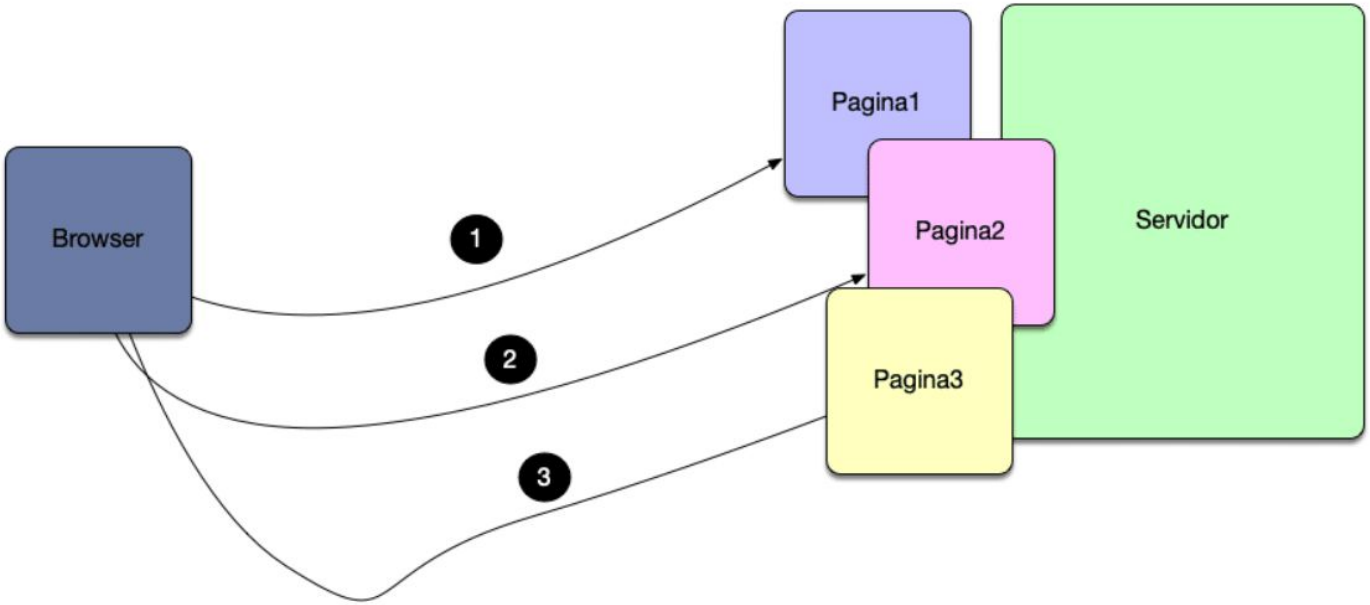
3. Arquitecturas Web

Arquitectura MPA



3. Arquitecturas Web

Arquitectura MPA



