

Arquitectura Web

Conceptos generales

Víctor Ponz



Institut Educació Secundària

El Caminàs

**Curso de especialización
en Ciberseguridad**

Contenidos

1 Introducción	3
2 Conceptos generales de arquitectura web	3
3 Aplicaciones Web	3
3.1 ¿Qué es la web?	3
3.2 Página web	4
3.3 Sitio web	4
3.4 Aplicación web	4
4 Tecnologías de desarrollo Web	14
4.1 Integración de tecnologías de desarrollo	15
4.2 Sistemas gestores de contenido (CMS)	19
5 Servidores web y de aplicaciones	22
5.1 Servidor Web	23
5.2 Servidor de aplicaciones	24
6 Despliegue de Aplicaciones web	25
Un caso real: Storyblocks	26
1 DNS	28
2 Balanceador de carga (load balancer)	28
3 Servidores de aplicaciones web	29
4 Servidores de bases de datos	30
5 Servicio de caché	31
6 Colas de trabajos y servidores	31
7 Servicio de búsqueda de texto completo	32
8 Servicios	33
9 Datos	34
10 Almacenamiento en la nube	34
11 CDN	34
12 Desde el punto de vista de la seguridad	35

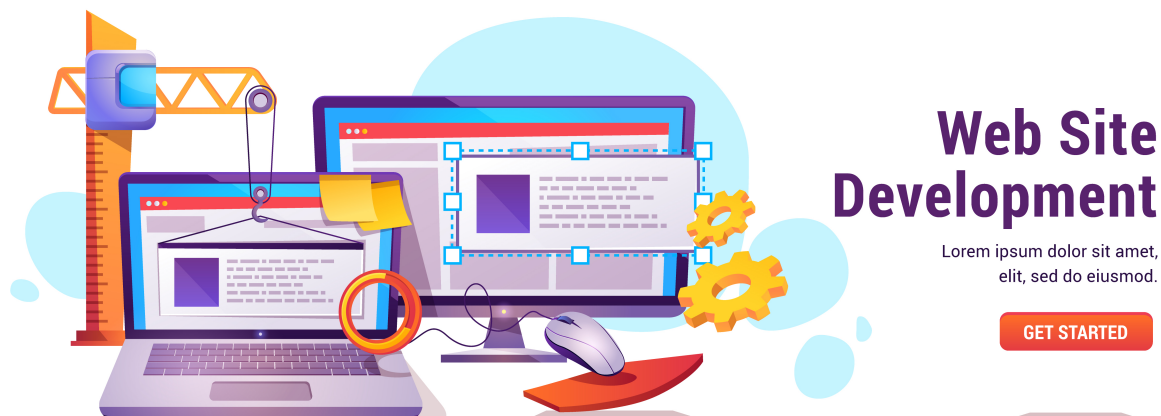


Figura 1: Logo

1 Introducción

En este tema trataremos los conceptos generales de una aplicación web, así como las tendencias actuales relativas a las tecnologías más usadas en el desarrollo de aplicaciones web dinámicas.

2 Conceptos generales de arquitectura web

En esta unidad aprenderemos los siguientes conceptos.

1. Qué es una aplicación web
2. Qué tecnologías se pueden utilizar para crear una aplicación web
3. Qué es un servidor web
4. Y, finalmente, en qué consiste el despliegue de una aplicación Web

3 Aplicaciones Web

3.1 ¿Qué es la web?

Según [Wikipedia](#),

En informática, la World Wide Web (WWW) o red informática mundial es un sistema de distribución de documentos de hipertexto o hipermedios interconectados y accesibles vía Internet. Con un navegador web, un usuario visualiza sitios web compuestos de páginas web que pueden contener textos, imágenes, vídeos u otros contenidos multimedia, y navega a través

de esas páginas usando hiperenlaces.

3.2 Página web

Una página web es un documento electrónico escrito en **HTML** (HyperText Markup Language). Las páginas web están enlazadas a través de hiperenlaces (links). Mediante un navegador un usuario puede navegar a través de la web siguiendo los hiperenlaces

Las páginas web enlazan contenidos de naturaleza heterogénea:

- Imágenes: JPG, GIF, PNG, ...
- Documentos: PDF, TXT, ...
- Audio: MP3, WAV, ...
- Vídeo: AVI, MPEG, ...

3.3 Sitio web

Un sitio web (o portal) es una colección de páginas web relacionadas entre sí que suelen compartir la primera parte de la dirección web (el dominio). Ejemplos:

- <https://www.ieselcaminas.org>: Sitio web del Instituto
- <https://es.wikipedia.org>: Sitio web de la Wikipedia en español

3.4 Aplicación web

Según la [Wikipedia](#),

En la ingeniería de software se denomina **aplicación web** a aquellas herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores web en la que se confía la ejecución al navegador.

Las aplicaciones web son populares debido a lo práctico del navegador web como cliente ligero, a la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales. Existen aplicaciones como los webmails, wikis, weblogs, tiendas en línea y la propia Wikipedia que son ejemplos bastante conocidos de aplicaciones web.

