

Operadores de búsqueda de Google

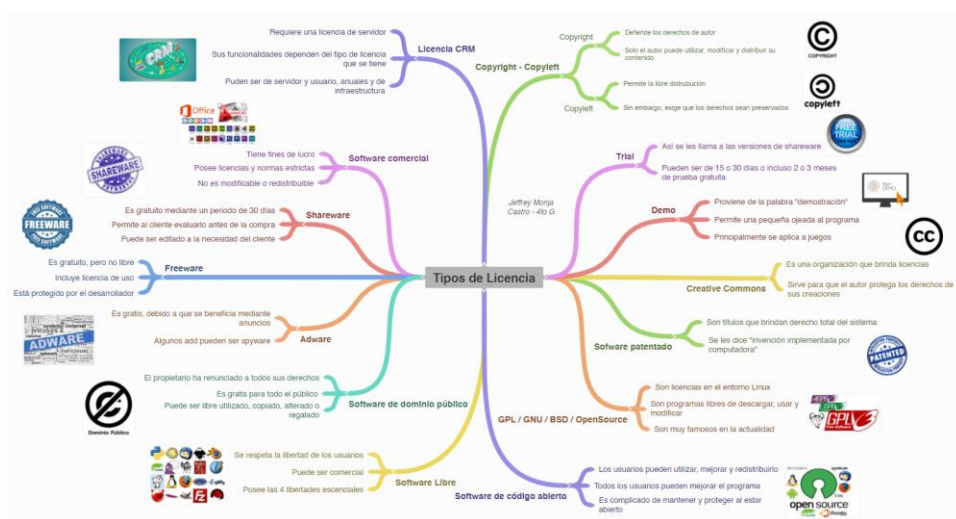
OPERADOR	EJEMPLO	QUÉ HACE
OR	Pelota OR palo OR paso	Te muestra resultados que contengan cualquiera de las palabras que hayas incluido.
AND	JavaScript and Sintaxis	Busca páginas que incluya los dos términos especificados.
" "	"Github flow" o "Subversion"	Te muestra resultados donde aparece el término o los términos exactos que hayas añadido entre los ".
-	Fullstack -MEAN	Te muestra resultados donde se excluya la palabra que hayas puesto detrás del -.
*	"SCRUM *meeting"	Un comodín que puede coincidir con cualquier palabra en la búsqueda.
#..#	Celular 20000..50000 pesos	Te muestra resultados donde se añade un intervalo de números que tú especificas.
()	("redes sociales" OR "plataformas sociales") -Twitter	Te permite combinar operadores. En el ejemplo buscarás redes sociales o plataformas sociales, pero excluyendo Twitter de los resultados.
AROUND	Trucos around(3) Instagram	Resultados donde aparecen las dos palabras especificadas, pero con el número que determines de términos entre ellas.
EN	300 dólares en euros	Sirve para convertir unidades de un mismo tipo de medida.
MAP	map:BuenosAires	La búsqueda te devuelve resultados con mapas del sitio donde le digas.
DEFINE	define:Lunfardo	Busca la definición de una palabra que no conozcas
SITE	Cursos site: www.argentina.gob.ar	Te busca los resultados dentro de una web que hayas especificado.
INFO	info: www.argentina.gob.ar/	Te muestra resultados donde se ofrezca información sobre una página web.

RELATED	related: www.argentina.gob.ar/	Te muestra en los resultados otras páginas similares a la que has escrito.
LINK	Teléfonos link: https://developer.mozilla.org/	Te muestra en los resultados páginas que tienen enlaces a la web que hayas especificado.
CACHÉ	cache: https://developer.mozilla.org/	Te muestra la copia de la página que hay en el caché de Google.
FILETYPE	filetype:pdf presupuestos 2021	Busca resultados que contengan archivos con el formato que hayas especificado
ALLINTEXT O INTEXT	allintext:"desarrollador fullstack"	Encuentra páginas que incluyan en su texto algunos o todos los términos que hayas incluido en el comando.
ALLINTITLE O INTITLE	allintitle:recursos desarrollador web	Te muestra páginas que tengan algunos o todos los términos que hayas incluido en el comando en su título
INURL O ALLINURL	inurl:"apple iphone" allinurl:"apple sfera"	Te muestra páginas con algunos o todos los términos que hayas incluido en el comando
ALLINANCHOR O INANCHOR	allinanchor:"desarrollador fullstack"	Resultados con páginas donde se incluya un enlace con un texto anclado donde se incluya uno o varios términos especificados.
STOCKS	stocks:Facebook	Busca el estado actual de la empresa que busques en bolsa.
WEATHER	weather:Rosario,ar	El tiempo en la ciudad elegida. Mira que después del nombre de la ciudad puedes poner una coma y el país para ser más concreto.
TIME	time:Nueva York	Te muestra la hora en la localidad que decidas.
MOVIE	movie:Avengers	Te muestra resultados relacionados con una película que establezcas.
@	@ArgentinaPrograma	Busca etiquetas sociales asociadas con Twitter.
#	#ArgentinaPrograma	Busca términos publicados con hastags en redes sociales que tengan sistema de hashtags.

Chrome DevTools is a set of web developer tools built directly into the Google Chrome browser. DevTools can help you edit pages on-the-fly and diagnose problems quickly, which ultimately helps you build better websites, faster.

<https://developer.chrome.com/docs/devtools/overview/>

TIPOS DE LICENCIAS:




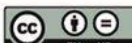
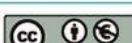
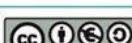
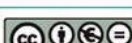


En el caso de las licencias celebradas por otros medios válidos de expresión del consentimiento, cuya aceptación de los términos de la licencia es tácita, se distinguen dos casos:

1. Cuando existen actos positivos por parte de quien adquiere el software.
2. Cuando existe silencio o inactividad por parte de quien adquiere el software.







En algunas doctrinas, se establece que la aceptación tácita debe ser manifestada por hechos inequívocos de ejecución del contrato propuesto.

LAS COMBINACIONES, DE UN VISTAZO

	SI VEO ESTA LICENCIA...	YO PUEDO ...	SI...
MÁS PERMISIVA	 Domínio Público (CC0): Europeana, Figshare, Open Goldberg V.	<ul style="list-style-type: none"> • Compartir • Copiar • Remezclar • Ganar dinero 	<ul style="list-style-type: none"> • Menciono al autor (en algunas jurisdicciones)
	 Reconocimiento (by): PLOS, Saylor.org.	<ul style="list-style-type: none"> • Compartir • Remezclar • Ganar dinero 	<ul style="list-style-type: none"> • Menciono al autor
	 Reconocimiento – CompartirIgual (by-sa): Wikipedia, Wikimedia, Arduino, P2PU	<ul style="list-style-type: none"> • Compartir • Remezclar • Ganar dinero 	<ul style="list-style-type: none"> • Menciono al autor • Mantengo la misma licencia (by-sa)
	 Reconocimiento – SinObraDerivada (by-nd): Drupal, Behance, GNU, Free Software Foundation.	<ul style="list-style-type: none"> • Compartir • Ganar dinero 	<ul style="list-style-type: none"> • Menciono al autor • No hago remezclas
	 Reconocimiento – NoComercial (by-nc): Brooklyn Museum, Wired.com Photography	<ul style="list-style-type: none"> • Compartir • Remezclar 	<ul style="list-style-type: none"> • Menciono al autor • No gano dinero
	 Reconocimiento – NoComercial – CompartirIgual (by-nc-sa): MIT Open CourseWare	<ul style="list-style-type: none"> • Compartir • Remezclar 	<ul style="list-style-type: none"> • Menciono al autor • No gano dinero • Mantengo la misma licencia (by-nc-sa)
MÁS RESTRICTIVA	 Reconocimiento – NoComercial – SinObraDerivada (by-nc-nd): Videos TED Talks, Propublica	<ul style="list-style-type: none"> • Compartir 	<ul style="list-style-type: none"> • Menciono al autor • No hago remezclas • No gano dinero • Mantengo la misma licencia (by-nc-nd)

Las **licencias Creative Commons (CC)** son una herramienta legal de carácter gratuito que permite a los usuarios (licenciarios) usar obras protegidas por derecho de autor sin solicitar el permiso del autor de la obra.

Existen seis licencias de Creative Commons:⁴

Atribución	(CC BY)	
Atribución-Compartir Igual	(CC BY-SA)	
Atribución-No Derivadas	(CC BY-ND)	
Atribución-No Comercial	(CC BY-NC)	
Atribución-No Comercial-Compartir Igual	(CC BY-NC-SA)	
Atribución-No Comercial-No Derivadas	(CC BY-NC-ND)	

Sitios de proyectos y comunidades:

Todas las tecnologías tienen y cuentan con documentación oficial, para acceder a ella es importante que conozcas el origen de ella. Aquí te damos algunos ejemplos con las tecnologías que trabajarás en este curso y que te ayudarán durante toda tu vida como programador:

- Documentación oficial HTML - <https://developer.mozilla.org/es/docs/Web/HTML>
- Documentación oficial CSS - <https://developer.mozilla.org/es/docs/Web/CSS>
- Documentación oficial JavaScript - <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference>
- Documentación oficial Angular - <https://angular.io/docs>
- Documentación oficial Java - [JDK 11 Documentation - Home \(oracle.com\)](https://docs.oracle.com/javase/11/docs/api/)
- Documentación oficial SpringBoot - <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>
- Página Oracle University (Java) - https://education.oracle.com/software/java/pFamily_48

Glosario de recursos útiles

Cursos de Programación (y más):

Udacity: <https://www.udacity.com/>

Coursera: <https://es.coursera.org/>

Canal Youtube MIT: <https://www.youtube.com/c/mitocw>

EdX: <https://www.edx.org/es/>

Herramientas CSS:

CSS Gradient: <https://cssgradient.io/>

CSS Layout: <https://csslayout.io/>

Getwaves: <https://getwaves.io/>

WebGradients: <https://webgradients.com/>

Cubic-bezier: <https://cubic-bezier.com/#.17,.67,.83,.67>

CSS3: <https://www.css3.me/>

CSS-Minifier: <https://cssminifier.com/>

Herramientas BootStrap:

Icons.getbootstrap: <https://icons.getbootstrap.com/>

Themes.getbootstrap: <https://themes.getbootstrap.com/>

Bootswatch: <https://bootswatch.com/>

Getbootstrap: <https://getbootstrap.com/>

Herramientas HTML:

HTML-minifier: <https://html-minifier.com/>

HTML ColorCode: <https://htmlcolorcodes.com/es/>

W3schools: <https://www.w3schools.com/>

Herramientas e Imágenes Alta resolución:

