Arquitectura web

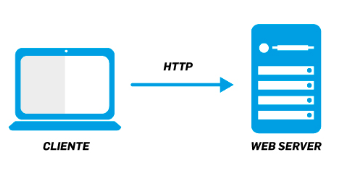
Breve reseña

Se trata entonces de una arquitectura cliente-servidor en la que cada dispositivo electrónico en la red (internet, intranet o extranet) actúa como cliente o servidor lo que implica la comunicación entre procesos hacen peticiones (clientes) y procesos que responden a esas peticiones (servidores). Esta comunicación es posible gracias al protocolo HTTP.

**Historia:** A partir del desarrollo de ARPANET en 1969 empieza un crecimiento vertiginoso del uso de la internet.

En 1990 Tim Berners-Lee creo la WWW, la “WorldWideWeb” que realizo la primera conexión desde un navegador a un servidor web mientras trabajaba en CERN desarrollando así, las tres tecnologías fundamentales de la web que son: HTML I URL I HTTP.

* **HTML** (HyperText Markup Lenguage). Lenguaje de marcado o etiquetado que se emplea para escribir los documentos o páginas web
* **URL** (Universal Resouce Locator). El localizador de recursos uniforme, sistema de localización o direccionamiento de los documentos web.
* **HTTP** (HyperText transfer Protocol). El lenguaje con el que se comunica el navegador con el servidor web y el que se emplea para trasmitir los documentos web.



En 1994 (1 de octubre) Tim Berners-Lee abandona el CERN y funda la W3C, en inglés, “World Wide Web Consortium”, organismo internacional que propone recomendaciones y estándares web que aseguran el crecimiento de la World Wide Web.

* En 1991 surge Http definido como “protocolo de red para sistemas de información hipermedia distribuidos”.
* Muy próximo aparece HTML1, es el lenguaje de mercado predominante de las páginas web.
* En 1995 Netcape creo JavaScript, un lenguaje de secuencias de comandos basado en prototipos y “orientado a objetos”. El objetivo de este lenguaje de programación fue darle capacidad de ejecución al cliente de esta arquitectura web, o sea, al navegador.
* En 1998 aparecen las hojas de estilo, en su versión 2. Se denominada CSS, del inglés “Cascading Style Sheets”, que es un lenguaje de hojas de estilo empleado para describir la semántica de presentación de un documento, en este caso un documento web.



Arquitectura de las aplicaciones web

Las aplicaciones web se basan en una arquitectura cliente/servidor. Es decir que por un lado está el cliente (navegador) y por otro lado el servidor. Existen diferentes variantes de la arquitectura básica según como se implemente, pero es importante mencionar que en tecnología la mayoría de estructuras está compuesta por capas.

Generalmente las historias se escriben en lenguaje que el usuario pueda entender y que refleje una descripción sintetizada de lo que este desea. En lo posible se debe tratar de eliminar ambigüedades y malas interpretaciones.

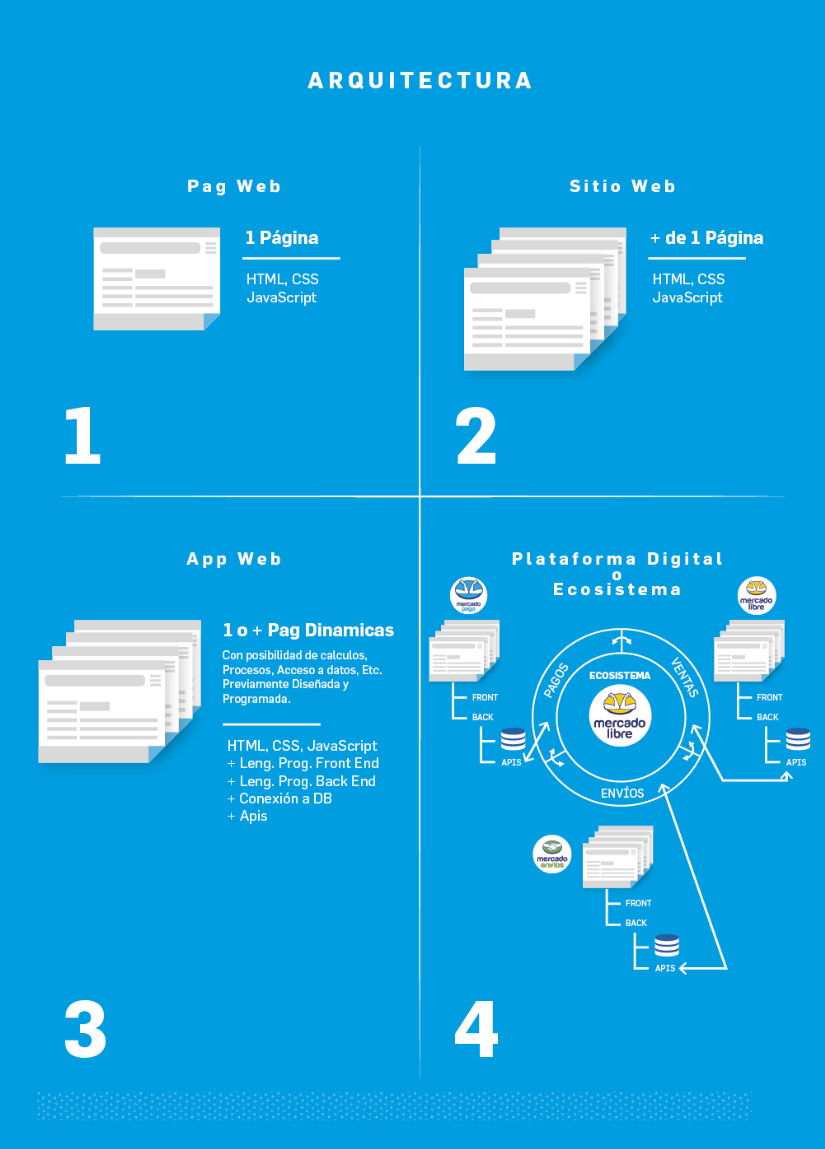
Un grupo de páginas web dinámicas se conceptualiza como Front End (pensadas para que el cliente acceda) y el otro grupo de páginas dinámicas web como Back End (pensadas para el procesamiento y acceso de datos), además de que la base de datos puede existir otro servidor. Esto da lugar a un concepto muy importante de POO de DESACOPLAMIENTO, en este caso el diseño del Front End, Back End y base de Datos puede desacoplarse. Pero es lo veremos más adelante