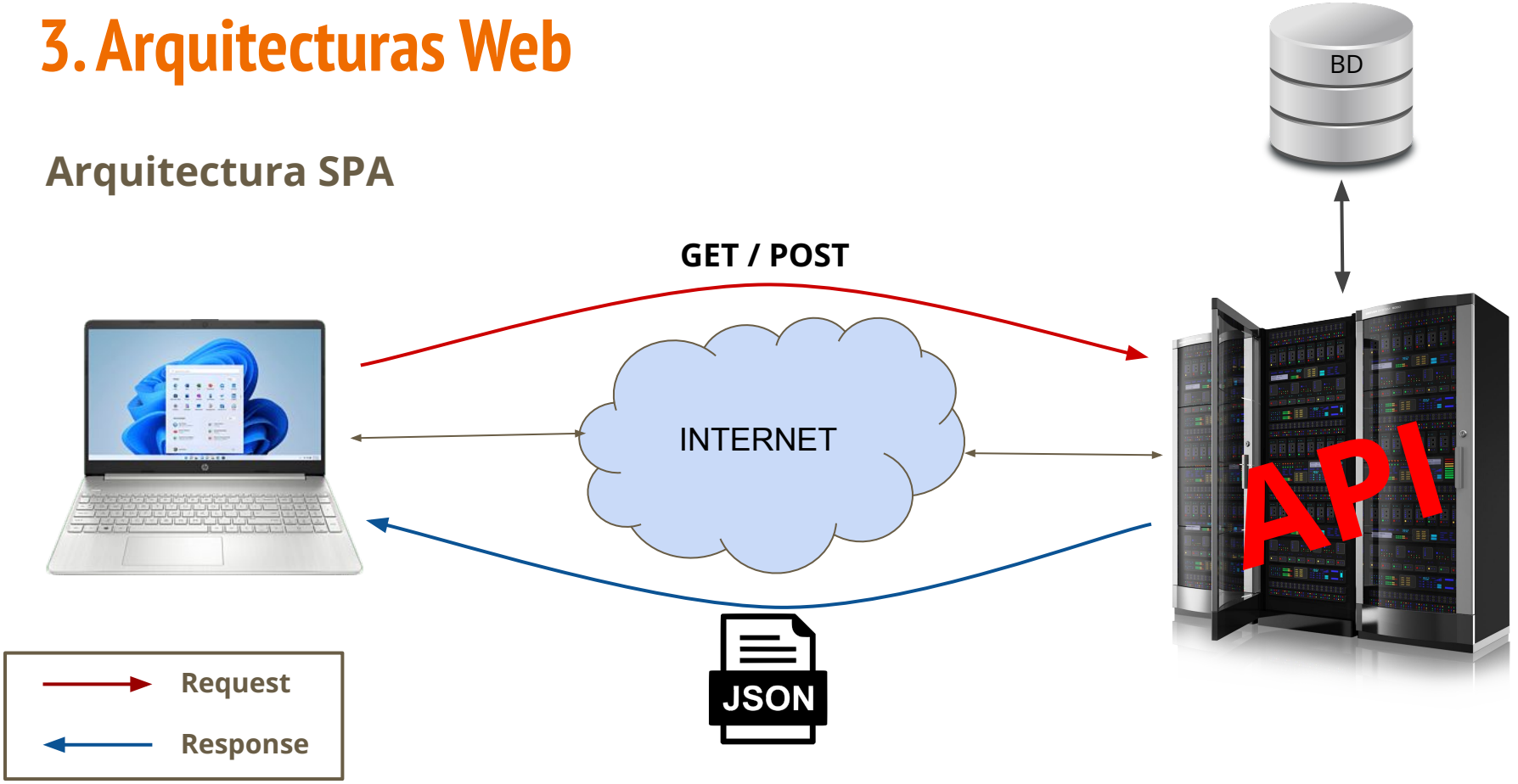
3. Arquitecturas Web

Arquitectura MPA

La navegación entre páginas se realiza de una forma natural.