Freepik: <https://www.freepik.es/>

Pexels: <https://www.pexels.com/es-es/>

MotionArray: <https://motionarray.com/>

UnDraw: <https://undraw.co/>

UnSplash: <https://unsplash.com/>

FreelImages: <https://www.freeimages.com/es>

Pixabay: <https://pixabay.com/es/>

Thenounproject: <https://thenounproject.com/>

Tinypng: <https://tinypng.com/>

Manypixels: <https://www.manypixels.co/>

Humaans: <https://www.humaaans.com/>

Uigradients: <https://uigradients.com/#Cocoaalce>

Flaticon: <https://www.flaticon.es/>

Boxicons: <https://boxicons.com/>

Herramientas FrontEnd:

Codepen: <https://codepen.io/>

Modernizr: <https://modernizr.com/>

BrowserShots: <https://browsershots.org/>

CDNJS: <https://cdnjs.com/>

Waybackmachine: <https://archive.org/web/>

Dummyimage: <https://dummyimage.com/>

Colorzilla: <https://www.colorzilla.com/>

Caniuse: <https://caniuse.com/>

Spritecow: <http://www.spritecow.com/>

Figma: <https://www.figma.com/>

JavaScript-minifier: <https://www.minifier.org/>

Fuentes:

fonts.google: <https://fonts.google.com/>

Es.lipsum: <https://es.lipsum.com/>

FontAwesome: <https://fontawesome.com/>

Editores de código:

VScode: <https://code.visualstudio.com/>

VCS:

Subversión: <https://subversion.apache.org/>

Mercurial: <https://www.mercurial-scm.org/>

Git: <https://git-scm.com/>

Github: <https://github.com/>

Gitlab: <https://about.gitlab.com/>

Bitbucket: <https://bitbucket.org/>

Programas para VCS:

Github-desktop: <https://desktop.github.com/>

Git-kraken: <https://www.gitkraken.com/>

Sistema de persistencia:

SQLite: <https://www.sqlite.org/index.html>

MariaDb: <https://mariadb.org/>

MongoDb: <https://www.mongodb.com/es>

Oracle: <https://www.oracle.com/ar/index.html>

PostgreSQL: <https://www.postgresql.org/>

Redis: <https://redis.io/>

MySQL: <https://www.mysql.com/>

DynamoDb: <https://aws.amazon.com/es/dynamodb/>

AmazonRelationDataBaseService: <https://aws.amazon.com/es/rds/>

Navegadores:

Chrome: <https://www.google.com/intl/es-419/chrome/>

Brave: <https://brave.com/es/>

Chrome Canary: <https://www.google.com/intl/es-419/chrome/canary/>

Chromium: <https://www.chromium.org/>

Firefox Developer Edition: <https://www.mozilla.org/es-AR/firefox/developer/>

Servicios web:

Rest API

GraphQL API – Serverless Firebase

AWS Amplify

Cloudinary

Desarrollo API:

Postman: <https://www.postman.com/product/graphql-client/>

Rest Client de VSCode: <https://marketplace.visualstudio.com/items?itemName=humao.rest-client>

Servicios cloud:

Azure: <https://azure.microsoft.com/es-es/>

Amazon Web Services: <https://aws.amazon.com/es/>

GoogleCloud: <https://cloud.google.com/>

Docker - para empaquetar: <https://www.docker.com/>

CMS:

Wordpress: <https://wordpress.com/es/>

Woocommerce: <https://woocommerce.com/>

Magento: <https://magento.com/>

Blogger: <https://www.blogger.com/about/>

Ghost: <https://ghost.org/>

Framework de Backend:

GO (Gorilla, Buffalo, goji)

Python (django, flask)

Typescript/JavaScript/Node (Loopback, Nest, Next.js, Nuxts.js)

Servers:

Nginx: <https://www.nginx.com/>

Apache: <https://www.apache.org/>

Windowsiis: <https://www.iis.net/>

Tomcat: <http://tomcat.apache.org/>

Mobiledev:

ionic (angular react vim)

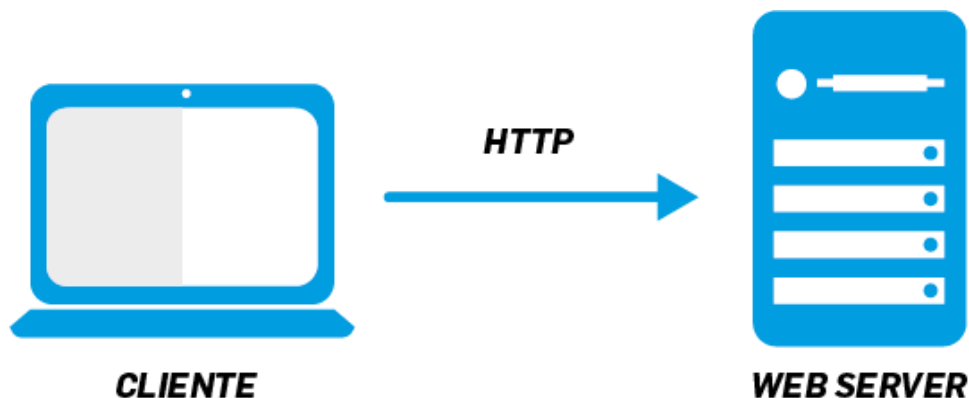
react native + expo, flutter

Breve reseña

A partir del desarrollo de ARPANET en 1969 empieza un crecimiento vertiginoso del uso de la internet. En 1990 [Tim 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](#) , [intranet](#) o [extranet](#)) 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](#), en inglés, "World Wide Web Consortium", organismo internacional que propone recomendaciones y estándares web que aseguran el crecimiento de la World Wide Web.

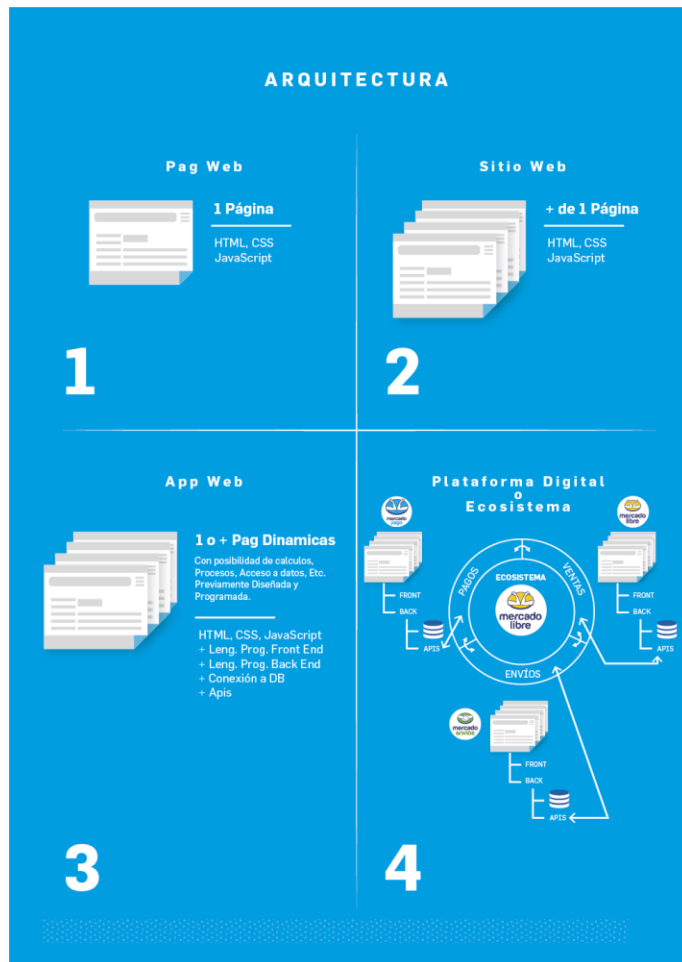
Haciendo [clic aquí](#) podrás ver la evolución de la web. Mencionaremos los hitos más relevantes:

- 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

EVOLUCION DE LA WEB – VIDEOS UTILES:

<https://sites.google.com/site/elrendimientoscolar/evolucion-de-la-web/web-1-0-2-0-3-0-4-0>

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.



- **En el escenario 1:** Vemos una páginas 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 tienen 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 conectarnos con otros sistemas o bien que otros sistemas se conecten con el nuestro, más adelante iremos profundizando más en el tema.
- **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, notarás 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.

Hasta aquí hemos conocido más sobre arquitecturas de aplicaciones web, te invitamos a que apliques tus habilidades de búsqueda para profundizar más del tema.

Arquitectura Web distribuida

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.

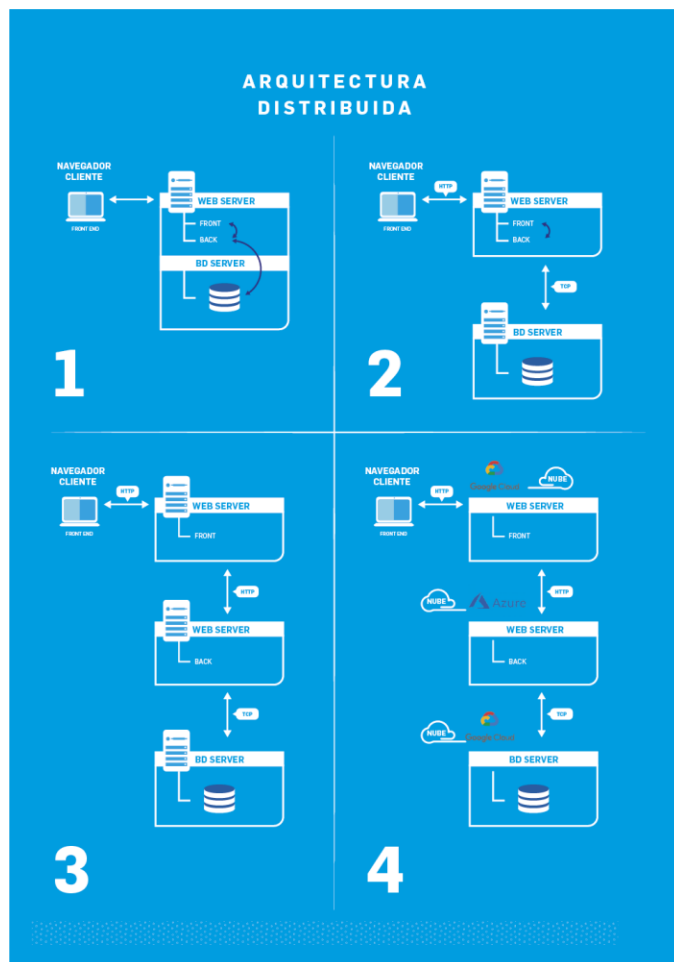
¿Qué es la arquitectura distribuida?