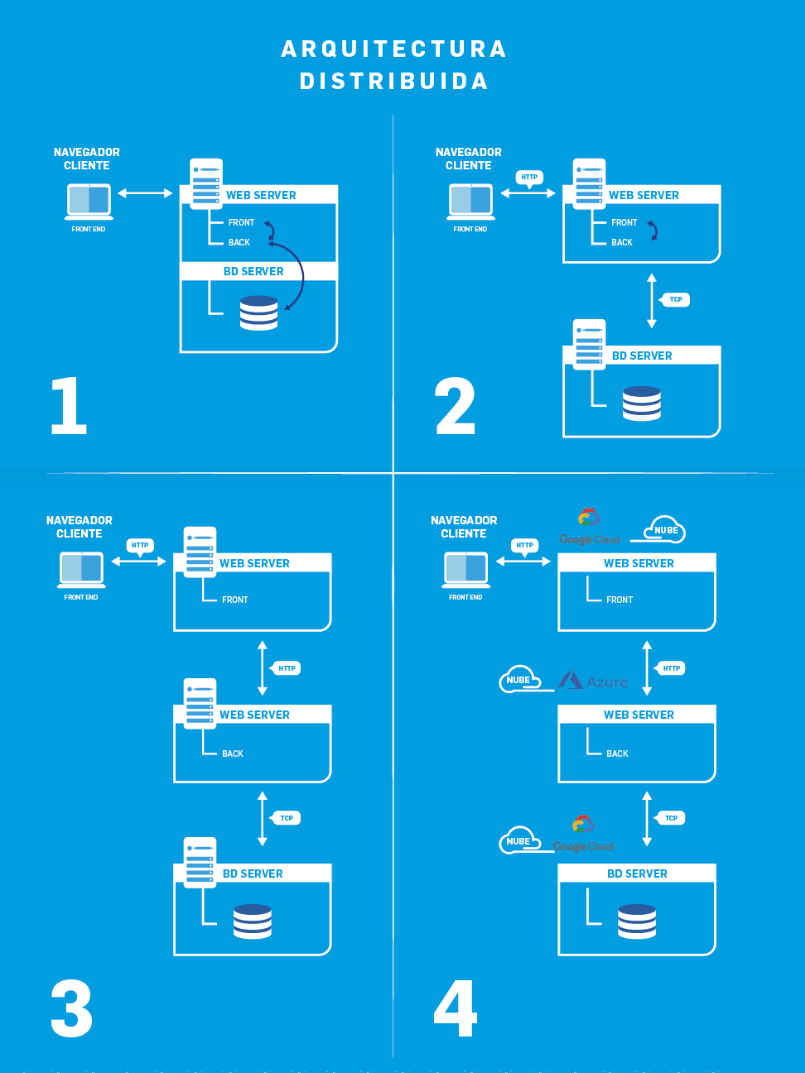
Variantes de arquitectura

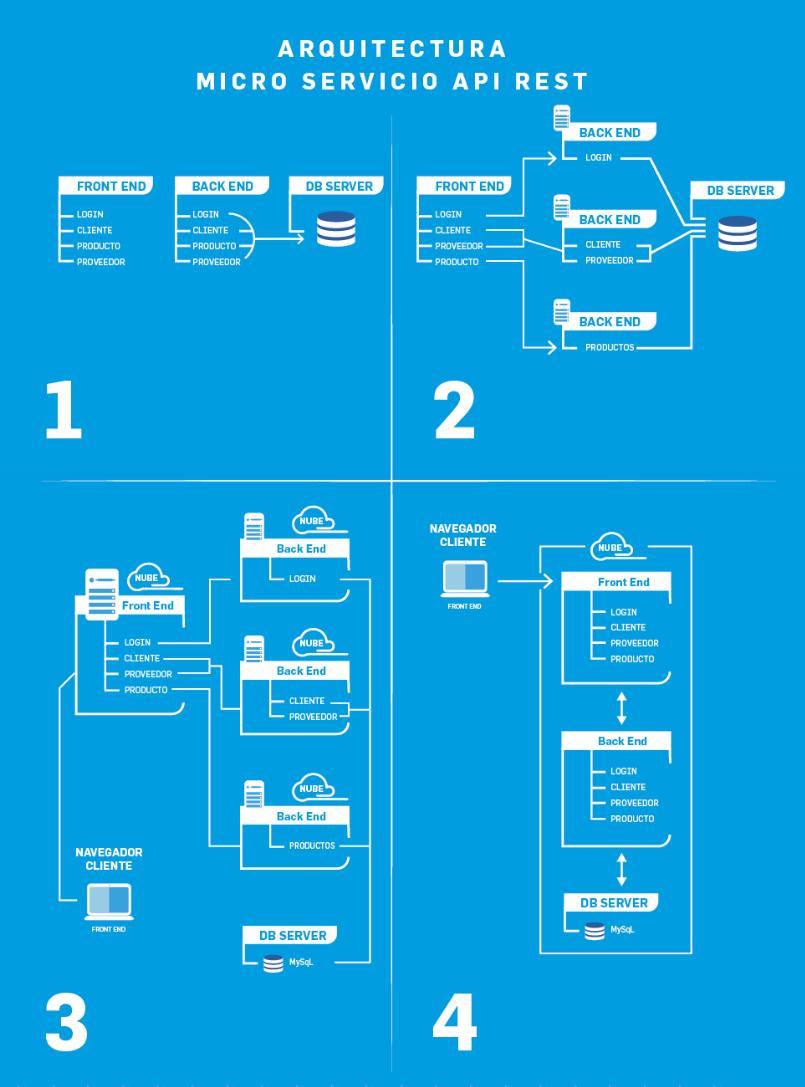


Patrón de capas

Como vemos, la arquitectura web tiene un patrón de diseño en capas (arquitectura distribuida), y cada capa puede estar en un servidor diferente y aun así se pueden interconectar. El objetivo de separar las distintas funcionalidades en distintos servidores es aumentar la escalabilidad y rendimiento. Por ejemplo, el servidor web al ofrecer servicios de http deberá tener una buena conexión a internet mientras que el servidor de base de datos requerirá tener una buena capacidad de almacenamiento y procesamiento.







 Elementos de la arquitectura Web

A continuación, se enumeran los elementos de la arquitectura web (pueden variar según la arquitectura elegida):

**Elementos:**

**La Infraestructura:** si bien es cierto que, en fase de desarrollo, para probar nuestra aplicación web, no necesitaríamos esta infraestructura. Una vez nuestra aplicación se instale en el hosting definitivo, será necesaria una red ethernet y todos los componentes que hacen posible la conectividad de los equipos informáticos. Con esto nos referimos a cables de red, sea utp o fibra óptica, placas de red, sea wifi o cableada, swith, router, modem, etc. Y ESTOS COMPONENTES FISICOS SON NECESARIOS TANTO DEL LADO DEL CLIENTE COMO DEL SERVIDOR. Con esto se quiere decir que no es posible implementar una aplicación web sino existe una infraestructura de red preexistente o se diseña e implementa una nueva. Esta puede ser: internet, intranet o extranet.

**Isp:** es el proveedor del servicio de internet.

**Cliente Web:** es el navegador. Ej Chrome, safari, Firefox, etc. El cliente se ejecuta en un hardware y hoy sabemos que puede ser desde un pc de escritorio, una notebook, o un dispositivo móvil tal como un teléfono celular o una Tablet. Pero ya no se restringe solo a estos dispositivos, sino que podría ser, por ejemplo, un sistema embebido ejecutándose en una SBC (small borad computer). Incluso ejemplo un reloj inteligente, o un dispositivo vinculado a una máquina de producción seriada como los surgidos de la mano concepto de Industria 4.0.

**Nombre de dominio:** dicho de forma sencilla, el nombre de dominio (o, simplemente, “dominio”) es el nombre de un sitio web. Es decir, es lo que aparece después de “@” en una dirección de correo electrónico o después de [www.](http://www.) En una dirección web. Si alguien te pregunta cómo encontrarte en internet, normalmente tendrás que decirle tu nombre de dominio. Las computadoras para comunicarse utilizan direcciones IP (números únicos en la red) un ejemplo de una dirección IP de un servidor es 192.168.0.1. Para nosotros es imposible recordar tantos números y saber qué servicio o que aplicación hay en esa dirección IP