3. Arquitecturas Web

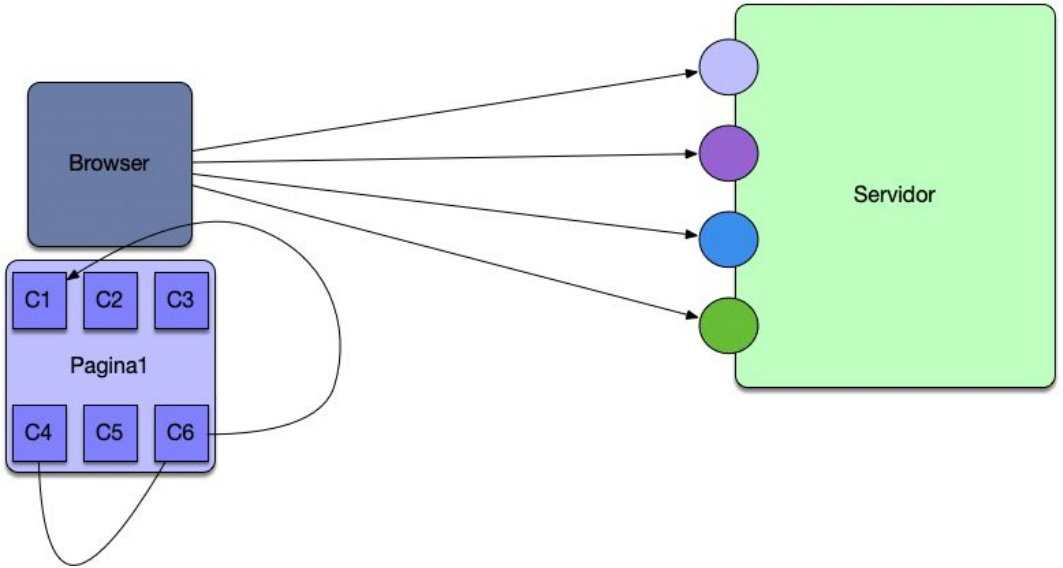
Arquitectura SPA



3. Arquitecturas Web

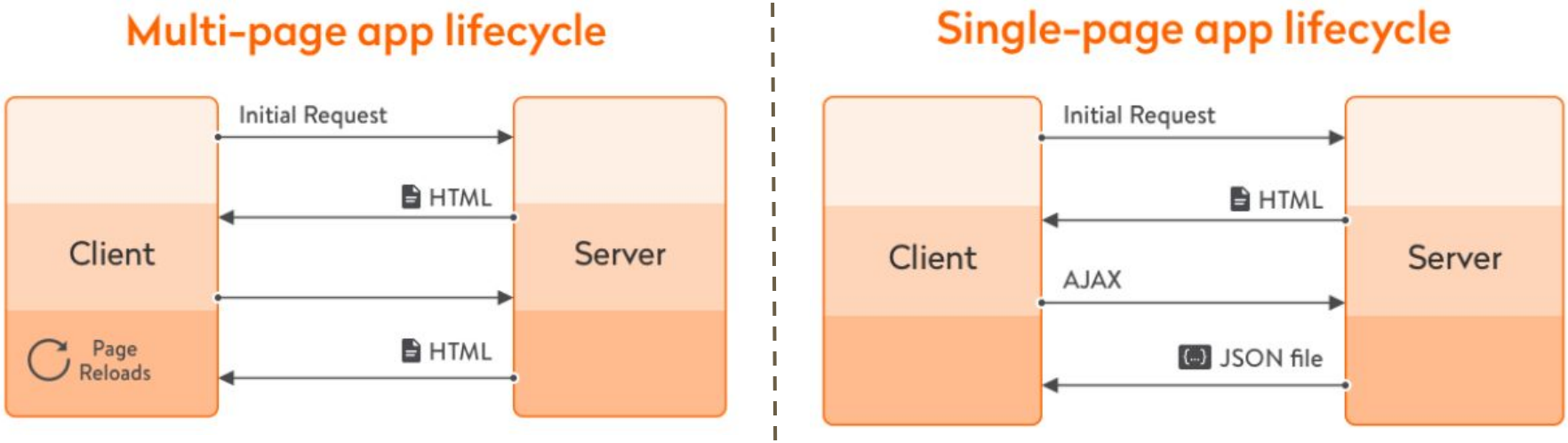
Arquitectura SPA

Se utiliza JavaScript para solicitar datos al servidor y actualizar partes de la página



3. Arquitecturas Web

MPA vs SPA



4. Páginas Estáticas y Dinámicas

Páginas Estáticas

- Son aquellas que se encuentran almacenadas en el servidor en su forma y apariencia definitivas.
- El servidor únicamente envía los ficheros de la página (HTML, CSS, JS) al cliente cuando éste lo solicita.
- Son útiles para mostrar una información concreta.
- Siempre tienen la misma apariencia.
- Si se quieren modificar hay que editar los archivos (HTML, CSS y JS) de la página manualmente.