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:
 - - a. **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.
 - b. **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.
 - c. **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.



Analizamos 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?

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:

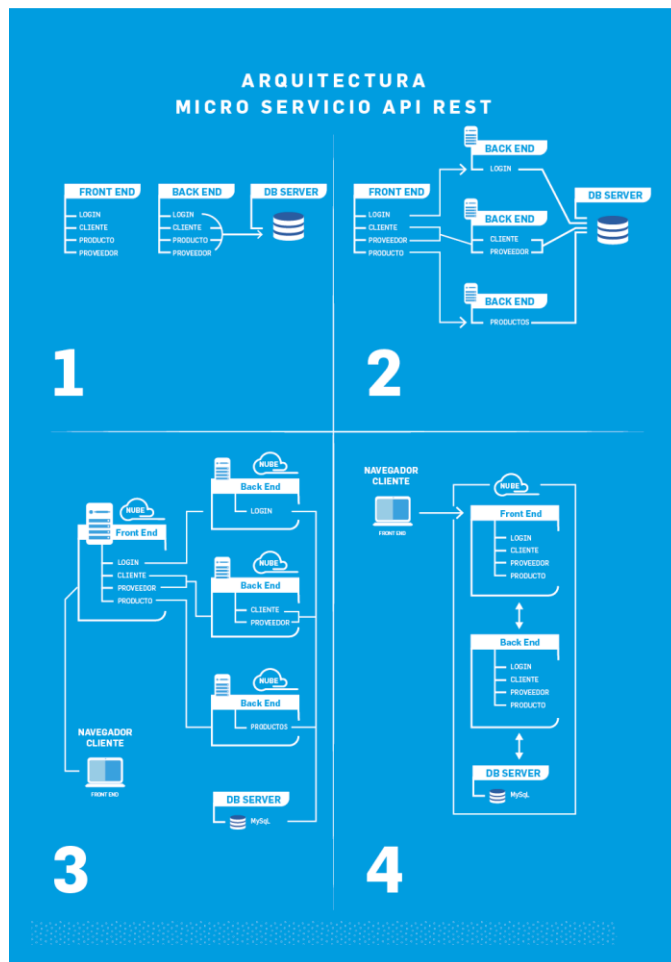
- **Por alta demanda:** Cuando el sistema tiene una funcionalidad que es compleja, consume mucho recurso del servidor o es muy demandada por distintas partes del sistema.
- **Por interconexión:** Cuando un sistema tiene funcionalidades que se necesita dar acceso a otros sistemas para consumir ese proceso, 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:



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 Front End, Back End. Pero existen otras formas de separar en partes mas pequeñas la aplicación y eso lo hacemos con la ayuda de las APIs REST.

Estándares web

Los estándares web son las tecnologías que utilizamos para crear sitios web. Estos estándares existen como extensos documentos técnicos llamados *especificaciones*, que detallan exactamente cómo debería funcionar la tecnología. Estos documentos no son muy útiles para aprender a usar las tecnologías que describen (es por eso que tenemos sitios como *MDN Web Docs*), sino que están pensados para que los utilicen los ingenieros de software para implementar esas tecnologías (generalmente en los navegadores web).

Por ejemplo, el lleno de vida Estándar HTML describe exactamente cómo se debe implementar HTML (todos los elementos HTML y sus APIs asociadas y otras tecnologías circundantes).

Los estándares web son creados por organismos de estándares — instituciones que invitan a grupos de personas de diferentes compañías de tecnología a unirse y acordar cómo deberían funcionar las tecnologías de la mejor manera posible para cumplir con todos sus casos de uso. El W3C es el organismo de estándares web más conocido, pero hay otros como WHATWG (que fueron responsables de la modernización del lenguaje HTML), ECMA (que publica el estándar para ECMAScript, en el que se basa JavaScript), Khronos (que publica tecnologías para gráficos 3D, como WebGL) y otras.

PÁGINA COPADA CON MÁS INFO:

https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/The_web_and_web_standards

¿Pero qué son los Stacks?

Recordemos lo que ya hemos visto en la arquitectura Cliente /Servidor a su vez al código que navega el cliente por el navegador le llamamos Front End y la parte de código que accede a los datos se le llama Back End.

Cada una de estas partes de la arquitectura puede crearse con distintos lenguajes de programación, el que quieras, por citar algunos ejemplo:

Front End: Para el desarrollo del Front End se necesita.

- Estructura y estilos: HTML, CSS o framework como Bootstrap
- Lenguaje programación: JavaScript, java, PHP o bien pueden ser framework como Angular

Back End: Para el desarrollo de Back End se necesita un web server, una base de datos y un lenguaje de programación.

- Base de datos: puede ser MySQL, PostgreSQL, etc
- Lenguaje: Java, PHP, etc
- Web server: Apache, NGINX, etc

Como son muchas las opciones de lenguaje de programación, bases de datos y frameworks. Se ha creado el concepto de Stacks o pila que agrupa las tecnologías (que funcionan bien en conjunto o mas utilizadas) para crear una aplicación desde el Front End y Back End usando ese Stack y ahorrarnos tener que investigar al detalle que tecnología podemos utilizar, es como un combo de tecnologías. Esto también ayuda a las empresas a identificar el conocimiento que tiene un programador al nombrar el Stack tecnológico que utiliza.

Sitios como la red social Facebook, han sido desarrolladas por una combinación de frameworks de codificación y lenguajes, entre los que se incluyen JavaScript, HTML, CSS, PHP y ReactJS. Así podemos decir que este es el "stack tecnológico" de Facebook.

Otro stack muy utilizado es el llamado **MEAN** , que se compone de **MongoDB**, **Express**, **Angular** y **NodeJS**. te dejamos un imagen ilustrativa del Stack MEAN.



Algunos ejemplos de Stacks

- **LAMP**: Linux - Apache - MySQL - PHP. Es la más antigua y el formato de aplicaciones predominante se denomina "**Multi Page Application o MPA**" o aplicación de múltiples páginas.
- **MEAN**: MongoDB - Express - AngularJS - Node.js. Es la más moderna y el formato de aplicaciones predominante se denomina "**Aplicación de una sola página o SPA**" o aplicación de una sola página.
- **.NET**: .NET + WebApi + IIS - SQL Server. Utilizan Microsoft . [Núcleo neto](#) como plataforma de desarrollo.
- **Patrón BFF**: Se puede usar una arquitectura Backend for Frontend (BFF) para crear backends para aplicaciones web o móviles orientados al cliente. Los BFF pueden ayudar a respaldar una aplicación con múltiples clientes al mismo tiempo que mueven el sistema a un estado menos acoplado que un sistema monolítico. El [acoplamiento](#) es "es el grado en que los módulos de un programa **depende** unos de otros". Una aplicación monolítica es una unidad cohesiva de código. Este patrón de código ayuda a los equipos de desarrollo a iterar las funciones más rápido y tener control sobre los backends para aplicaciones móviles sin afectar la experiencia de la aplicación móvil o web correspondiente.
- **Django**: Python - Django - MySQL
- **Ruby on Rails**: Ruby - SQLite - Rails

- **LEMP:** Linux - Nginx - MySQL - PHP
- **MERN:** MongoDB - Express - React - Node.js.

¿Todas las empresas usan si o si estos Stacks?

No siempre, tengamos en cuenta que los stacks son un grupo de herramientas tecnológica sugeridas para crear una aplicación web, por ende, las empresas pueden elegir las herramientas que necesiten sin tener en cuenta los Stack antes nombrados, Pero recuerda cuando una empresa solicita un programador LAMP y te presentan ellos asumirán que sabrás manejar ese grupo de tecnologías.

Existen otras empresas que buscan perfiles con conocimiento solo en Front End o en Back End, estas empresas focalizan al programador en un solo área independientemente del stack.

¿Con qué tecnología trabajaremos en el curso?

Luego de investigar y consultar la demanda requerida de las empresas vs las tecnologías mas utilizadas en Argentina, utilizaremos las siguientes tecnologías:

Para Front End:

- Estructura y estilos: HTML, CSS y framework como Bootstrap
- Lenguaje programación: TypeScript y el framework como Angular ambos basados en Javascript

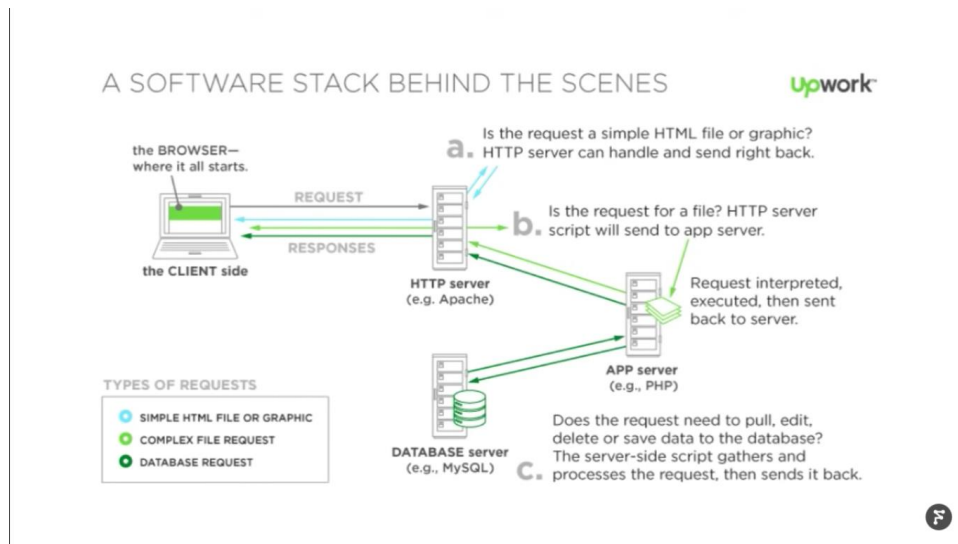
Para Back End:

- Base de datos: Usaremos MySQL
- Lenguaje: Java con Framework Spring Boot
- Web server: Apache Tomcat