o servidor. Para resolver estos problemas se usan palabras que las personas pueden leer, que son intuitivas, fáciles de recordar y dicen mucho sobre el servicio web que ofrecen, se denominan nombres de dominio. ¿Puedo comprar un nombre de dominio? No, los nombres no se pueden comprar, solo se pueden pagar por el derecho a usarlo por cierto periodo de tiempo. Para registrar un dominio a tu nombre debes hacerlo por medio de una empresa que se encarga de administrar las registraciones de nombres de dominio. En el caso de Argentina es <http://nic.ar>

Aquí tienes algunos ejemplos de nombres de dominio: google.com, Wikipedia.org, youtube.com, <https://domains.google/intl/es_es/learn/web-terms-101>

**URL:** una URL (o localizador uniforme de recursos) es una dirección web completa que se utiliza para encontrar una página web específica. Mientras que el dominio es el nombre del sitio web, la URL es una dirección que remite a los usuarios a una de las páginas de este sitio web. Cada URL contiene un nombre de dominio y otros componentes necesarios para localizar una página o un contenido concreto.

**Sitio web:** aunque una cosa lleve a la otra, comprar un nombre de dominio no implica tener un sitio web. EL domino es el nombre del sitio web, la URL es la forma de encontrarlo y el sitio web es lo que los usuarios ven en su pantalla y con lo que interactúan. Es decir, cuando compres un dominio, abras adquirido el nombre de tu sitio web, pero te faltara crear el sitio web en cuestión.

**Servidor DNS:** (sistema de nombre de dominio) Se ocupa de la administración del espacio de nombre de dominio. Este servidor se encarga de hacer las conversiones de nombres de dominio a direcciones IP. Cuando el cliente realiza una petición web, por ejemplo google.com, una de las primeras acciones del sistema es invocar a un servidor DNS para que le devuelva la dirección IP del/ o de algún de los servidores de google. Por ejemplo, devolverá la ip 171.217.162.14.

**Hosting:** es el nombre que se le da al servicio de alojamiento en la web a nuestras páginas, aplicaciones, bases de datos (los hostings son servidores que están siempre encendidos y conectados a internet). Los programadores una vez terminado el trabajo suben su aplicación web al hosting para que todo el mundo pueda acceder

**Servidor web:** o servidor HTTP, es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales o unidireccionales y sincrónicas o asincrónicas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente.

Contenedor de aplicaciones web (o servidor de aplicaciones web): modulo que permite la ejecución de aplicaciones web. Por ejemplo, modulo PHP o Python del Servidor Web. Componente ASP o ASPX de IIS. Servidor o Contenedor de aplicaciones Web Java: Tomcat, Webblogib, Webphere, Jboss, Geronimo, etc.