3.4.1 Beneficios de las aplicaciones web

- Las aplicaciones Web **se ejecutan en múltiples plataformas, independientemente del sistema operativo o dispositivo**, siempre y cuando el navegador sea compatible
- Todos los usuarios acceden a la **misma versión**, eliminando cualquier problema de compatibilidad
- **Reducen los costes** tanto para el negocio como para el usuario final, ya que hay menos soporte y mantenimiento requeridos por el negocio y menores requerimientos para el equipo del usuario final
- **No requiere instalar software especial** (en los clientes) En esencia, para acceder a un software web solo necesitamos disponer de un navegador de páginas web (Internet Explorer, Firefox, Opera, Chrome, etc.). No es necesario tener nada más. Debido a la arquitectura de las aplicaciones web, el navegador suele quedar relegado a mostrar la interfaz de usuario (menús, opciones, formularios, etc.), mientras que toda la compleja lógica de negocio se lleva en el lado del servidor.
- **Información centralizada.** En una aplicación web, no solamente la lógica de negocio está centralizada en el servidor, sino también los datos que se ubican en una base de datos centralizada (en ese servidor u otro destinado a tal fin). La centralización tiene la ventaja de facilitar el acceso a la misma.
- **Seguridad y copias de seguridad** . Este es un corolario del punto anterior, es decir, una consecuencia. Como disponemos de los datos centralizados es más fácil establecer y llevar el control de una política de copias de seguridad centralizada.

3.4.2 Procesamiento de páginas en el lado del servidor Desde el punto de vista del **servidor web**, una aplicación Web es un conjunto de páginas Web estáticas y dinámicas. Una página Web estática es aquella que no cambia cuando un usuario la solicita: el servidor Web envía la página al navegador Web solicitante sin modificarla. Por el contrario, el servidor modifica las páginas Web dinámicas antes de enviarlas al navegador solicitante. La naturaleza cambiante de este tipo de página es la que le da el nombre de dinámica.

Por ejemplo, podría diseñar una página para que mostrara los resultados de las notas de un alumno y dejara cierta información fuera (como el nombre del alumno y sus notas) para calcularla cuando la página la solicite un alumno en particular.

3.4.3 Páginas web estáticas en el servidor Un sitio Web estático consta de un conjunto de páginas y de archivos HTML relacionados alojados en un equipo que ejecuta un servidor Web.

Un servidor Web es un software que suministra páginas Web en respuesta a las peticiones de los navegadores Web. La petición de una página se genera cuando el usuario hace clic en un vínculo

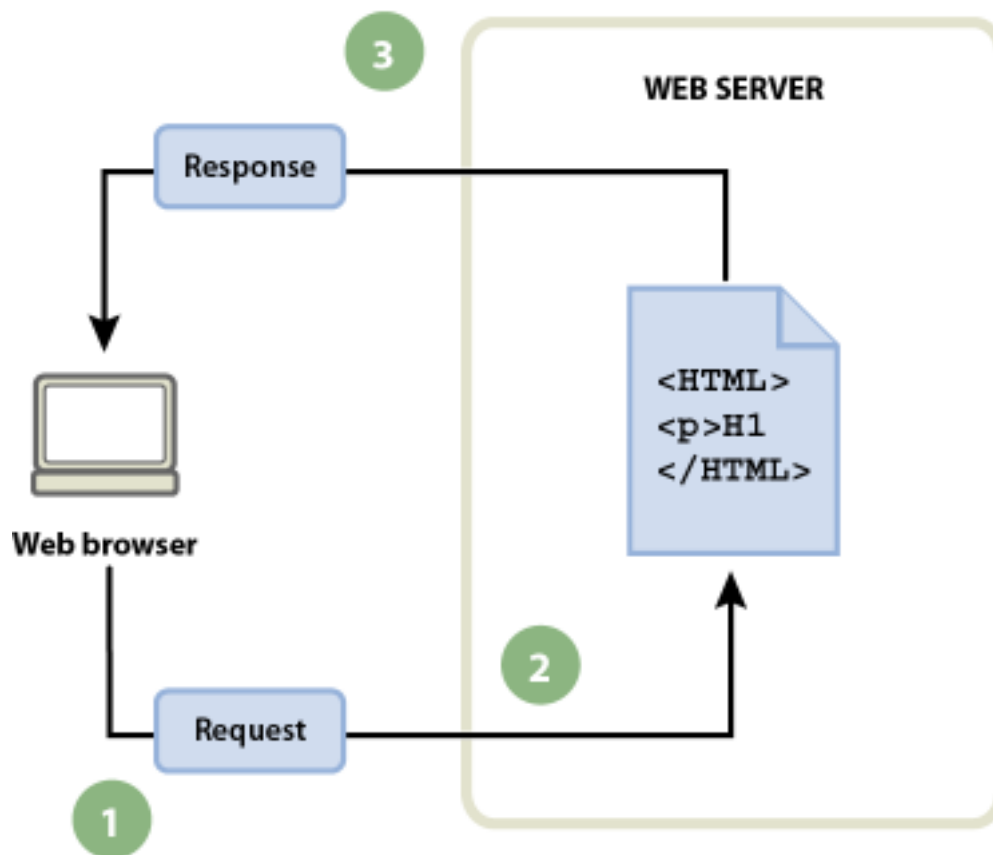
de una página Web, elige un marcador en un navegador o introduce una URL en el cuadro de texto Dirección del navegador.