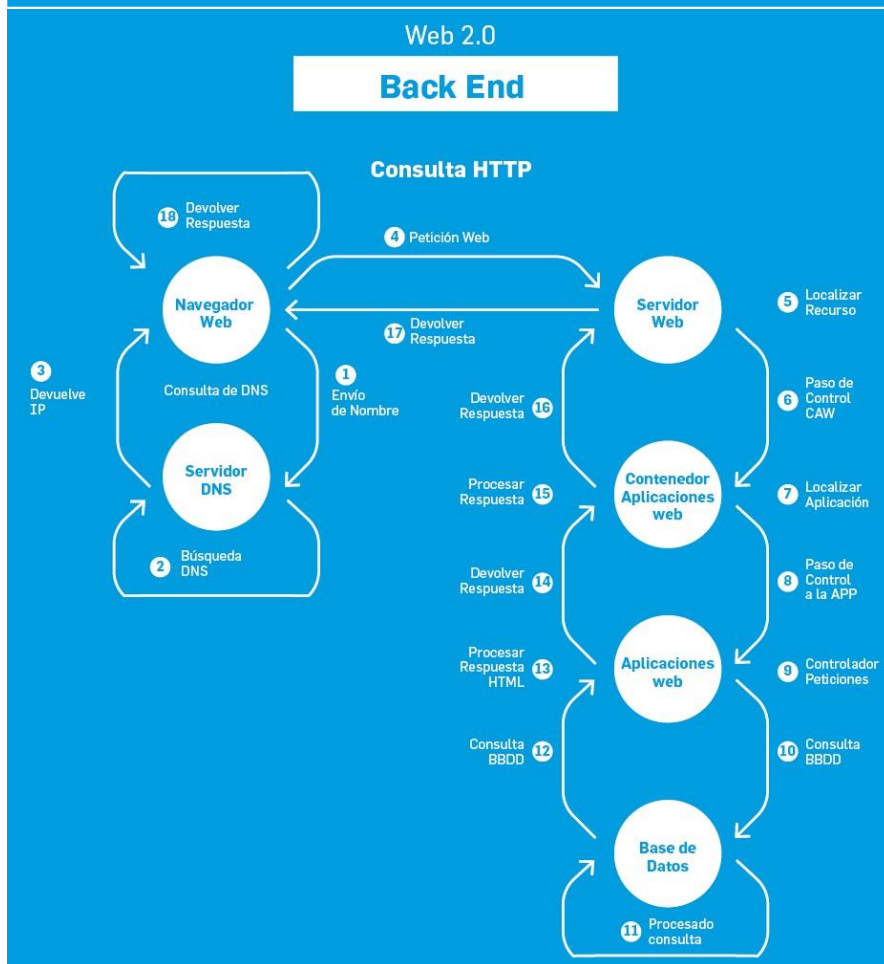
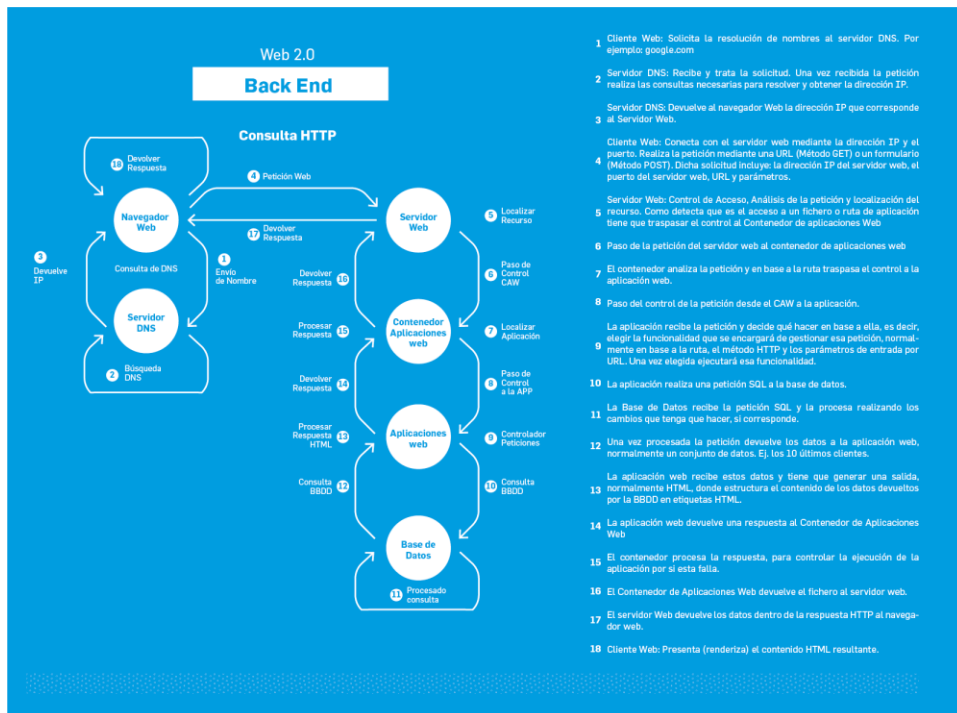
¿Qué significa ser un desarrollador full stack?

El desarrollador full stack es un perfil que tiene conocimiento de lenguajes de programación de Front End, Back End, APIs y Bases de datos. Esta amplitud de conocimiento es muy buscada hoy en día por las empresas, esto significa que al momento de trabajar en un proyecto podrías estar asignado a cualquiera de esas áreas.

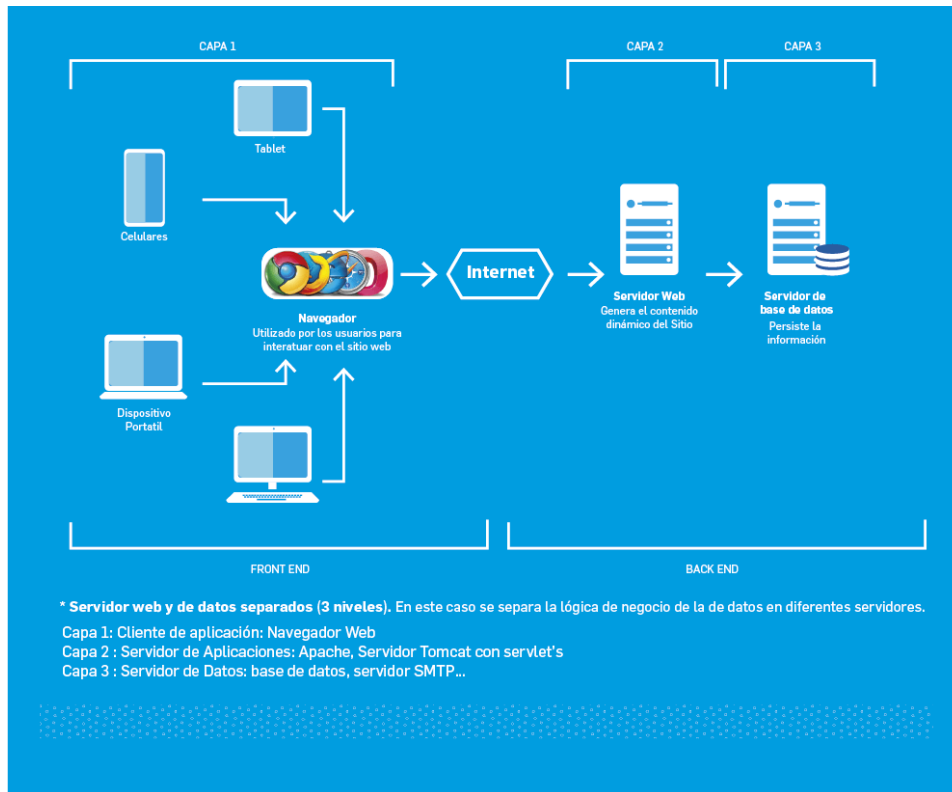
Recuerda que tener el conocimiento de un programador full stack te amplía las posibilidades de trabajo, por ello, vale la pena el esfuerzo de aprender y practicar. En el proceso podrás descubrir que tienes facilidad o te gusta mas alguna parte del Full Stack (Front End, Back End o Bases de datos) y posiblemente te conviertas en experto en alguno. Por eso a seguir estudiando y practicando.



PROCESO DE UNA PETICIÓN WEB:

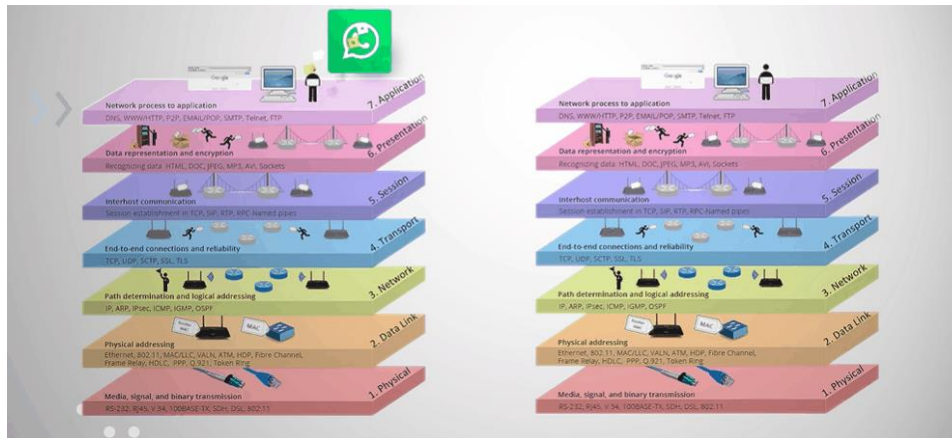


Pero a medida que ampliamos los horizontes aparecen nuevos problemas y estos generan nuevas soluciones. Es que ya no son simples páginas web coloridas y dinamizadas, sino sistemas completos, distribuidos, multiplataformas, para usos generales y específicos, como por ejemplo una plataforma de ecommerce:



En cuanto a los desarrolladores, es importante agregar que si bien, lo más habitual es que se especialicen en frontend o en backend, hoy existe un tercer perfil: desarrollador “Full-Stack” (muy solicitado por las empresas de desarrollo de software). Este perfil se caracteriza por tener una visión integral de toda la aplicación web (frontend + backend).

MODELO OSI



1. Capa Física

Aquí se encuentran los medios materiales para la comunicación como las placas, cables, conectores, es decir los medios mecánicos y eléctricos.

La capa física se ocupa de la transmisión de bits a lo largo de un canal de comunicación, de cuantos microsegundos dura un bit, y que voltaje representa un 1 y cuantos un 0. La misma debe garantizar que un bit que se manda llegue con el mismo valor. Muchos problemas de diseño en la parte física son problema de la ingeniería eléctrica.

2. Capa De Enlace

Se encarga de transformar la línea de transmisión común en una línea sin errores para la capa de red, esto se lleva a cabo dividiendo la entrada de datos en **tramas de asentimiento**, por otro lado se incluye un patrón de bits entre las tramas de datos. Esta capa también se encarga de solucionar los problemas de reenvío, o mensajes duplicados cuando hay destrucción de tramas. Por otro lado es necesario controlar el tráfico.