**Servidor de bases de datos:** estos son contenedores de bases de datos que permiten organizar y administrar los datos que deben permanecer en un medio de almacenamiento permanente. Resuelven problemas de seguridad, mecanismos de comunicación, concurrencia, inconsistencias de los datos, respaldo, entre otros. Hay varios tipos de bases de datos, por ejemplo, las relaciones que organizan los datos en forma de tablas, en filas y columnas. Otros tipos son los orientados a objetos u orientados a documentos donde el concepto de tablas se cambia por el de colección con formatos similares a “json”. JavaScript Object Notation (JSON) es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. Es comúnmente utilizado para transmitir datos en aplicaciones web (por ejemplo: enviar algunos datos desde el servidor al cliente, así estos datos pueden ser mostrados en páginas web, o viceversa).

Ahora que entendemos los principios básicos de la arquitectura WEB y algunos de sus elementos, veamos el modelo OSI.

El modelo OSI o modelo Interconexión de sistemas abiertos (en inglés Open Systems Interconection). Es un modelo de comunicación de 7 capas y es la base por el cual viaja toda la información por las redes.

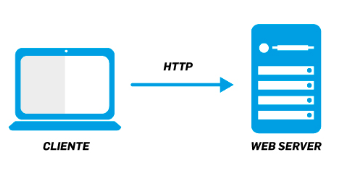


Breve reseña

 A partir del desarrollo de ARPANET en 1969 empieza un crecimiento vertiginoso del uso de la internet. En 1990 [Tim Berners-Lee](https://www.w3.org/People/Berners-Lee/)creó la **WWW**, la "WorldWideWeb" que realizó la primera conexión desde un navegador a un servidor web mientras trabajaba en el CERN desarrollando así, las tres tecnologías fundamentales de la web que son:

* ● **HTML**(lenguaje de marcado de hipertexto). Lenguaje de marcado o etiquetado que se emplea para escribir los documentos o páginas web.
* ● **URL**(localizador de recursos universal). El localizador de recursos uniforme, sistema de localización o direccionamiento de los documentos web.
* ● **HTTP**(Protocolo de transferencia de hipertexto) El lenguaje con el que se comunica el navegador con el servidor web y el que se emplea para transmitir los documentos web.

 Se trata entonces de una arquitectura cliente-servidor en la que cada dispositivo electrónico en la red ( [internet](https://dle.rae.es/internet), [intranet](https://definicion.de/intranet/)o [extranet](https://www.definicionabc.com/tecnologia/extranet.php)) actúa como cliente o servidor lo que implica la comunicación entre procesos que hacen peticiones (clientes) y procesos que responden a esas peticiones (servidores). Esta comunicación es posible gracias al protocolo HTTP.



Arquitectura cliente / servidor básico

En 1994 (1 de octubre) Tim Berners-Lee abandona el CERN y funda la [W3C](https://www.w3.org/), en inglés, "World Wide Web Consortium", organismo internacional que propone recomendaciones y estándares web que aseguran el crecimiento de la World Wide Web.

La evolución puede resumirse en:

* [Web 1.0](https://es.wikipedia.org/wiki/Web_1.0) - Personas conectándose a la Web y la Web como punto de información estática.
* [Web 2.0](https://es.wikipedia.org/wiki/Web_2.0) - Personas conectándose a personas, la inteligencia colectiva como centro de información y la Web es sintáctica.
* [Web 3.0](https://es.wikipedia.org/wiki/Web_3.0) - Aplicaciones Web conectándose a aplicaciones Web, las personas siguen siendo el centro de la información y la Web es semántica.
* [Web 4.0 -](http://www.paradigmatecnologico.com/portfolio/web-4-0/) Personas conectándose con Personas y aplicaciones Web de forma ubicua, se añaden tecnologías como la Inteligencia Artificial, la Voz como vehículo de intercomunicación para formar una Web Total.

. Mencionaremos los hitos más relevantes:

* ● En 1991 surge **HTTP** definido como "protocolo de red para sistemas de información hipermedia distribuidos".
* ● Muy próximo aparece **HTML** 1, es el lenguaje de marcado predominante de las páginas web.
* ● En 1995, Netscape creó **JavaScript**, un lenguaje de secuencias de comandos basado en prototipos y "orientado a objetos". El objetivo de este lenguaje de programación fue darle capacidad de ejecución al cliente de esta arquitectura web, es decir, al navegador.
* ● En 1998 aparecen las hojas de estilo, en su versión 2. Se denominaron **CSS**, del inglés "Cascading Style Sheets", que es un lenguaje de hojas de estilo empleado para describir la semántica de presentación de un documento, en este caso un documento web

 Arquitectura de las aplicaciones web

Las aplicaciones web se basan en una arquitectura cliente / servidor. Es decir que, por un lado, está el cliente (navegador) y por otro lado el servidor. Existen diferentes variantes de la arquitectura básica según como se implementa, pero es importante mencionar que en la tecnología la mayoría de las estructuras están compuestas por capas.

A continuación, enumeramos algunas de las arquitecturas más comunes:

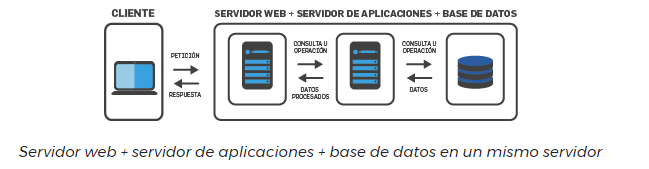
* Servidor web + base de datos en un mismo servidor (2 niveles o capas). En este caso el servidor gestiona tanto la lógica de negocio como la lógica de los datos y los datos.



* Servidor web y de datos separados (3 niveles). En este caso se separa la lógica de negocio a la de datos en diferentes servidores.