4. Páginas Estáticas y Dinámicas

Páginas Dinámicas

- Su contenido puede variar en función de diversas variables:
  - El navegador que realice la petición al servidor
  - El tipo de dispositivo en el que vaya a ser cargada la página
  - El usuario que se haya identificado en el sistema (Ejem: un buzón de correo)
  - Las acciones realizadas con anterioridad
- El contenido de estas páginas suele variar por que se crean con datos distintos obtenidos de la Base de Datos del servidor.
- Las páginas dinámicas pueden existir en arquitecturas MPA y SPA. En el primer caso su apariencia final se genera en el servidor y en el segundo caso en el navegador.

4. Páginas Estáticas y Dinámicas

Páginas Dinámicas

- El proceso de generación de este tipo de páginas, en arquitectura MPA es:
  - El cliente web (navegador) de tu ordenador solicita a un servidor web una página web.
  - El servidor busca esa página y la recupera.
  - En el caso de que se trate de una página web dinámica, es decir, que su contenido deba ejecutarse para obtener el HTML que se devolverá, el servidor web contacta con el módulo responsable de ejecutar el código y se lo envía.
  - Como parte del proceso de ejecución, puede ser necesario obtener información de algún repositorio, como por ejemplo consultar registros almacenados en una base de datos.
  - El resultado de la ejecución será una página en formato HTML, similar a cualquier otra página web no dinámica.
  - El servidor web envía el resultado obtenido al navegador, que la procesa y muestra en pantalla

4. Páginas Estáticas y Dinámicas

Páginas Dinámicas

- Este procedimiento tiene lugar constantemente mientras consultamos páginas web. Por ejemplo, cuando consultas tu correo en GMail, HotMail, Yahoo o cualquier otro servicio de correo vía web, lo primero que tienes que hacer es introducir tu nombre de usuario y contraseña. A continuación, lo más habitual es que el servidor te muestre una pantalla con la bandeja de entrada, en la que aparecen los mensajes recibidos en tu cuenta. Esta pantalla es un claro ejemplo de una página web dinámica
- Obviamente, el servidor no envía esa misma página a todos los usuarios, sino que la genera de forma dinámica en función de quién sea el usuario que se conecte. Para generarla ejecuta un programa que obtiene los datos de tu usuario (tus contactos, la lista de mensajes recibidos) y con ellos compone la página web que recibes desde el servidor web

5. Desafíos a Enfrentar en el Desarrollo Web

- **Rendimiento**
  - Ofrecer alto throughput (n.º transacciones por segundo que puede atender el sistema
  - Bajo tiempo de respuesta del servidor
  - Asegurar la disponibilidad del servicio
- **Escalabilidad**
  - Capacidad de ampliar el sistema cuando aumenta la carga de trabajo
- **Seguridad**
  - Evitar: suplantación, modificación de recursos sin autorización (tampering), alterar evidencias, revelación de información, denegación de servicio (DoS), elevación de privilegios.

5. Desafíos a Enfrentar en el Desarrollo Web

- **Accesibilidad**
  - Es la práctica inclusiva de garantizar la accesibilidad a los sitios web, y que las herramientas y las tecnologías estén diseñados y desarrollados para que las personas con discapacidad puedan usarlas
- **Usabilidad y experiencia de usuario**
  - Enfoque de diseño del software para ser lo más eficaz, eficiente y satisfactorio. ISO 9241-11 define usabilidad como: "grado en que un producto puede ser usado por determinados usuarios para conseguir objetivos específicos con efectividad, eficiencia y con satisfacción en un contexto de uso"
- **Inclusión**
  - Diseño inclusivo, universal y para todos implica el diseño de sitios web, de forma que lleguen a todo tipo de usuarios. Un área importante es el Responsive Web Design (RWD), que permite que el sitio Web se adapte al dispositivo desde el que se accede (ejemplo: móvil). Inclusión abarca, además, el acceso y la calidad del hardware, software y conectividad a Internet; conocimientos de los usuarios; ubicación geográfica y lenguaje, así como la edad y la discapacidad (aunque ésta se trata más específicamente en Accesibilidad)