3. Capa De Red

Se ocupa del control de la operación de la subred. Lo más importante es eliminar los cuellos de botella que se producen al saturarse la red de paquetes enviados, por lo que también es necesario encaminar cada paquete con su destinatario.

Dentro de la capa existe una contabilidad sobre los paquetes enviados a los clientes.

Otro problema a solucionar por esta capa es la interconexión de redes heterogéneas, solucionando problemas de protocolo diferentes, o direcciones desiguales.

Este nivel encamina los paquetes de la fuente al destino final a través de encaminadores (routers) intermedios. Tiene que saber la topología de la subred, evitar la congestión, y manejar saltos cuando la fuente y el destino están en redes distintas.

4. Capa de transporte

La función principal es de aceptar los datos de la capa superior y dividirlos en unidades más pequeñas, para pasarlos a la capa de red, asegurando que todos los segmentos lleguen correctamente, esto debe ser independiente del hardware en el que se encuentre.

El nivel de red es una parte de la subred y los usuarios no tienen ningún control sobre ella. El nivel de transporte permite que los usuarios puedan mejorar el servicio del nivel de red (que puede perder paquetes, puede tener routers que no funcionan a veces, etc.). El nivel de transporte permite que tengamos un servicio más confiable que el nivel de red.

5. Capa de sesión

Permite a los usuarios sesionar entre sí permitiendo acceder a un sistema de tiempo compartido a distancia, o transferir un archivo entre dos máquinas.

Uno de los servicios de esta capa es la del seguimiento de turnos en el tráfico de información, como así también la administración de tareas, sobre todo para los protocolos.

Otra tarea de esta capa es la de sincronización de operaciones con los tiempos de caída en la red.

6. Capa de presentación

Se ocupa de los aspectos de sintaxis y semántica de la información que se transmite, por ejemplo la codificación de datos según un acuerdo.

Esto se debe a que los distintos formatos en que se representa la información que se transmite son distintos en cada máquina. Otro aspecto de esta capa es la compresión de información reduciendo el nº de bits.

7. Capa de aplicación

Contiene una variedad de protocolos que se necesitan frecuentemente, por ejemplo para la cantidad de terminales incompatibles que existen para trabajar con un mismo editor orientado a pantalla. Para esto se manejan terminales virtuales de orden abstracto.

Otra función de esta capa es la de transferencias de archivos cuando los sistemas de archivos de las máquinas son distintos solucionando esa incompatibilidad. Aparte se encarga de sistema de correo electrónico, y otros servicios de propósitos generales.

El nivel de aplicación es siempre el más cercano al usuario.

Por nivel de aplicación se entiende el programa o conjunto de programas que generan una información para que esta viaje por la red.

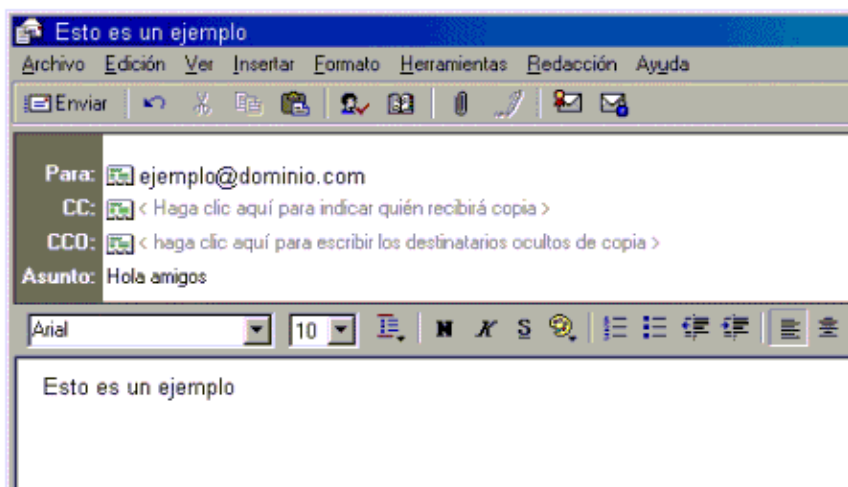
El ejemplo más inmediato sería el del correo electrónico. Cuando procesamos y enviamos un correo electrónico este puede ir en principio a cualquier lugar del mundo, y ser leído en cualquier tipo de ordenador.

Los juegos de caracteres utilizados por el emisor y el receptor pueden ser diferentes por lo que alguien se ha de ocupar de llevar a cabo estos ajustes. También se ha de crear un estándar en lo que la asignación de direcciones de correo se refiere.

De todas estas funciones se encarga el nivel de aplicación. El nivel de aplicación, mediante la definición de protocolos, asegura una estandarización de las aplicaciones de red.

En nuestro ejemplo del correo electrónico esto es lo que sucedería.....

Supongamos que escribimos un mensaje como el siguiente:



En nuestro caso hemos escrito este e-mail en un ordenador PC con Windows98 con el programa de correo Microsoft Outlook. Fuese cual fuese el ordenador, sistema operativo o programa de correo que utilizásemos, lo que finalmente viajaría por la red cuando enviáramos el correo sería algo como esto:

From:"Remitente"	Email	del	remitente
To:			Destinatario
Subject:	Hola		amigos
Date:	Thu,	25	Feb
		2001	09:44:14
MIME-Version:			+0100
Content-Type:			1.0
charset="iso-8859-1"			text/plain;
Content-Transfer-Encoding:			7bit

X-Priority:					3
X-MSMail-Priority:					Normal
X-Mailer:	Microsoft		Outlook	Express	4.72.3110.5
X-MimeOLE:	Produced	By	Microsoft	MimeOLE	V4.72.3110.3

Hola amigos

El estándar que define esta codificación de mensajes es el protocolo SMTP. Cualquier ordenador del mundo que tenga un programa de correo electrónico que cumpla con el estándar SMTP será capaz de sacar por pantalla nuestro mensaje.

TRANSFERENCIA DE DATOS

" (...) una red de computadoras, también llamada red de ordenadores, red de comunicación de datos o red informática, es un conjunto de equipos nodos y software conectados entre sí por medio de dispositivos físicos o inalámbricos que envían y reciben impulsos eléctricos, ondas electromagnéticas o cualquier otro medio para el transporte de datos con la finalidad de compartir información recursos y ofrecer servicios (...) " (Tanenbaum, 2003)



La finalidad principal para la creación de una red de ordenadores es:

- compartir recursos e información a grandes distancias
- asegurar la confiabilidad y la disponibilidad de la información (características propias que definen lo que es información)
- aumentar la velocidad de transmisión de los datos y

- reducir los costos. (CENS, 2018)

La comunicación por medio de una red se lleva a cabo en dos categorías diferentes: una capa denominada física y otra lógica.

La **capa física** incluye todos los elementos de los que hace uso un equipo para comunicarse con otro equipo dentro de la red, como por ejemplo, tarjetas de red, los cables, las antenas, etc.

Con respecto a la **capa lógica** la comunicación se rige por normas muy rudimentarias que por sí mismas resultan de escasa utilidad. Sin embargo, haciendo uso de dichas normas es posible construir los protocolos denominados, que son normas de comunicación más complejas (de alto nivel) capaces de proporcionar servicios útiles.

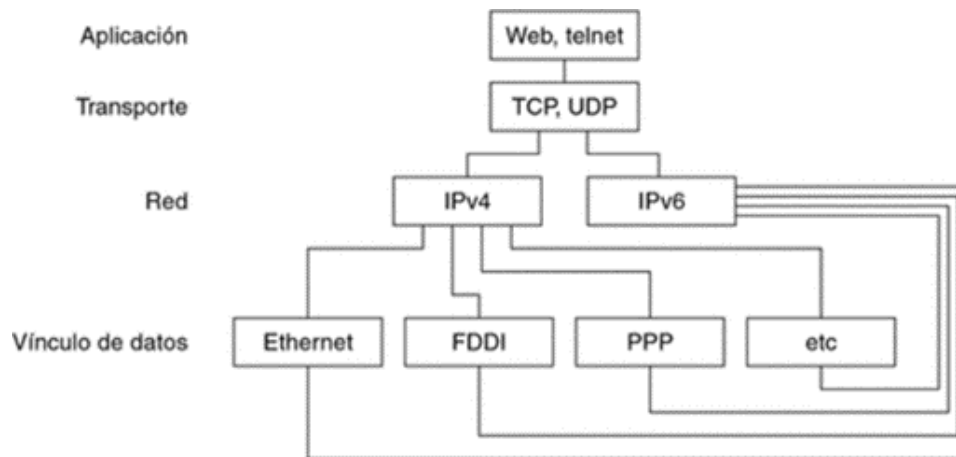
Los protocolos son un concepto muy similar al de los idiomas de las personas. Es decir, si dos personas hablan el mismo idioma, y respetan ciertas reglas (tales como hablar y escucharse por turnos), es posible comunicarse y transmitir ideas e información. Ese es el modelo de un protocolo.

Entre los principales podemos nombrar el conjunto de protocolos TCP / IP que hace referencia a los dos protocolos más importantes que componen la Internet, que fueron los primeros en definirse y qué son los más utilizados.

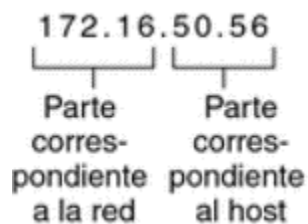
TCP (protocolo de control de transmisión) se usa para crear conexiones entre computadoras a través de las cuales pueden enviarse un flujo de datos. Por la forma en la que está implementado este protocolo los datos serán entregados en su destino sin errores y en el mismo orden en el que se transmitieron. ¿Esto qué quiere decir? que es un protocolo orientado a conexión, ya que el cliente y el servidor deben anunciarse y aceptar la conexión antes de comenzar a transmitir los datos entre ellos. Es decir que hay un intercambio de mensajes entre ellos para abrir una línea de conexión que permanece abierta durante toda la comunicación.

TCP/IP

TCP (protocolo de control de transmisión) se usa para crear conexiones entre computadoras a través de las cuales pueden enviarse un flujo de datos. Por la forma en la que está implementado este protocolo los datos serán entregados en su destino sin errores y en el mismo orden en el que se transmitieron. ¿Esto qué quiere decir? que es un protocolo orientado a conexión, ya que el cliente y el servidor deben anunciarse y aceptar la conexión antes de comenzar a transmitir los datos entre ellos. Es decir que hay un intercambio de mensajes entre ellos para abrir una línea de conexión que permanece abierta durante toda la comunicación.