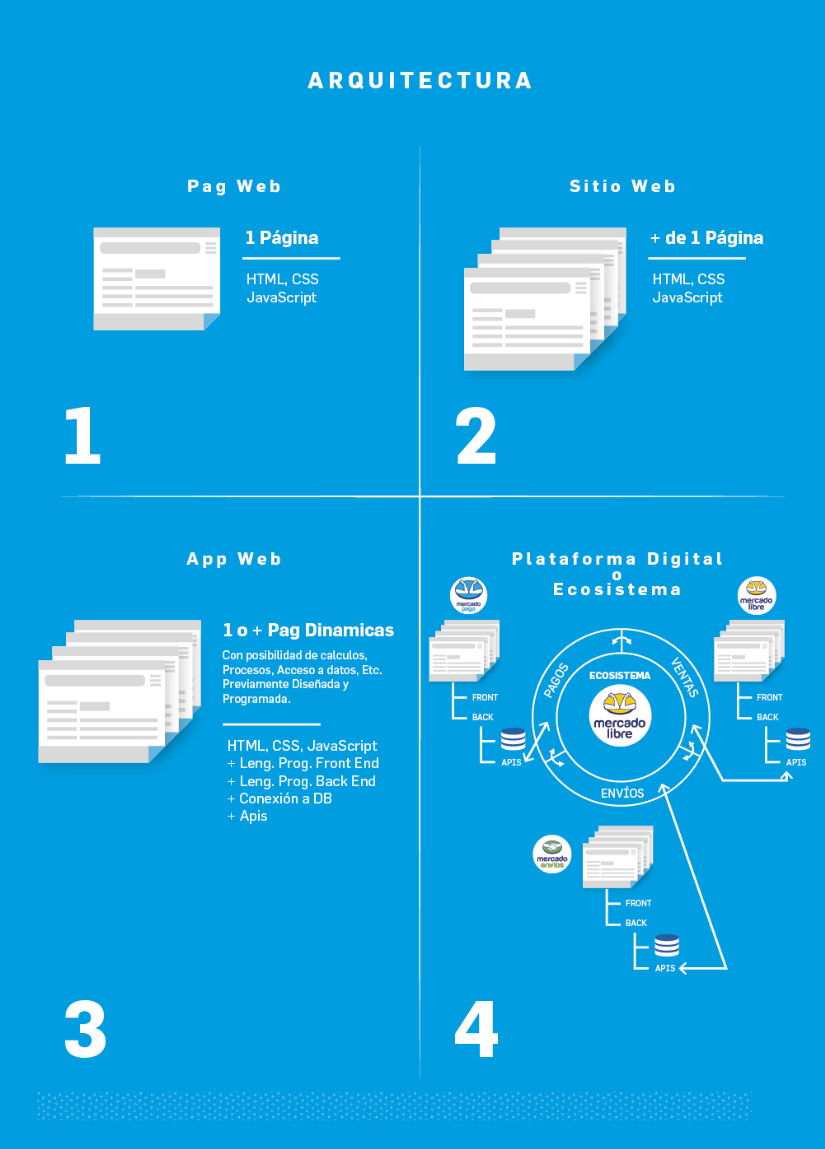
* Servidor web + servidor de aplicaciones + base de datos en un mismo servidor (4 niveles).



Como vemos, la arquitectura web tiene un patrón de diseño en capas (arquitectura distribuida), y cada capa puede estar en un servidor diferente y aun así se pueden interconectar. El objetivo de separar las distintas funcionalidades en distintos servidores es aumentar la escalabilidad y el rendimiento. Por ejemplo, el servidor web al ofrecer servicios de http deberá tener una buena conexión a internet mientras que el servidor de base de datos requiere tener una buena capacidad de almacenamiento y procesamiento.

Un grupo de páginas web dinámicas se conceptualiza como Front End (pensadas para que el cliente acceda) y el otro grupo de páginas dinámicas web como Back End (pensadas para el procesamiento y acceso a datos), además de que la base de datos puede existir en otro Servidor. Esto da lugar a un concepto muy importante en POO el DESACOPLAMIENTO, en este caso el diseño del Front End, Back End y Base de Datos puede desacoplarse.

Ahora veamos distintos escenarios de arquitecturas web, debemos tener presente que nos referimos a como se dividen o distribuyen los distintos componentes de una aplicación en una red, en algunos casos dependiendo su complejidad del desarrollo necesita ser distribuida en varios servidores. La complejidad es determinada por las necesidades de la problemática que se pretende resolver o el diseño de la aplicación. Hagamos un repaso por las distintas arquitecturas que se nos pueden presentar en la siguiente imagen.



Analicemos juntos con más detalles cada escenario planteado.

* **En el escenario 1:**Vemos una página web con contenido estática, es decir, no tiene conexión con ningún servidor, significa que no actualiza su información y solo se puede navegar dentro de la misma página. Para crearla se utilizó HTML, CSS y JavaScript.
* **En el escenario 2:**Vemos una arquitectura centralizada (todo está ubicado en el mismo sitio), contiene varias páginas web con contenido estática, a esto le llamamos sitio web estático, como en el caso anterior tampoco tiene conexión con otro servidor lo que significa que solo se puede navegar entre las mismas páginas. Para crearla se utilizó HTML, CSS y JavaScript.
* **En el escenario 3:**Vemos una Aplicación Web o Web Dinámica, en este caso además de utilizar HTML, CSS y JavaScript se utilizaron lenguajes de programación para poder hacer intercambio de información con las distintas capas de la aplicación, hacer cálculos, crear nueva información en la base de datos, actualizarla o conectar con otros sistemas mediante API. Esta arquitectura tiene la siguiente separación conceptual o de arquitectura:
  + - **Front End:** Es el nombre conceptual que se le da al código programado o la parte de la aplicación web que ve un usuario cuando entra desde el navegador a nuestra aplicación. Por ejemplo, cuando entras a una página como Mercado libre lo que ves en tu navegador es el Front End.
    - **Back End:** Es el nombre conceptual que se le da al código programado o la parte de la aplicación web *que no se ve a simple vista, pero ejecuta acciones que pide el usuario desde el Front End*, es decir, al realizar una búsqueda de un producto determinado dentro de la web de Mercado Libre, el encargado de buscar el producto es el Back End y el encargado de mostrar el producto encontrado al usuario es el Front End.
    - **Conexión a BD:**Es el nombre que se le da a la conexión con la base de datos, allí se guardan todos los productos en el caso de Mercado Libre, de esta manera facilita el almacenamiento y búsquedas de los datos. Más adelante profundizaremos más sobre bases de datos.
    - **Apis:**Por el momento diremos que las Apis nos permiten conectemos con otros sistemas o bien que otros sistemas se conecten con el nuestro, más adelante iremos profundizando más en el tema.