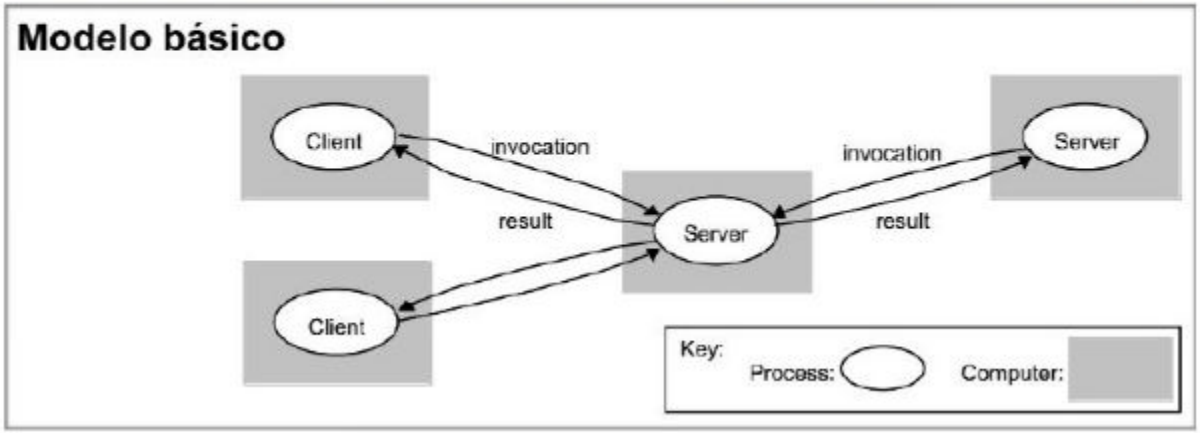
5. Desafíos a Enfrentar en el Desarrollo Web

- **Integración con otras aplicaciones y backend**
  - Nuestra aplicación ofrece un **front-end** (en nuestro caso un cliente web, denominado cliente ligero), con el que los usuarios pueden trabajar.
  - Además desarrolla una lógica de negocio, proporcionando servicios a los usuarios.
  - Para desarrollar esta lógica de negocio la aplicación precisa acceder a los datos del **back-end**; ya sea de forma directa (conectando directamente con bases de datos), o empleando servicios de terceros (otras aplicaciones).
  - En este segundo caso aparece el concepto de **Middleware**, que es una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red) para facilitar el intercambio de datos entre aplicaciones, proporcionando una API para la programación y manejo de aplicaciones distribuidas.
  - Sea cual sea el enfoque (normalmente se combinan ambos), nuestras aplicaciones deberán utilizar diversos protocolos, conectores, frameworks para integrar datos procedentes de diversas fuentes, consolidarla y generar información en base a esos datos.

6. Arquitecturas Hardware-Software

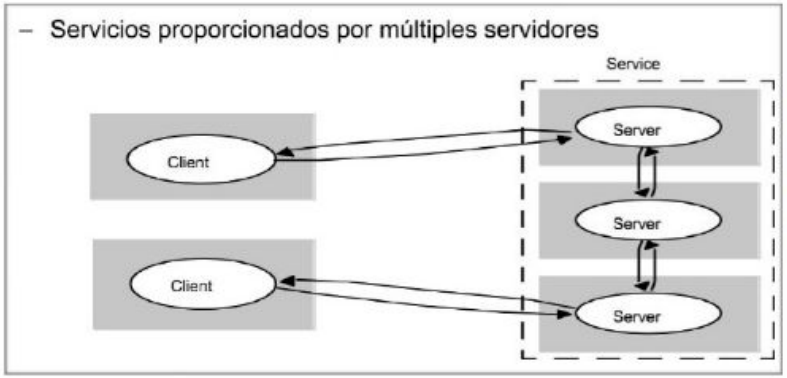
Ejemplo: Configuración de servidores

Los Sistemas Distribuidos son los sistemas de software más complejos, la corporación Nortel Networks por ejemplo, crea switches los cuales pueden contener entre 25-30 millones de líneas de código, interviniendo 3000 desarrolladores de software, y con un ciclo de vida de 20 años para actualizar.