Por otro lado, el protocolo IP es un protocolo cuya función principal es el uso direccional en origen o destino de comunicación para transmitir datos mediante un protocolo no orientado a conexión que transfiere paquetes conmutados a través de distintas redes previamente enlazadas según la norma OSI.



hay una jerarquía entre capas y el protocolo IP pertenece a una capa denominada de red que está por encima de una capa denominada de transporte en dónde se encuentra TCP.

Entonces, en conclusión, se utiliza la combinación de estos dos protocolos para la comunicación en Internet, en donde TCP aporta la fiabilidad entre la comunicación e IP la comunicación entre distintas computadoras ya que las cabeceras de IP (cabecera por ser una parte del protocolo) contienen las direcciones de destino de las máquinas de origen y llamadas direcciones IP. Estas direcciones serán usadas por los routers para decidir el tramo de red por el que se enviarán los paquetes.

Para entender mejor el funcionamiento de la Internet vamos a decir que **dentro de la red de redes que es Internet debe existir un mecanismo para conectar dos computadoras. Este mecanismo lo proporciona el protocolo de Internet, el cual hace que un paquete de una computadora, llegue a la otra de manera segura a través del protocolo TCP y que llegue a destino a través de las direcciones IP.**

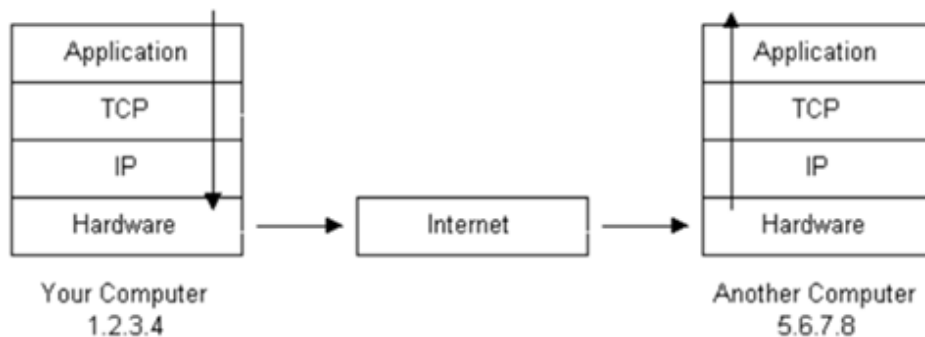
Para terminar, una dirección IP es un número que identifica de manera lógica y jerárquica una interfaz de un dispositivo dentro de una red que utiliza el protocolo de internet.

Todas las computadoras en internet tienen una dirección IP. A modo informativo vamos a decir que existe otro sistema que se denomina sistema de nombres de dominio o DNS que asocian nombres

comunes a direcciones IP por ejemplo la dirección www.cba.gov.ar tiene asociado un número de IP correspondiente, pero este mecanismo existe para que sea más fácil llegar a esa página web sin tener que recordar el número de IP.



Ya vimos que la estructura de red se maneja en capas. También mencionamos que hay una capa de red en dónde está el protocolo IP, una capa Superior de transporte en dónde está el protocolo TCP y ahora vemos una nueva capa que es la de aplicación en dónde se usa el protocolo HTTP.



Una interacción determinada puede darse entre dos sistemas informáticos o involucrar cientos de sistemas. Sin embargo, como sucede al pasar una carta o un paquete de mano en mano, cada transacción se produce entre solo dos equipos cada vez. Para que esto suceda, los dos equipos deben saber, por adelantado, cómo se espera que se comuniquen.

- ¿Cómo inician la conversación?
- ¿A quién le toca comunicarse?
- ¿Cómo sabe un equipo si su mensaje se ha transmitido correctamente?
- ¿Cómo terminan la conversación?

Los equipos lo resuelven mediante protocolos. Un protocolo es un conjunto de reglas convenido. En términos humanos, utilizamos protocolos sociales para saber cómo comportarnos y comunicarnos con otras personas. Las tecnologías tienen su propia forma de establecer reglas de comunicación, como el telégrafo cuando empleaba el código Morse o una radio CB en la que se utilizan códigos como "10-4".

Con los equipos sucede lo mismo, aunque las reglas son más estrictas. Cuando todos los equipos emplean el mismo protocolo, es posible transferir información. Cuando no es así, cunde el caos.

La comunicación era más complicada cuando la gente comenzaba a intercambiar información entre equipos. Cada fabricante tenía un sistema de comunicación propio entre sus máquinas, pero dichos sistemas no permitían la comunicación con los equipos de los demás fabricantes. Pronto quedó claro que era necesario un estándar convenido que permitiera a los equipos de todos los fabricantes comunicarse entre ellos. Ese estándar es TCP/IP.

IP es la parte que obtiene la dirección a la que se envían los datos. **TCP** se encarga de la entrega de los datos una vez hallada dicha dirección IP.

Las cuatro capas del modelo TCP/IP

TCP/IP es un protocolo de enlace de datos que se utiliza en Internet. Su modelo se divide en cuatro capas diferenciadas. Cuando se emplean juntas, es posible referirse a ellas como un paquete de protocolos.

Capa de enlace de datos

La capa de enlace de datos (también denominada capa de enlace, capa de interfaz de red o capa física) es la que maneja las partes físicas del envío y recepción de datos mediante el cable Ethernet, la red inalámbrica, la tarjeta de interfaz de red, el controlador del dispositivo en el equipo, etcétera.

Capa de Internet

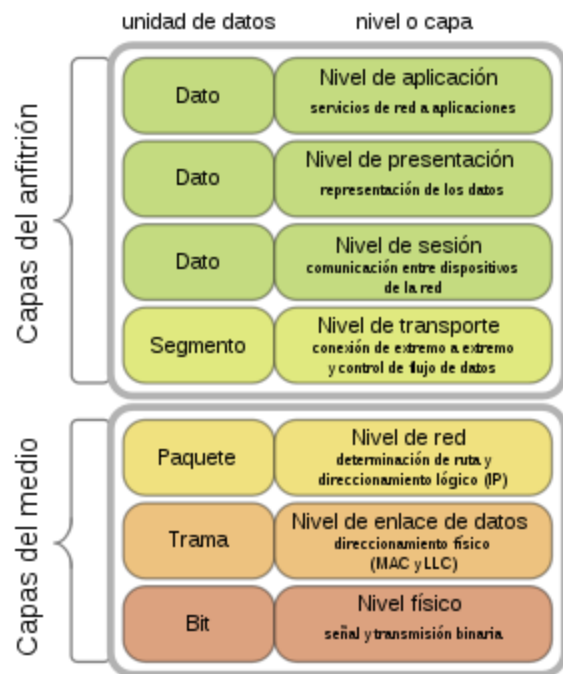
La capa de Internet (también denominada capa de red) controla el movimiento de los paquetes alrededor de la red.

Capa de transporte

La capa de transporte es la que proporciona una conexión de datos fiable entre dos dispositivos. Divide los datos en paquetes, hace acuse de recibo de los paquetes que recibe del otro dispositivo y se asegura de que el otro dispositivo haga acuse de recibo de los paquetes que recibe a su vez.

Capa de aplicaciones

La capa de aplicaciones es el grupo de aplicaciones que requiere comunicación de red. Es con lo que el usuario suele interactuar, como el correo electrónico y la mensajería. Como la capa inferior gestiona los detalles de la comunicación, las aplicaciones no tienen que preocuparse por ello.



HTTP

Para que entendamos la forma de cómo se programa una aplicación web, necesitamos en alguna medida entender los mensajes HTTP. Los mensajes HTTP son en texto plano lo que hace más legible y fácil de depurar. Estos tienen la siguiente estructura:

.- Primero, hay una línea inicial en donde se diferencian dependiendo de si son peticiones y respuestas. Para las solicitudes la línea comienza con una acción requerida por el servidor, a esto se le denomina método de petición seguido de la url del recurso y la versión http que soporte al cliente. Lo importante es el método de petición y la URL (Uniform Resource Locator o localizador de recursos uniforme).

.- Para las respuestas, la línea comienza con la versión de HTTP seguido por un código de respuesta y con una frase asociada a dicho retorno.

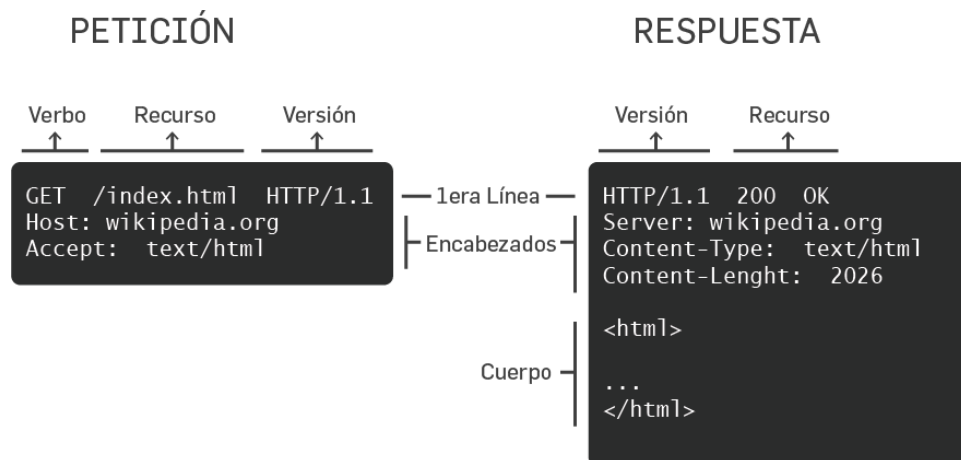
.- También los mensajes tienen una cabecera que son metadatos con información diversa y el cuerpo de mensaje que es opcional. Típicamente este cuerpo tiene los datos que se intercambian entre el cliente y el servidor.

Los métodos de petición (o también llamados verbos) son varios. Cada método indica la acción que desea que se efectúe sobre el recurso identificado lo que este recurso representa dependiente de la aplicación del servidor.

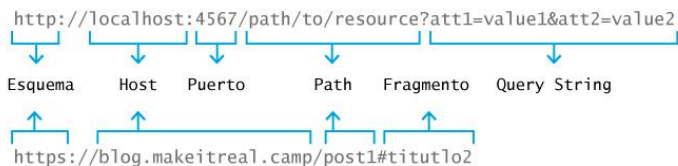
Los métodos que vamos a usar son los de **get** y **post**. Cabe destacar que hay convenciones en donde se utilizan otros métodos. En este punto entran en juego el estándar REST y las API's.

El método get solicita una representación del recurso especificado. Las solicitudes que usan sólo deben recuperar datos y no deben tener ningún otro efecto.