El contenido final de una página Web estática lo determina el diseñador de la página y no cambia cuando se solicita la página. A continuación se incluye un ejemplo:

```
<html>
<head>
<title>Implantación de arquitecturas Web</title>
</head>
<body>
<h1>Acerca del módulo</h1>
<p>Puesta en producción segura ...</p>
</body>
</html>
```

El diseñador escribe todas y cada una de las líneas de código HTML de la página antes de colocarla en el servidor. El código HTML no cambia una vez colocado en el servidor y por ello, este tipo de páginas se denomina página estática.

Cuando el servidor Web recibe una petición de una página estática, el servidor lee la solicitud, localiza la página y la envía al navegador solicitante, como se muestra en el siguiente ejemplo:

**Figura 2:** Página Estática

1. El navegador web solicita la página estática **2.** El servidor localiza la página **3.** El servidor Web envía la página al navegador solicitante. Finalmente, el navegador renderiza la página mostrando el siguiente resultado.

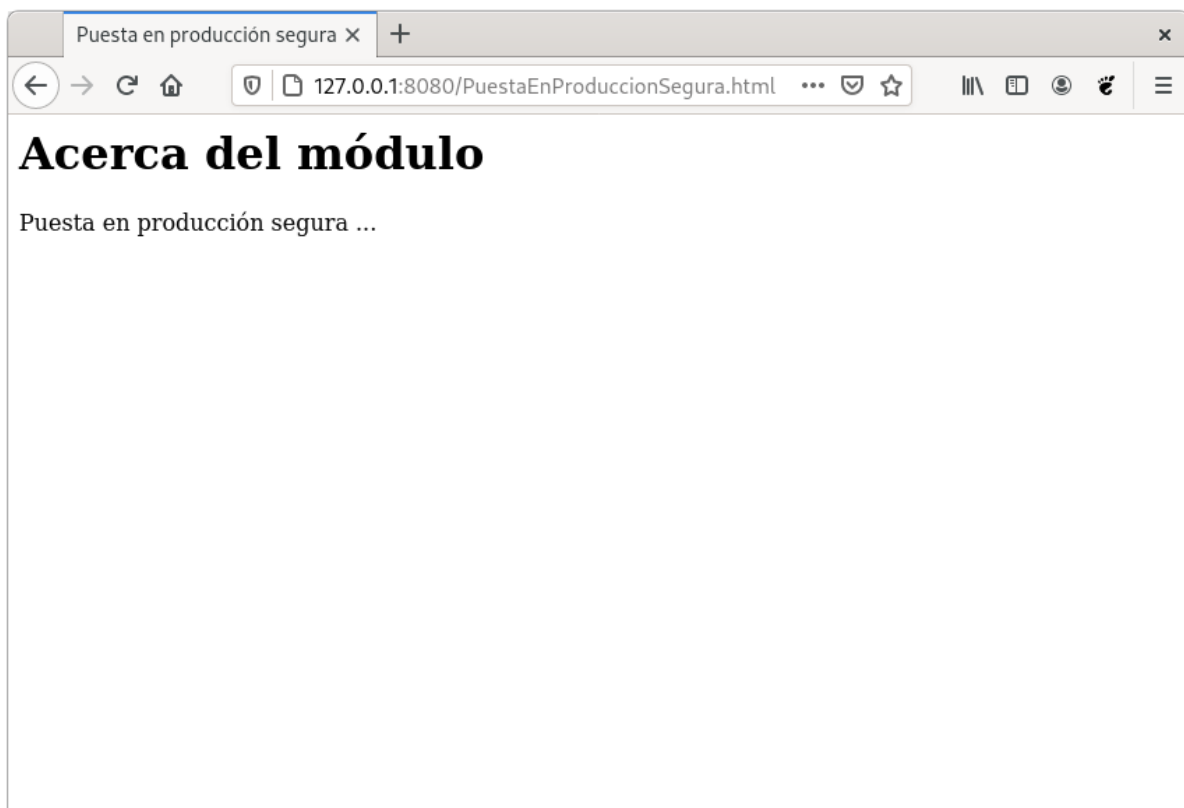


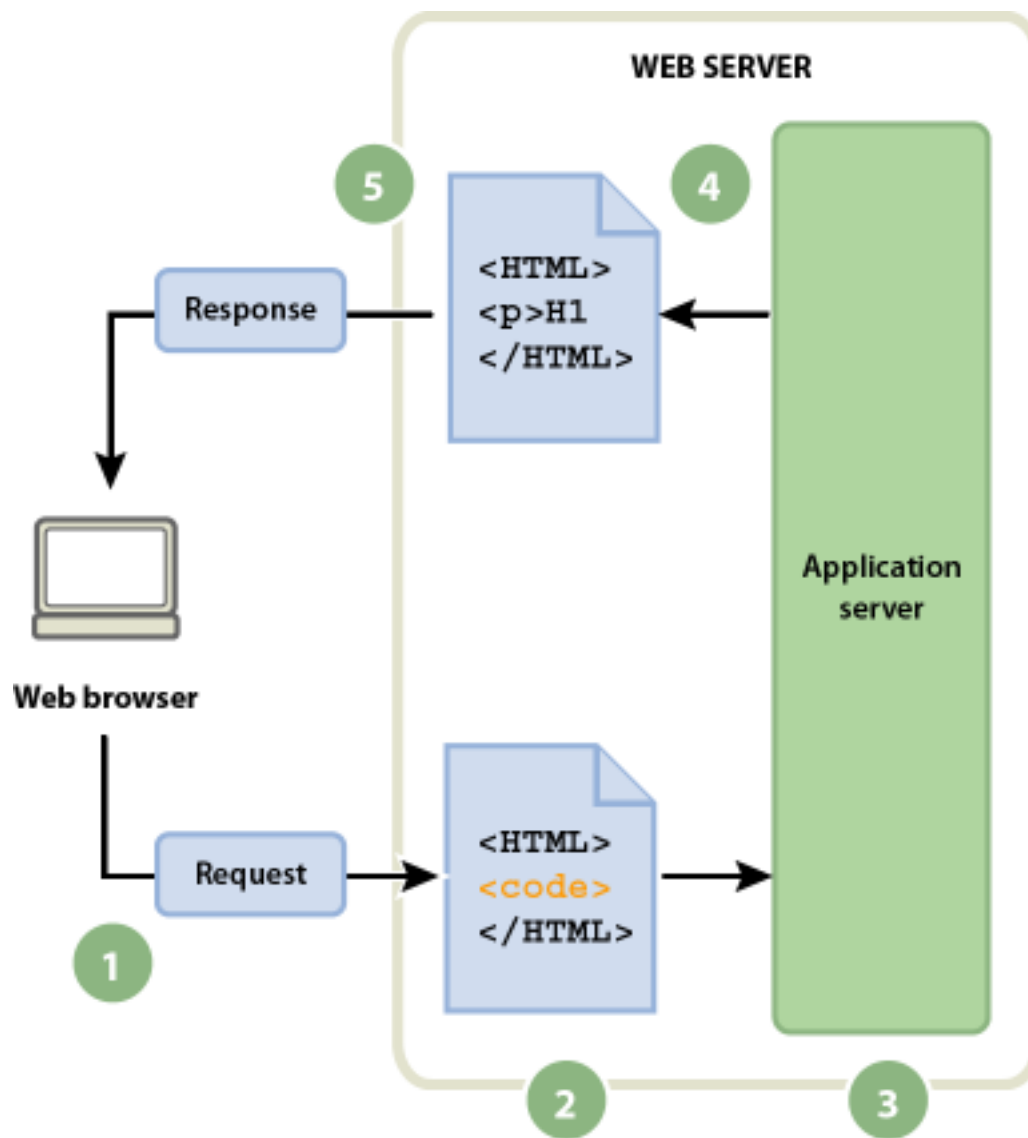
Figura 3: Página estática. Ejemplo

Actualmente esta arquitectura se usa principalmente para:

1. Páginas personales
2. Páginas de proyectos software
3. Documentación técnica ([JavaDoc](#) en Java, Maven site, etc...)

3.4.4 Páginas dinámicas en el servidor Cuando un servidor Web recibe una petición para mostrar una página Web estática, el servidor la envía directamente al navegador que la solicita. Cuando el servidor Web recibe una petición para mostrar una página dinámica, sin embargo, reacciona de distinta forma: transfiere la página a un software especial encargado de finalizar la página. Este software especial se denomina servidor de aplicaciones.

El servidor de aplicaciones lee el código de la página, finaliza la página en función de las instrucciones del código y elimina el código de la página. El resultado es una página estática que el servidor de aplicaciones devuelve al servidor Web, que a su vez la envía al navegador solicitante. Lo único que el navegador recibe cuando llega la página es código HTML puro. A continuación se incluye una vista de este proceso:

**Figura 4:** Página Dinámica

1. El navegador web solicita la página dinámica. **2.** El servidor web localiza la página y la envía al servidor de aplicaciones. **3.** El servidor de aplicaciones busca instrucciones en la página y la termina. **4.** El servidor de aplicaciones pasa la página terminada al servidor web. **5.** El servidor web envía la página finalizada al navegador solicitante que la renderiza.