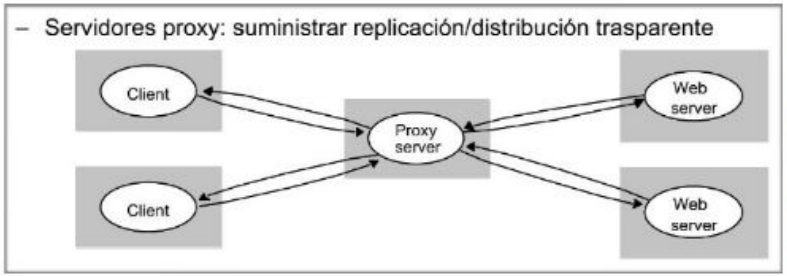


6. Arquitecturas Hardware-Software

Ejemplo: Configuración de servidores



(...) Muchos servicios de comercio Web están implementados en diferentes servidores. son muy confiables gracias a que mantienen bases de datos replicadas o distribuidas.



(...) Uso de Caching. Los servidores proxy mantienen caches, como almacenes de recursos solicitados recientemente. Son redes Utilizados frecuentemente en motores de búsqueda como google,etc.

6. Arquitecturas Hardware-Software

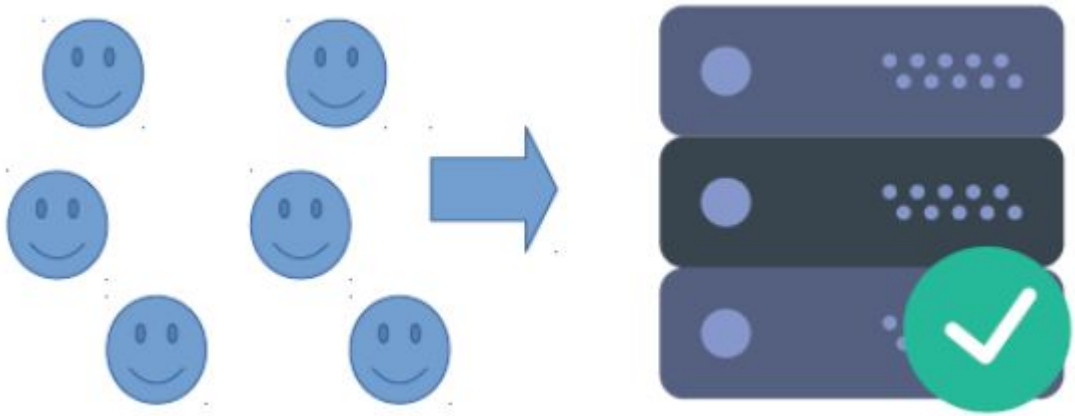
Problema: Aplicación web → Alto número de potenciales clientes

- La aplicación responde bien hasta un número determinado de usuarios, pero cuando se sobrepasa la capacidad del sistema éste se satura y deja de dar servicio. El cliente recibe Errores HTTP 5xx

6. Arquitecturas Hardware-Software

Problema: Aplicación web → Alto número de potenciales clientes

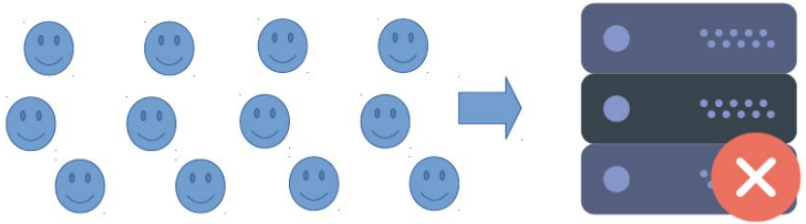
- Una solución es aumentar la capacidad del servidor (comprar uno más grande)



6. Arquitecturas Hardware-Software

Problemas para aumentar la capacidad del sistema (escalabilidad)

- El problema es que nuevos aumentos en la carga del sistema hacen que necesitemos cada vez máquinas “más gordas”