El método post envía los datos para que sean procesados por el recurso identificado. Los datos enviados se incluirán en el de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes.



COMPOSICIÓN DE UNA URL



Method	Description	Sec.
GET	Transfer a current representation of the target resource.	4.3.1
HEAD	Same as GET, but only transfer the status line and header section.	4.3.2
POST	Perform resource-specific processing on the request payload.	4.3.3
PUT	Replace all current representations of the target resource with the request payload.	4.3.4
DELETE	Remove all current representations of the target resource.	4.3.5
CONNECT	Establish a tunnel to the server identified by the target resource.	4.3.6
OPTIONS	Describe the communication options for the target resource.	4.3.7
TRACE	Perform a message loop-back test along the path to the target resource.	4.3.8

Código de estados

Cuando el servidor devuelve una respuesta se indica un código de estado:

- **1xx:** Mensaje informativo.
- **2xx:** Exito
 - 200 OK
 - 201 Created
 - 202 Accepted
 - 204 No Content
- **3xx:** Redirección
 - 300 Multiple Choice
 - 301 Moved Permanently
 - 302 Found
 - 304 Not Modified
- **4xx:** Error del cliente
 - 400 Bad Request
 - 401 Unauthorized
 - 403 Forbidden
 - 404 Not Found
- **5xx:** Error del servidor
 - 500 Internal Server Error
 - 501 Not Implemented
 - 502 Bad Gateway
 - 503 Service Unavailable

Bases de datos: Una base de datos es una colección organizada de información estructurada, o datos, típicamente almacenados electrónicamente en un sistema de computadora (<https://www.oracle.com/mx/database/what-is-database>)

DOM: El *DOM* (**Modelo de Objetos de Documento** en español **Modelo de Objetos del Documento**) es una API definida para representar e interactuar con cualquier documento HTML o XML. El DOM es un modelo de documento que se carga en el navegador web y que representa el documento como un árbol de nodos, en donde cada nodo representa una parte del documento (puede tratarse de un elemento, una cadena de texto o un comentario). (<https://developer.mozilla.org/es/docs/Glossary/DOM>)

- API: Una **Interfaz de Programación de Aplicaciones** (API, por sus siglas en inglés) define un conjunto de directivas que pueden ser usadas para tener una pieza de software funcionando con algunas otras.
- REST: El término "Transferencia de Estado Representacional" (**REST**) representa un conjunto de características de diseño de arquitecturas software que aportan confiabilidad, eficiencia y escalabilidad a los sistemas distribuidos. Un sistema es llamado RESTful cuando se ajusta a estas características. La idea básica de REST es que un recurso, e.j. un documento, es transferido con su estado y sus relaciones (hipertexto) mediante formatos y operaciones estandarizadas bien definidas. Como HTTP, el protocolo estandar de la Web, también transfiere documentos e hipertexto, las APIs HTTP a veces son llamadas APIs RESTful, servicios RESTful, o simplemente servicios REST, aunque no se ajusten del todo a la definición de REST. Los principiantes

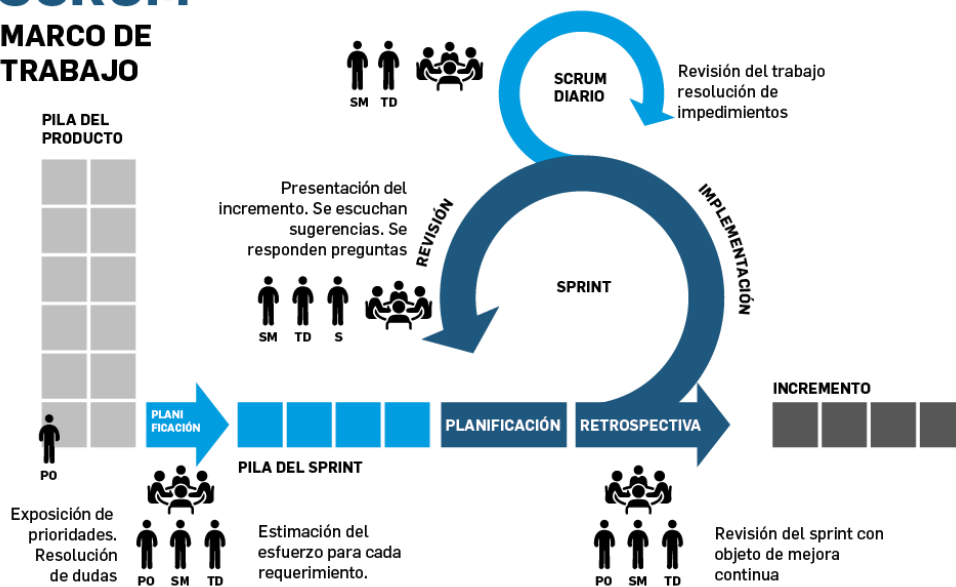
pueden pensar que una API REST es un servicio HTTP que puede ser llamado mediante librerías y herramientas web estándar.

Frameworks: La traducción literal de *framework* es ' **marco de referencia** ', y explica muy bien lo que significa. Un *framework* es un patrón o esquema que ayuda a la programación a estructurar el código y a ahorrar tiempo y esfuerzos a los programadores. (<https://fp.uoc.fje.edu/blog/que-es-un-framework-en-programacion/>)

SCRUM

SCRUM

MARCO DE TRABAJO



ROLES



PO (Product Owner)
Quien determina prioridades.
Una sola persona.



SM (Scrum Master)
Gestiona y facilita el proceso Scrum.



TD (Team Developer)
Quienes construyen el producto.



S (Stackholders)
Interesados, usuarios.

ARTEFACTOS



Product Backlog
Listado de requisitos del producto. Sin mucho detalle. Priorizados. El PO es el responsable y quién decide.



Sprint Backlog
Requisitos comprometidos por el equipo para el sprint. Deben tener suficiente detalle para su ejecución.



Product Increment
Parte del producto desarrollada en el Sprint.

CEREMONIAS



Planning
Jornada de trabajo. El propietario define prioridades y responde dudas. El equipo estima el esfuerzo y se elabora el sprint backlog. Se definen objetivos y tareas.



Daily
15min. de duración. Dirigida por el SM. Se actualiza el sprint backlog.



Review
Informativa. 4 hs aprox. moderada por el SM.



Retrospective
Apunta a la mejora continua.

SPRINT



Ciclo de desarrollo que tiene por objeto un incremento al producto operativo. Tiene una duración de 30 días como máximo.

VALORES

- Valorar a los individuos por encima de los procesos.
- El software que funciona por encima de la documentación exhaustiva.
- La colaboración del cliente por encima de la negociación contractual.
- La respuesta al cambio por encima del seguimiento de un plan.

Según la [IEEE](#) , el software es el “ *conjunto de programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación* ”. Es la parte de un sistema que se puede codificar para ejecutarse en una computadora como un conjunto de instrucciones, e incluye la documentación asociada necesaria para comprender, transformar y usar esa solución. Estos documentos describen la organización del sistema, explican al usuario cómo utilizarlo y obtener eventuales actualizaciones del producto.

El software es un producto intangible, de alto contenido intelectual, que no sufre desgaste alguno y que puede ser potencialmente modificado de forma permanente. No se manufactura sino que se desarrolla a través de proyectos que son realizados por equipos de personas altamente formadas. A pesar de la tendencia a desarrollar componentes que puedan ser reutilizados y adaptados a diferentes necesidades, la gran mayoría del software se construye a medida.

Metodología en Cascada

El modelo de desarrollo en cascada fue presentado por primera vez en 1970 por [Royce](#) . Se lo conoce también como modelo lineal secuencial. Su principal característica es que cada fase del desarrollo está bien definida y separada, siguiendo un orden secuencial en el que cada etapa depende de la finalización de la anterior y que esta última haya pasado un proceso de validación que apruebe el paso a la siguiente.

Comúnmente las etapas del modelo son:

3. Análisis y definición de los requerimientos.
4. Diseño del sistema y del software.
5. Implementación y pruebas unitarias.
6. Integración y pruebas de sistema.
7. Funcionamiento y mantenimiento.

La metodología iterativa e incremental deriva del proceso de desarrollo en cascada pero, con la diferencia de que aquí se admite que las etapas se solapan en tiempo con la finalidad de flexibilizar el tiempo de desarrollo total y así poder alcanzar resultados funcionales de manera temprana.

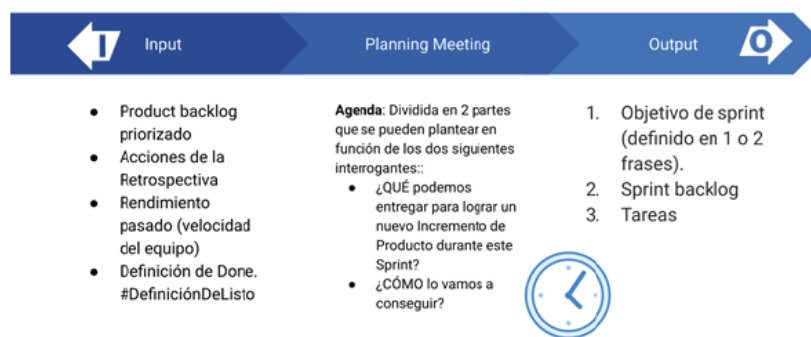
- El proceso **incremental**: con esto se busca desarrollar una parte del producto que se pueda integrar al conjunto a medida que se alcanza un grado de completitud.}
- El proceso **iterativo**: se realiza en ciclos donde se revisa y mejora el producto. De manera que la calidad del producto aumente, y no siempre implica la integración de nuevas funcionalidades.

Si bien hay diferentes métodos ágiles, todos comparten algunos **principios elementales** :

- **Participación del cliente:** Los clientes deben estar implicados en todo el proceso de desarrollo, su rol está enfocado en proporcionar y dar prioridad a nuevos requerimientos del sistema, como así también participar en la evaluación de los entregables de cada iteración.
- **Entrega incremental:** El software se desarrolla en incrementos y el cliente es quien determina qué necesita incluir en cada incremento.
- **Enfocado en personas, no en procesos:** Hay que identificar y explotar las habilidades de cada una de las personas del equipo. Cada equipo debe desarrollar sus propias formas de trabajar, es decir, sin imponer procesos formales.
- **Aceptar el cambio:** Siempre se debe tener en cuenta que los requerimientos del sistema pueden cambiar, y por lo tanto el diseño debe contemplar esto.
- **Mantener la simplicidad:** Se debe enfocar la simplicidad tanto en el software que se desarrolla como en el proceso de desarrollo.