Si no has visto en la graficas estos nombres conceptuales en el escenario 3, te invitamos a que vuelvas a mirarlo para ir asociando el concepto.

* **En el escenario 4:**Vemos una Plataforma Digital o un Ecosistema, en este caso podemos ver que consiste en muchos sistemas que trabajan en conjunto, colaborando para resolver una necesidad o problema. En el ejemplo de Mercado Libre se dedica a vender online, pero también tiene que cobrar y hacer envíos. Por eso han desarrollado aplicaciones independientes pero que saben cómo comunicarse a otros sistemas para pedir o enviar datos para realizar alguna tarea. De esta manera las aplicaciones pueden dar solución integral, comprar, pagar y enviar el producto sin tener que salir de la página. Esta arquitectura es más compleja porque son varios los sistemas que componen el ecosistema, pero si miras con atención el grafico, notaras que existen los mismos elementos que describimos anterior, revisemos:
  + - **Front End**: Parte de la aplicación web que ve un usuario al entrar.
    - **Back End:** Parte de la aplicación web que no se ve y que realiza las acciones en el servidor.
    - **Conexión a BD:** En este caso se representa con un icono de una base de datos.
    - **Apis:**Parte de la aplicación que permite conectarse a otras aplicaciones y que otras aplicaciones se conecten con nuestro sistema.

Arquitectura Web Distribuida

Para comenzar tengamos presente que hemos visto que una aplicación web tiene varios conceptos como arquitectura cliente servidor, Front End, Back End, Base de datos y Apis. Pero qué pasa cuando una aplicación sabemos que va a recibir muchas consultas de nuestra cliente.

Para poder dar respuesta a ello y antes de entender qué es una arquitectura distribuida hagamos el ejercicio de organizar cada concepto en su lugar y veamos cómo podemos distribuir nuestra aplicación.

Supongamos que nos juntamos varias compañeras del curso y tenemos un sistema web o aplicación web con la arquitectura Cliente / Servidor, en ella debemos organizar los conceptos que ya vimos como Front End, Back End, Base de datos y Api. Esta organización quedaría de esta manera:

Arquitectura Cliente / Servidor:

* **Cliente:**
  + Front End: El código o programación que muestra información al navegador web llamado cliente.
* **Servidor:**
  + Back End: El código o programación que ejecutas las acciones en el servidor y conecta con la base de datos.
  + Base de dato o DB: Donde se almacena la información.
  + Api: es quien nos permite conectar a otros sistemas.

#### **¿Qué es una arquitectura centralizada?**

Es cuando tenemos todos los elementos de nuestra aplicación web de arquitectura de Cliente / Servidor en un solo lugar equipo o servidor, es decir, tener el Back End, Front End, Bases de datos y APIs en el mismo equipo. Esto hace que en el caso de una falla del equipo toda nuestra aplicación también fallará.

**¿Qué es la arquitectura distribuida?**

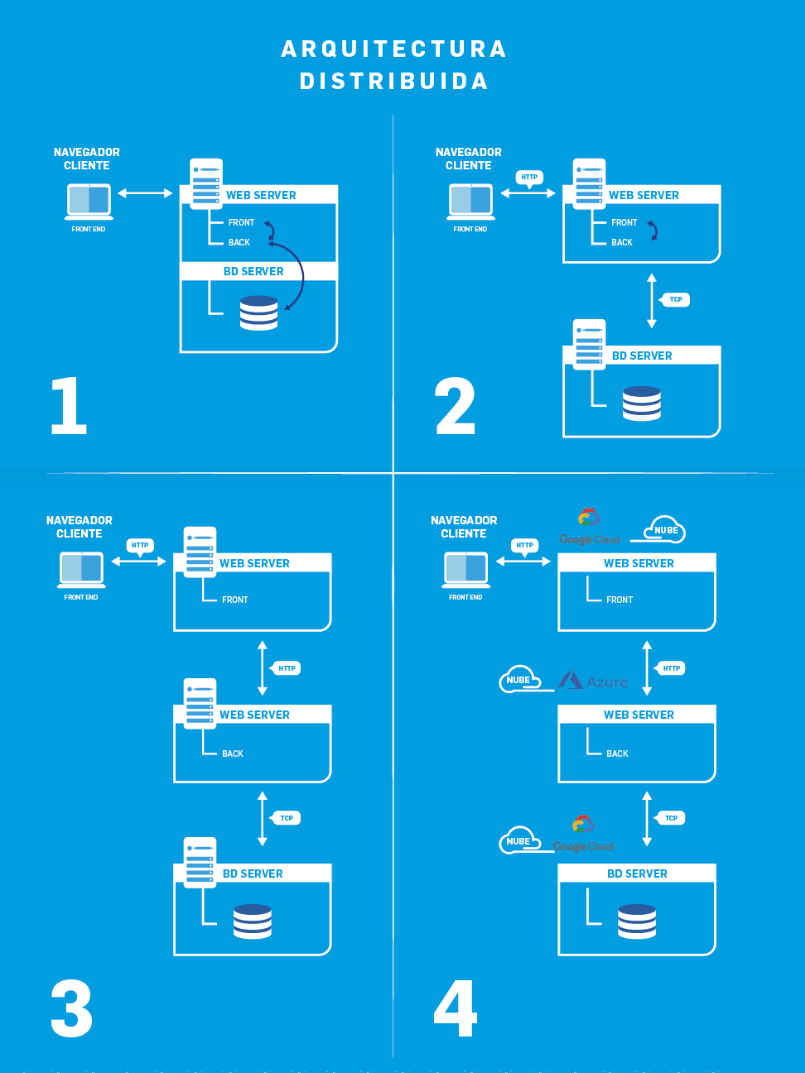
Es tener la posibilidad y capacidad de separar nuestro sistema en distintos servidores de la red (sea red local o internet). Ya sabemos que cuando hablamos de arquitecturas estamos refiriéndonos a una estrategia de cómo construir nuestro sistema dependiendo de lo grande que sea, de las funcionalidades que tenga, esto es mas bien una forma de pensar en cómo escalar nuestro sistema para que soporte más usuarios o más transacciones.