Un ejemplo sería el siguiente código PHP.

```
<!DOCTYPE html>
<html>
<body>
```

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;
echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
?>
</body>
</html>
```

Y este sería el resultado mostrado en el navegador, donde el servidor de aplicaciones ha interpretado las órdenes PHP

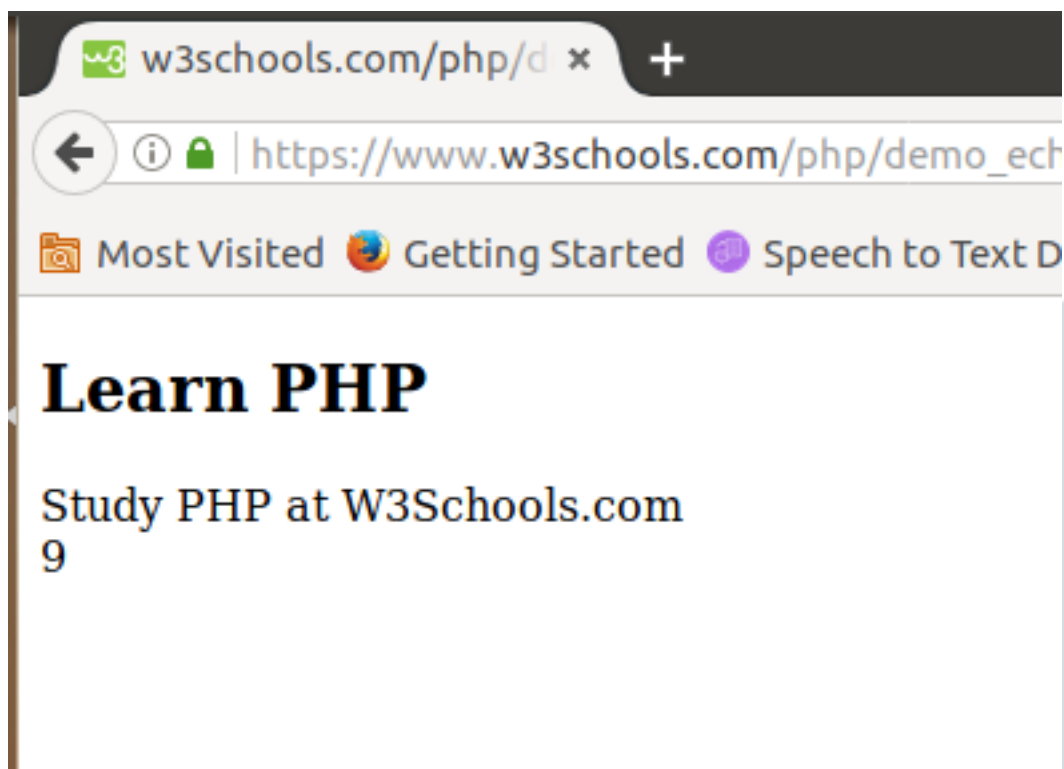


Figura 5: Página Dinámica - Ejemplo

3.4.5 Páginas dinámicas con acceso a una base de datos. Un servidor de aplicaciones nos permite trabajar con recursos del lado del servidor, como las bases de datos. Por ejemplo, una página dinámica puede indicar al servidor de aplicaciones que extraiga datos de una base de datos y

los inserte en el código HTML de la página.

El uso de una base de datos para almacenar contenido permite separar el diseño del sitio Web del contenido que se desea mostrar a los usuarios del sitio. En lugar de escribir archivos HTML individuales para cada página, sólo se necesita escribir una página —o plantilla— para los distintos tipos de información que se desea presentar. Posteriormente, podremos cargar contenido en una base de datos y, seguidamente, hacer que el sitio Web recupere el contenido en respuesta a una solicitud del usuario.

La instrucción para extraer datos de una base de datos recibe el nombre de **consulta de base de datos**. Una consulta consta de criterios de búsqueda expresados en un lenguaje de base de datos denominado **SQL** (*Structured Query Language*, lenguaje de consulta estructurado). La consulta SQL se escribe en los scripts o etiquetas del lado del servidor de la página.

Un servidor de aplicaciones no se puede comunicar directamente con una base de datos porque el formato de esta última impide que se descifren los datos, de una forma bastante similar a cuando un documento de Microsoft Word no puede descifrarse al abrirlo con el Bloc de Notas. El servidor de aplicaciones sólo se puede comunicar con la base de datos a través de un **controlador** que actúe de intermediario con la base de datos: el software actúa entonces como un intérprete entre el servidor de aplicaciones y la base de datos.

Una vez que el controlador establece la comunicación, la consulta se ejecuta en la base de datos y se crea un juego de registros. Un **juego de registros** es un conjunto de datos extraídos de una o varias tablas de una base de datos. El juego de registros se devuelve al servidor de aplicaciones, que emplea los datos para completar la página.

A continuación se ofrece una consulta de base de datos sencilla escrita en SQL:

```
SELECT NOMBRE, APELLIDOS, ASIGNATURA, NOTA  
FROM NOTAS_ALUMNOS
```

Esta instrucción crea un juego de registros de cuatro columnas y lo completa con filas que contienen el nombre, los apellidos, la asignatura y la nota de todos los alumnos de la base de datos.

En el siguiente ejemplo se muestra el proceso de consulta de base de datos y de devolución de los datos al navegador:

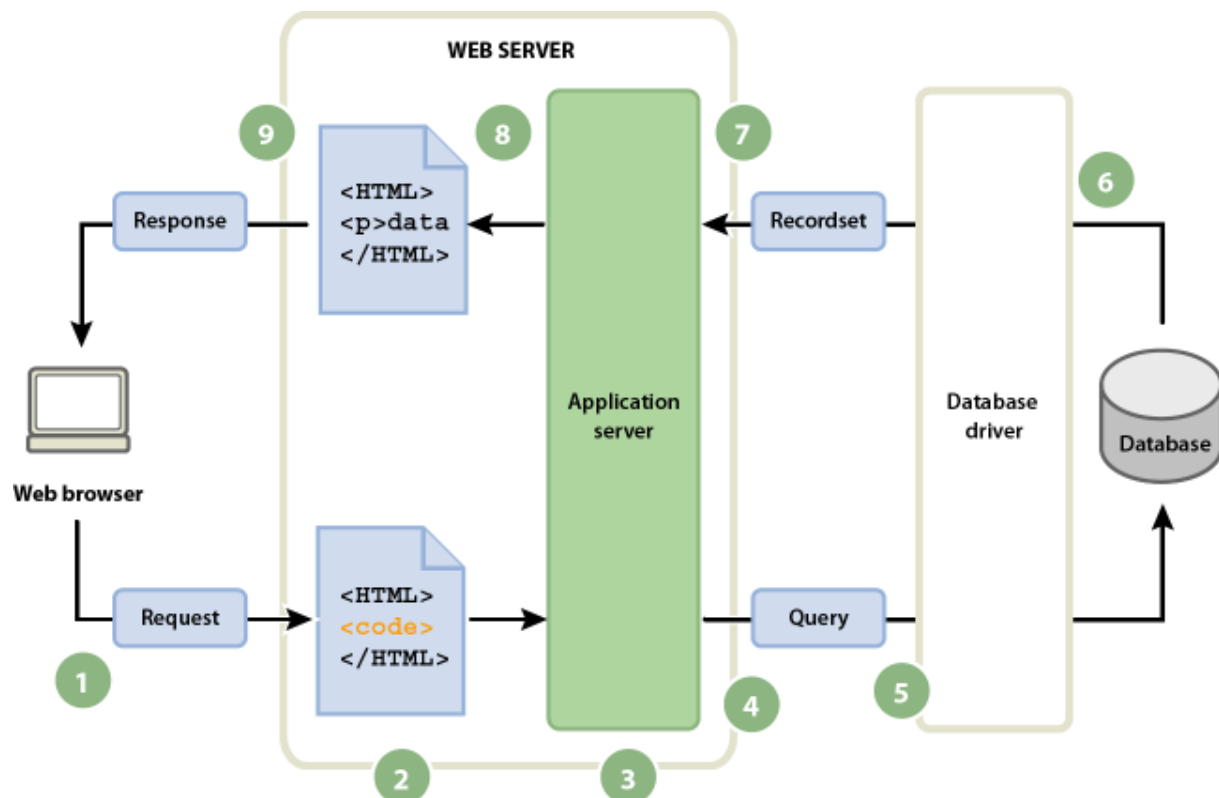


Figura 6: Página Dinámica con acceso a datos

1. El navegador web solicita la página dinámica. 2. El servidor web localiza la página y la envía al servidor de aplicaciones. 3. El servidor de aplicaciones busca instrucciones en la página. 4. El servidor de aplicaciones envía la consulta al controlador de la base de datos. 5. El controlador ejecuta la consulta en la base de datos. 6. El juego de registros se devuelve al controlador. 7. El controlador pasa el juego de registros al servidor de aplicaciones. 8. El servidor de aplicaciones inserta los datos en una página y luego pasa la página al servidor web. 9. El servidor Web envía la página finalizada al navegador solicitante.

3.4.6 Cliente estático o cliente dinámico Desde el punto de vista del **cliente web** (generalmente el navegador) las páginas que componen una aplicación web también se pueden dividir en estáticas y dinámicas.

Las estáticas son aquellas en las que el contenido visualizado por el cliente no cambia, a menos que se realice un clic en un enlace y se cargue una nueva página. Las dinámicas son aquellas en las que el contenido mostrado por el cliente cambia sin necesidad de recargar la página. Estos cambios pueden utilizarse para producir efectos gráficos o realizando peticiones asíncronas de tal forma que el contenido de la página cambie sin necesidad de obligar a una recarga de la misma.

Para crear páginas web de cliente dinámicas se usan técnicas de manipulación del *Document Object Model* (**DOM**) del navegador junto con peticiones asíncronas (**AJAX** - Asynchronous Javascript and XML). Esta técnica llevada al extremo se conoce como aplicación SPA (**Single Page Application**).

Hoy en día, la mayoría de aplicaciones web se construyen con métodos dinámicos tanto desde el punto de vista del servidor como del cliente. Sólo hay que pensar en aplicaciones tipo Gmail, Facebook, Google Maps, Twitter, etc.

3.4.7 Cliente estático creado a partir de texto plano (Static Site Generator) Los generadores de sitios estáticos funcionan convirtiendo texto simple y con formato ligero (generalmente **markdown**) en sitios web o blogs estáticos. No hay bases de datos para ralentizar las solicitudes y el sitio es más fácil de mantener, tiene opciones de seguridad mejoradas y, sin embargo, tiene una infraestructura menor en coste.