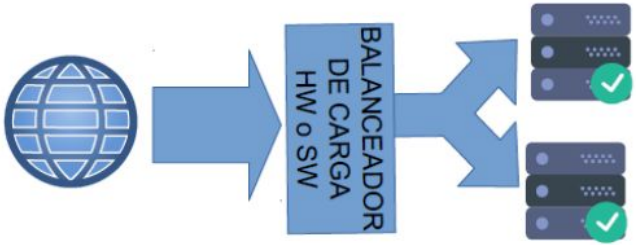


- Llega un punto que no podemos ampliar más la máquina para acomodar cargas mayores. Hay limitaciones físicas que nos impiden ampliar continuamente el servidor.

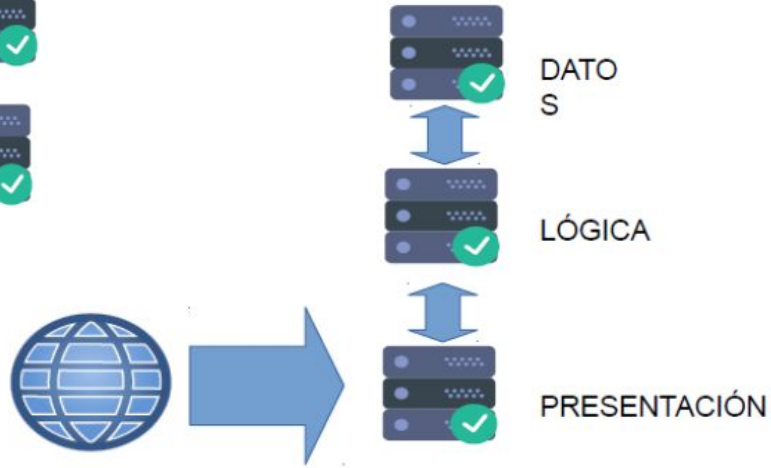
6. Arquitecturas Hardware-Software

Soluciones para la Escalabilidad del Sistema

- Escalabilidad horizontal
  - Creamos múltiples réplicas del sistema y se distribuye la carga entre ellas. El balanceador (hardware o software) reparte la carga entre los clones



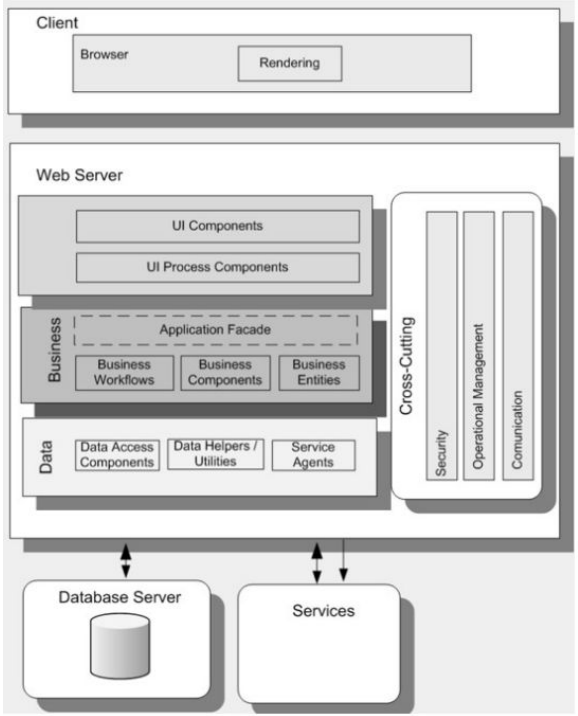
- Escalabilidad vertical
  - En este caso “partimos” el sistema entre varias máquinas. Cada una de ellas será responsable de una parte del mismo, de forma que la carga se distribuya.