¿Pero cómo te das cuenta cuando una aplicación es distribuida?

En este punto depende de que estés mirando el proyecto o la aplicación, te compartimos solo 2 enfoques:

* **Como usuario:** No te darías cuenta porque si el sistema está distribuido funciona como un conjunto único y sincronizado.
* **Como Programador:** Cuando te asignen a un proyecto o cliente en base a preguntas concretas podrás ir conociendo cómo se implementó o distribuyo el sistema, pero te compartimos algunas formas en las que puedes entender que estás frente a una arquitectura distribuida:
  1. **Por la Líder del Proyecto**: Cuando se comienza a trabajar en un proyecto generalmente la líder del mismo hace una explicación del tipo de aplicación con la que se está trabajando, además de indicarnos en qué parte del proyecto estaremos trabajando.
  2. ***Por el perfil asignado***: Cuando nos asignan el trabajo en una empresa nos especifican si trabajaremos en el Front End, en el Back End o en ambos Full Stack, de esa manera podemos inferir que la arquitectura es distribuida, igualmente siempre es mejor preguntar para estar seguros.
  3. **Por un diagrama:** Generalmente se utilizan diagramas de aplicación para documentar un sistema, en el se puede ver la separación del sistema y si esta distribuido en 1 o varios servidores.

En los siguientes diagramas o esquemas de una aplicación que fue creada y pensada en forma distribuida separando el código en Front End , Back End.  Podemos ver cómo se puede ir escalando o distribuyendo en distintos servidores y que en cualquier caso seguirá funcionando.



Analicemos juntos los escenarios  de la imagen anterior:

* **Escenario 1:**En este caso podemos ver que la aplicación está separada en  Front End, Back End y bases de datos están en el mismo servidor. Notemos que la aplicación fue diseñada de forma modular o separada ( para poder distribuirla) todas las partes del sistema están en un mismo servidor, es decir, en caso de falla del servidor afecta a todo el sistema.
* **Escenario 2:**En este caso podemos ver que se ha separado la base de datos y el sistema sigue funcionando porque el desarrollador Back End escribió el código pensando en una arquitectura distribuida.  Pero la parte del Front End y Back End aun están en un mismo servidor.
* **Escenario 3:**  En este caso podemos ver que cada parte del sistema Front End, Back End y Base de datos esta en un servidor diferente. Con esto comenzamos a ver los beneficios del diseño con arquitectura distribuida en los sistemas.
* **Escenario 4:** En este caso podemos ver que cada parte del sistema está en la nube de distintas empresas y nuestro sistema sigue funcionando por su diseño modular o distribuido.

##### **¿Hasta dónde puedo modularizar o distribuir mi sistema?**

El cómo distribuir el sistema es algo que se analiza en el diseño de la aplicación o se va cambiando a medida que el sistema va creciendo, normalmente cuando llegamos a un trabajo las aplicaciones ya están funcionando y necesitan de nuestro conocimiento para mantenerlas y agregarles mejoras.

Cuando un aplicación se hace más grande, compleja y con más usuarios necesitamos seguir modularizando el sistema, dado que no nos alcanza con separar en Front End, Back End y Base de datos. En esta situación ya debemos pensar en modularizar o separar algunas funcionalidades del sistema, algunos motivos pueden ser:

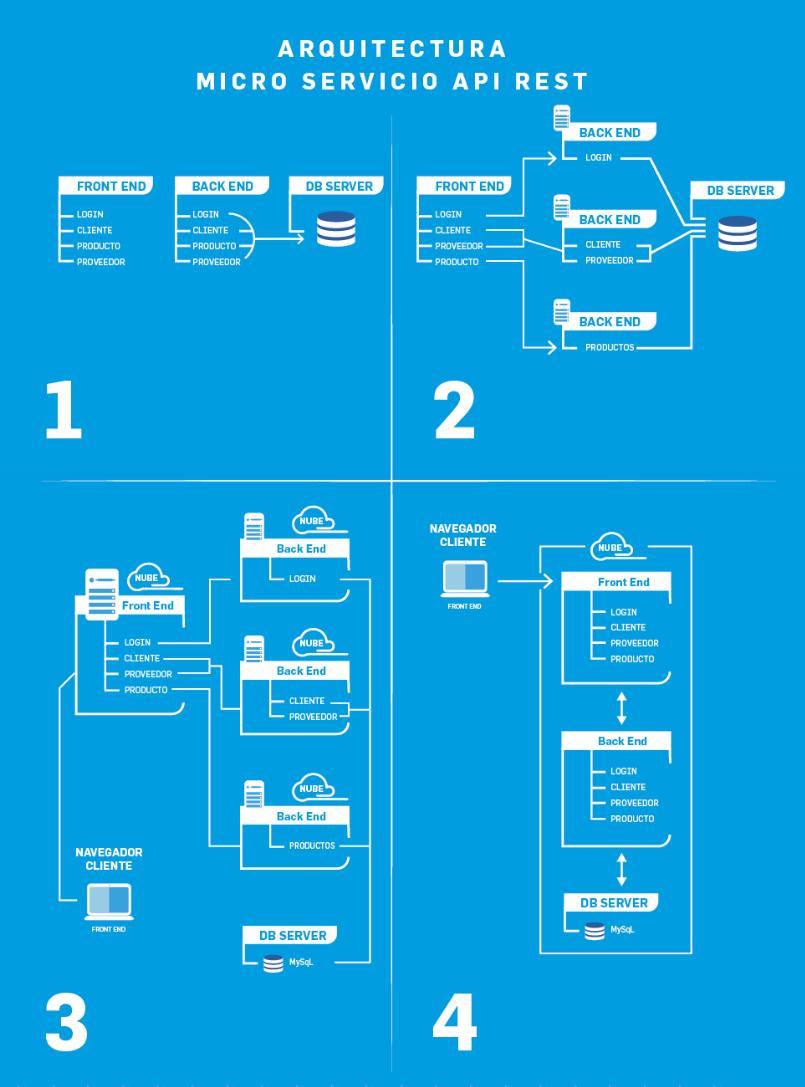
* **Por alta demanda**: Cuando el sistema tiene una funcionalidad que es compleja, consume mucho recurso del servidor o es muy demandada por distintas partes del sistema.
* **Por interconexión**: Cuando un sistema tiene funcionalidades que se necesita dar acceso a otros sistemas para consumir ese proceso, función o datos.
* **Por segregación de roles**: Cuando es necesario separar roles o funciones por motivos de seguridad o  aspectos técnicos, también puede ser porque negocio lo requiere, por ejemplo si se decide por seguridad separar el proceso de autenticación del sistema para reforzar la seguridad.
* **Por escalamiento:**  Cuando las proyecciones indican que en un periodo de tiempo la demanda aumentará considerablemente, será necesario agregar más servidores en la red con la misma funcionalidad para que satisfaga la demanda.

Existen varias **formas de separar estas funcionalidades que llamaremos API REST o Microservicios**, si bien a medida que avancemos iremos aprendiendo más sobre las API REST, ahora veremos cómo la arquitectura distribuida puede aplicarse para pasar de un gran sistema que tiene todo en un solo lugar a separarlo en pequeñas y que todo siga funcionando.

Para ejemplificar tomaremos un sistema que tiene lo siguiente:

* **Front  End:** Todos fue diseñado pensando en una arquitectura distribuida, las funcionalidades son las siguientes.
  + Login: Se conecta con el Back End para validar el usuario y clave.
  + Cliente: Se conecta con el Back End para consultar, editar, crear y eliminar clientes
  + Producto: Se conecta con el Back End para consultar, editar, crear y eliminar productos
  + Proveedor: Se conecta con el Back End para consultar, editar, crear y eliminar proveedores
* **Back End:** Todos fue diseñado pensando en una arquitectura distribuida, las funcionalidades son las siguientes,
  + Login: Recibe la petición, consulta la base de datos y valida si el usuario existe.
  + Cliente: Recibe la petición, consulta la base de datos y devuelve los datos de cliente.
  + Producto: Recibe la petición, consulta la base de datos y devuelve los datos del producto.
  + Proveedor: Recibe la petición, consulta la base de datos y devuelve los datos del proveedor.

Ahora veamos en un diagrama o esquema cómo estas funcionalidades se pueden ir distribuyendo en distintos servidores:



Analicemos juntos los escenarios de la imagen anterior:

* **Escenario 1:**En esta caso tenemos el sistema que está en 3 servidores para cada separación

1. Front End:  Todas las funcionalidades están juntas y en el mismo servidor.
2. Back End: Todas las funcionalidades están juntas y en el mismo servidor.
3. Base de datos: Todos los datos en una base de datos y en el mismo servidor.

* **Escenario 2:**En este caso **podemos ver que cada parte del sistema está distribuido en 5 servidores** y el sistema sigue funcionando porque el desarrollador Back End escribió el código pensando en una arquitectura distribuida.

1. Font End: Todas las *funcionalidades están juntas* y en el mismo servidor.
2. Back End: La funcionalidad de *login está en un servidor*exclusivo.
3. Back End Las funcionalidades de *cliente y proveedor están en un servidor*.
4. Back End: La funcionalidad *producto está en un servidor*exclusivo.
5. Base de datos: Todos los datos en una base de datos y en el mismo servidor.

* **Escenario 3:**  En este caso **podemos ver que cada parte del sistema está distribuido en 5 servidores en nubes diferente**s y el sistema sigue funcionando porque el desarrollador Back End escribió el código pensando en una arquitectura distribuida.
  1. Font End: Todas las *funcionalidades están juntas* y en el mismo servidor de la nube de Argentina.
  2. Back End: La funcionalidad de *login está en un servidor de la nube* de España.
  3. Back End Las funcionalidades de *cliente y proveedor están en un servidor*de la nube de Canadá.
  4. Back End: La funcionalidad *producto está en un servidor en la nube*de Nueva Zelanda.
  5. Base de datos: Todos los datos en una base de datos y en un servidor propio.
* **Escenario 4:** En este caso **podemos ver que el sistema está en la misma nube,**a pesar que las funcionalidades están todas juntas como el en escenario 1 y nuestro sistema sigue funcionando.

1. Font End: Todas las *funcionalidades están juntas*y en el mismo servidor de la nube de Argentina.
2. Back End: Todas las *funcionalidades están juntas* y en la nube de Argentina.
3. Base de datos: Todos los datos en una base de datos y en la nube de Argentina.

Como hemos visto hay varios niveles de como modularizar, distribuir o separar en capas una aplicación con sus funcionalidades, en algunos casos la separación solo puede ser Font End, Back End. Pero existen otras formas de separar en partes más pequeñas la aplicación y eso lo hacemos con la ayuda de las APIs REST.