Por ejemplo: **Jekyll**, **Hexo**, **Hugo**, **Pelican**, **Middleman**, **Metalsmith**, **Ghost**.

También entra dentro de esta categoría las páginas creadas a partir de **GitHub Pages**.

Los generadores de sitios estáticos (**SSG**) hacen lo mismo que una web dinámica, pero sin las desventajas de ésta. Aplican datos y contenido a las plantillas y generan una vista de una página que se puede mostrar a los visitantes de un sitio.

La mayor diferencia entre un generador de sitios estáticos y una pila de aplicaciones web tradicional es que, en lugar de esperar hasta que se solicite una página y luego generar su vista a petición cada vez, un generador de sitios estáticos hace esto con anticipación para que la vista esté lista para ser servida con anticipación. Y lo hace para cada vista posible de un sitio en el momento de la construcción.

Tienen una ventaja muy importante desde el punto de vista de la seguridad:

Dado que los generadores de sitios estáticos crean un conjunto de activos estáticos que pueden ser servidos desde un servidor web simplificado, o mejor aún, directa y completamente desde una red de distribución de contenido (CDN), tienen un perfil de seguridad notablemente bueno. Dado que se procesan con anticipación y están listos para servir, la infraestructura involucrada en servirlos puede simplificarse enormemente y tener muy pocos vectores de ataque malicioso. Cuando eliminamos la necesidad de que los servidores realicen la lógica y el trabajo, eliminamos las formas en que los malos actores les inyecten código malicioso y los engañen para que realicen acciones nefastas.

Y cuando no necesitamos acceder a bases de datos, realizar operaciones lógicas o modificar recursos para cada vista, podemos simplificar drásticamente nuestra infraestructura de alojamiento. Esto también mejora aún más la seguridad, ya que físicamente hay menos servidores involucrados en la gestión de solicitudes.

No hay servidor más seguro que la ausencia de servidor. Cuanto menos infraestructura necesite nuestra aplicación mejor desde el punto de vista de la seguridad. Sólo debemos instalar y/o activar aquellos recursos necesarios para que nuestra aplicación funcione. Cuantos menos módulos debamos instalar, mejor para la seguridad ya que nos hemos de preocupar de menos vectores de ataque

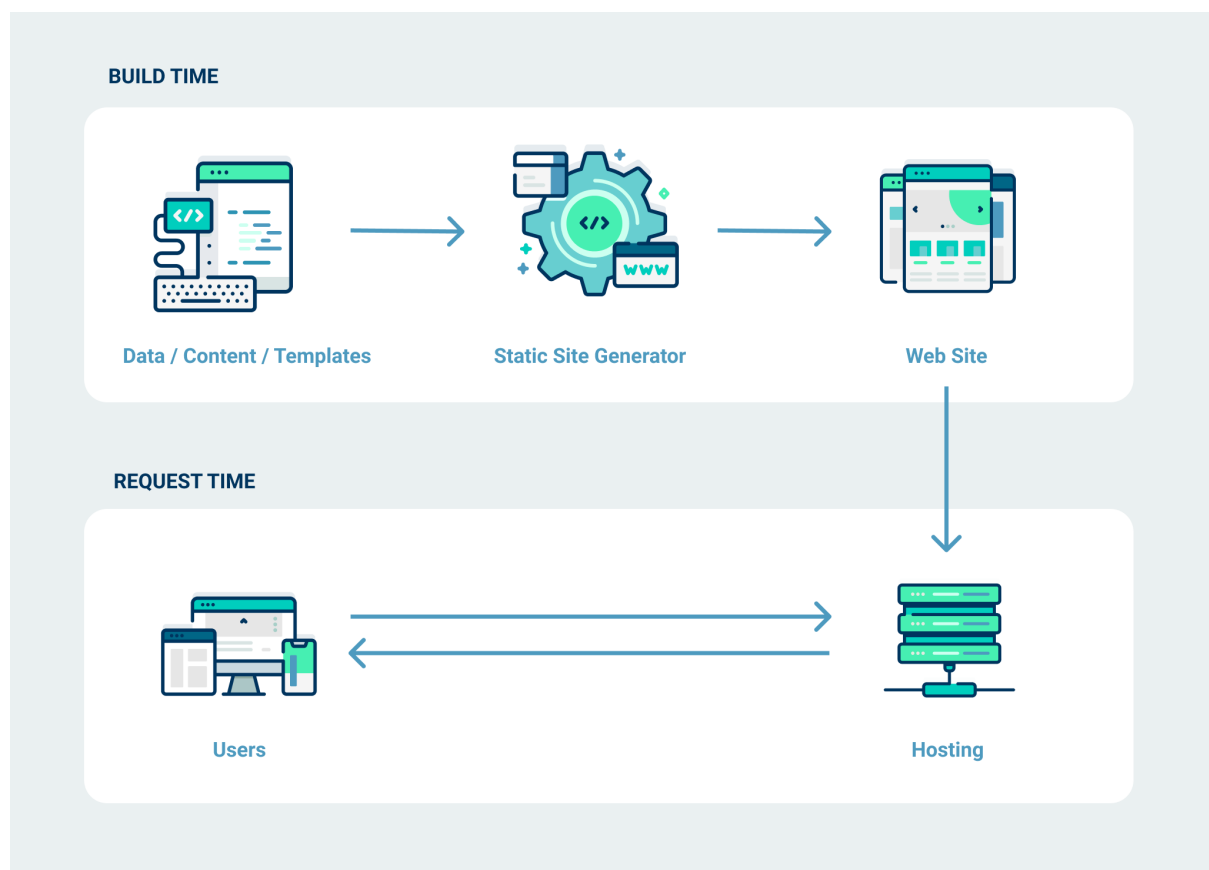


Figura 7: Flujo en SSG