6. Arquitecturas Hardware-Software

Escalabilidad Vertical: arquitectura software de división por capas

- Capa de presentación
  - Subcapa componentes de presentación
  - Subcapa de lógica de presentación
- Capa de lógica de negocio
  - Fachada (facade) de servicios
  - Componentes, lógica de negocio
- Capa de acceso a datos o persistencia
- Capa de infraestructura (transversal)
  - Seguridad
  - Gestión de servicios, etc.
- Helpers
  - Clases de transferencia entre capas

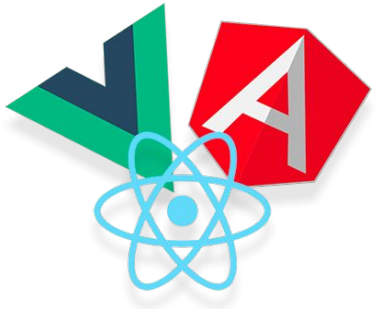


6. Tecnologías Usadas en el Desarrollo Web

Lado Cliente

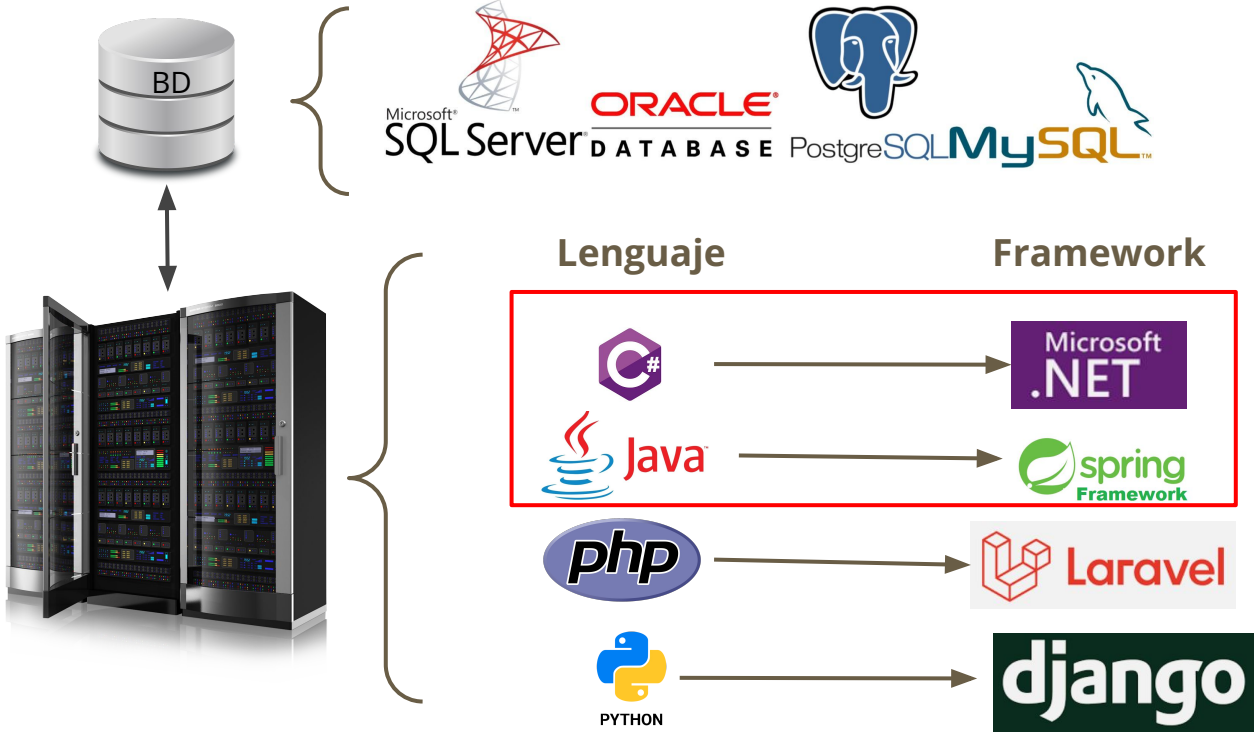
Lenguajes

Frameworks



6. Tecnologías Usadas en el Desarrollo Web

Lado Cliente



!Esto es lo que aprenderemos¡