La reunión que se realiza al comienzo de cada Sprint donde participa el equipo completo; sirve para inspeccionar el Product Backlog y que el equipo de desarrollo seleccione del Product Backlog Items en los que va a trabajar durante el siguiente Sprint. Durante esta reunión, el Propietario del producto presenta el Product Backlog actualizado que el equipo de desarrollo se encarga de estimar, además de intentar clarificar aquellos ítems que crea necesarios.

Durante esta etapa se inspeccionan el Product Backlog, los acuerdos de la Retrospectiva, la capacidad y la Definition of Done y se adaptan el Sprint Backlog, Sprint Goal y el plan para poder alcanzar ese Sprint Goal.



Es importante realizar una especificación de los requisitos, es decir documental de forma completa, precisa y verificable los requisitos, el diseño y el comportamiento u otras características de un sistema o componentes del mismo.

Requerimientos Funcionales

Los requisitos funcionales son declaraciones de los servicios que deben proporcionar el sistema. Describen cómo debe reaccionar el sistema a entradas particulares o cómo debe comportarse bajo determinadas condiciones.

Por tanto, la especificación de requerimientos debería cumplir las características de ser completa y consistente. Para que sea completa, todos los servicios descritos por el usuario deben estar definidos. Por otra parte, para que sea consistente los requerimientos no deben ser contradictorios.

Requerimientos no Funcionales

Los requisitos no funcionales son aquellas propiedades que deben tener, como fiabilidad, capacidad de almacenamiento, tiempo de respuesta, etc.

Generalmente surgen de necesidades del usuario que políticas tienen que ver con restricciones de presupuesto, la interoperabilidad con otros sistemas, factores externos, regulaciones, privacidad, seguridad, etc.

Requerimientos en Scrum y Estimación de tiempos

Dentro de las Metodologías Ágiles se suelen utilizar las Historias de Usuario (*User Stories*) como Herramienta para definir los Requerimientos del Sistema. Estas son descripciones o especificaciones de una función, validadas por un usuario o cliente del sistema.

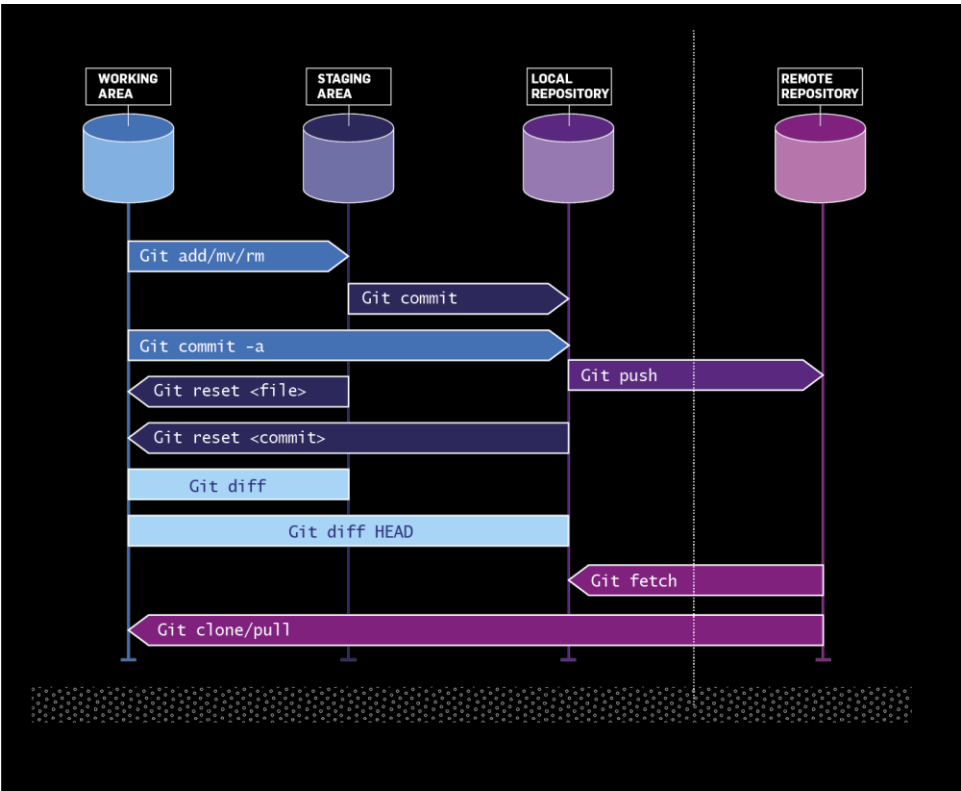
Generalmente las historias se escriben en un lenguaje que el usuario pueda entender y que refleje una descripción sintetizada de lo que este desea. En lo posible se debe tratar de eliminar ambigüedades y malas interpretaciones.

GIT

Para trabajar con git es fundamental entender los estados por los que pueden pasar los archivos durante todo el flujo de desarrollo.

En un proyecto de Git hay 4 secciones fundamentales:

- **Directorio de Trabajo de** (*Working Area*): Es una copia de una versión del Proyecto, Archivos sacados de la BASE DE DATOS comprimida y se colocan en el disco para ser usados o modificados.
- **Área de preparación** (*Staging Area*): Es un archivo que se encuentra dentro del directorio de Git y que contiene información acerca de lo que va a ir en la próxima confirmación.
- **Directorio de Git** (*Local Repository*): Es el lugar en donde se almacenan los metadatos y la base de datos de objetos del proyecto. Es lo que se copia cuando se clona un repositorio desde otra fuente.
- **Repositorio Remoto** (*Remote Repository*) : Es el repositorio que se encuentra en un servidor remoto y con el que eventualmente se sincroniza los trabajos entre los diferentes integrantes del equipo.



ÁREAS Y ESTADOS

Para trabajar con git es fundamental entender los estados por los que pueden pasar los archivos durante todo el flujo de desarrollo.

COMANDOS BÁSICOS

Es un proyecto de código abierto maduro y con un activo mantenimiento desarrollado originalmente por Linus Torvalds.

Este sistema de control de versiones distribuido que funciona bajo cualquier plataforma (Windows, MacOS, Linux, etc) y está integrado en una amplia variedad de entornos de desarrollo (IDEs).

[Ver gráfico](#)

git init

Es el comando para inicializar un directorio como repositorio Git, se ejecuta dentro del directorio del proyecto y, como resultado, crea un subdirectorio .git que contiene todos los archivos para poder realizar el seguimiento de los cambios, etiquetas, etc.

git add <file>

Luego de la creación, modificación o eliminación de un archivo, los cambios quedan únicamente en el área de trabajo, por lo tanto es necesario pasarlos al área de preparación mediante el uso del comando git add, para que sea incluido dentro de la siguiente confirmación (commit).

git status

Es un comando que permite conocer en qué estado se encuentran los archivos

git commit

Con este comando se confirman todos los cambios registrados en el área de preparación, o lo que es lo mismo, se pasan los cambios al repositorio local.

git push

Es el comando que se utiliza para enviar todas las confirmaciones registradas en el repositorio local a un repositorio remoto.

git pull

Funciona al inverso de git push, trayendo todos los cambios al repositorio local, pero también dejándolos disponibles directamente para su modificación o revisión en el área de trabajo. Es importante mencionar que se utiliza cuando ya se tiene un repositorio local vinculado a uno remoto, al igual que con el comando git push.

git clone

En el caso de necesitar "bajar" un repositorio remoto de algún proyecto ya existente se puede ejecutar este comando. Genera un directorio (con el nombre del repositorio o uno especificado explícitamente) que contiene todo lo propio al proyecto, además del subdirectorio .git necesario para poder gestionar los cambios y todo lo pertinente al repositorio Git.

SECCIONES FUNDAMENTALES

DIRECTORIO DE TRABAJO

ÁREA DE PREPARACIÓN

DIRECTORIO DE GIT

REPOSITORIO REMOTO

GIT FLOW

Crear una rama

Cuando esté trabajando en un proyecto, tendrá un montón de características o ideas diferentes en progreso en un momento dado, algunas de las cuales están listas para funcionar y otras no. La ramificación existe para ayudar a administrar este flujo de trabajo.

Agregar confirmaciones

Una vez que se haya creado su rama, es hora de comenzar a hacer cambios. Siempre que agrega, edita o elimina un archivo, está realizando una confirmación (*commit*) y agregándola a su rama. Este proceso de agregar confirmaciones realiza un seguimiento de su progreso a medida que trabaja en una rama de funciones.

Las confirmaciones también crean un historial transparente de su trabajo que otros pueden seguir para comprender lo que ha hecho y por qué. Cada confirmación de un mensaje asociado, que es una descripción que explica por qué se realizó un cambio en particular. Además, cada confirmación se considera una unidad de cambio separada. Esto le permite revertir los cambios si se encuentra un error o si decide ir en una dirección diferente.

Abrir una solicitud de incorporación

Las solicitudes de incorporación (*pull requests*) inician la discusión sobre sus confirmaciones. Debido a que están estrechamente integrados con el repositorio de Git subyacente, cualquiera puede ver exactamente qué cambios se fusionarán si aceptan su solicitud.

Discutir y revisar el código

Una vez que se ha abierto una solicitud de incorporación, la persona o el equipo que revisa sus cambios puede tener preguntas o comentarios. Quizás el estilo de codificación no coincida con las pautas del proyecto, al cambio le faltan pruebas unitarias o tal vez todo se ve bien y los accesorios están en orden. Las solicitudes de incorporación están diseñadas para fomentar y capturar este tipo de discusiones.

Desplegar

Con GitHub, se puede desplegar desde una rama para la prueba final en producción antes de fusionarse con **main**.

Una vez que se haya revisado su solicitud de incorporación y la rama pase las pruebas, puede desplegar sus cambios para verificarlos en producción. Si su rama causa problemas, puede revertirla implementando la rama principal existente en producción.

Fusionar

Ahora que sus cambios se han verificado en producción, es hora de fusionar su código en la rama **main**.

Una vez fusionadas, las solicitudes de incorporación conservan un registro de los cambios históricos en su código. Debido a que se pueden buscar, permite que cualquiera retroceda en el tiempo para comprender por qué y cómo se tomó una decisión.

ACCESS TOKEN GIT

[🔗 GITHUB ACCESS TOKEN - Configuración PASO A PASO en Español 2021 🌐](#) | [Introducción a GIT y GITHUB #8](#)



CANALES DE YT PIOLAS PARA APRENDER COSAS DE PROGRAMACIÓN:

-Todo Code:

<https://www.youtube.com/c/ToDoCode>

-Jorge Ferreiro:

<https://www.youtube.com/channel/UCA2Z3mQUqOj80M84vRu-3AQ>