Una de las **mayores plataformas para desplegar** Static Site Generators es <https://www.netlify.com/>. En el siguiente enlace encontrarás un libro gratuito de dicha plataforma con el título **Modern Web Development on the JAMStack**. Por cierto, JAM son las iniciales de **J**avascript, **A**PI's y **M**arkup

4 Tecnologías de desarrollo Web

Como hemos comentado en el apartado Aplicaciones Web, hoy en día se utilizan páginas dinámicas tanto del lado del servidor como del cliente.

El impacto de la Web ha propiciado la aparición de una gran cantidad de tecnologías, librerías, herramientas y estilos arquitectónicos para desarrollar una aplicación web. Para facilitar la tarea de escoger cuál es la tecnología más adecuada para un proyecto, es conveniente conocer los elementos más importantes desde un punto de vista de alto nivel para tener una visión global de la programación web.

Existen dos enfoques en el desarrollo de aplicaciones web:

- Creación de aplicaciones web con integración de tecnologías de desarrollo
- Creación de aplicaciones web con sistemas gestores de contenido

4.1 Integración de tecnologías de desarrollo

Una de las formas de crear una aplicación web es implementarla usando componentes ya existentes e integrándolos con tecnología propia para adecuarlos a la lógica de negocio. Una premisa importante en cualquier tipo de desarrollo es «*no reinventar la rueda*». Seguro que existe algún componente que ya hace lo que pretendemos hacer.

Dentro de las tecnologías de desarrollo, podemos distinguir entre:

- **Tecnologías de cliente:** Tecnologías que permiten crear interfaces de usuario atractivos y permiten la comunicación con el servidor. Basadas en HTML, CSS y JavaScript. Entre las de propósito general, cabe destacar [jQuery](#), [AngularJS](#), [Bootstrap](#). En el informe [Usage Statistics and Market Share of JavaScript Libraries for Websites, October 2020](#), podemos ver una relación más detallada y hay que tener en cuenta que una aplicación web puede usar más de una de estas tecnologías a la vez. También podemos consultar [List of Javascript Libraries](#), donde están clasificadas por categorías.

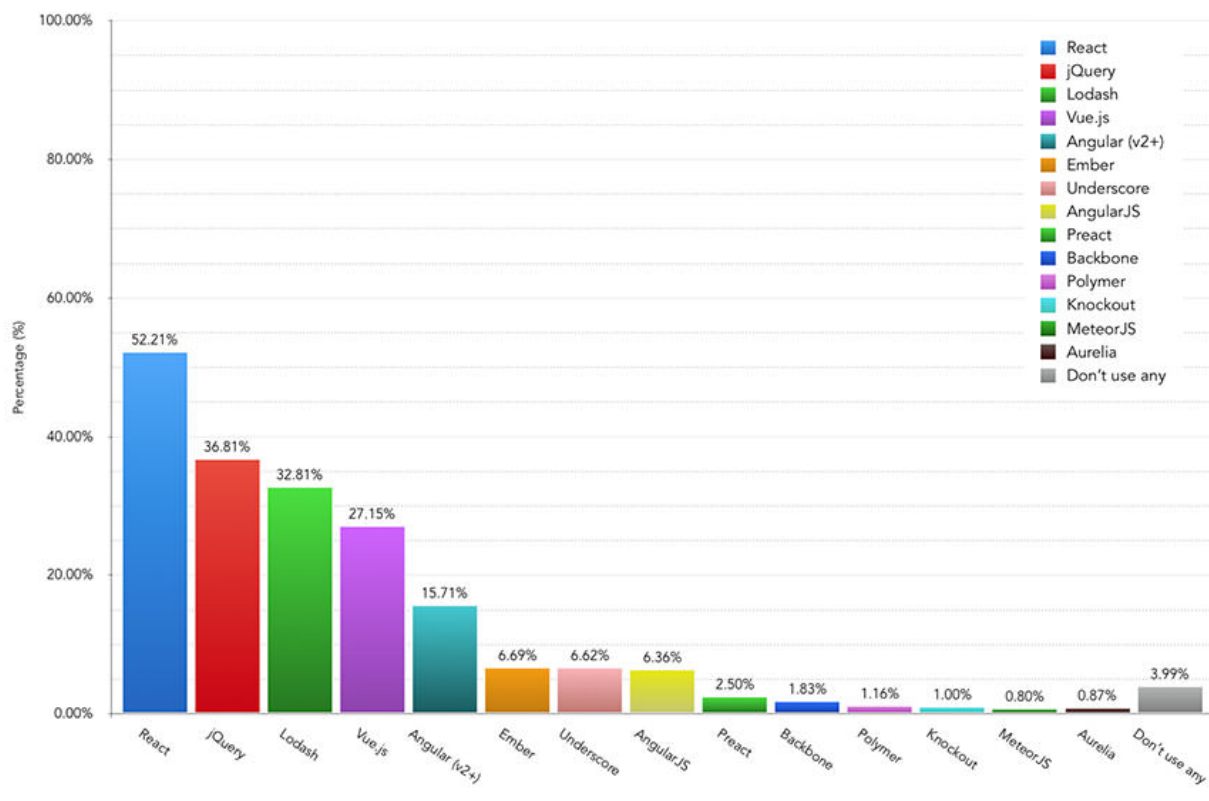


Figura 8: Lista de librerías Javascript

Fuente. <https://ashleynolan.co.uk/blog/frontend-tooling-survey-2019-results>

También es muy interesante consultar la web <https://2020.stateofjs.com/en-US/technologies/frontend-frameworks/>

- **Tecnologías de servidor:** Tecnologías que permiten implementar el comportamiento de la aplicación web en el servidor: lógica de negocio, generación de informes, compartir información entre usuarios, envío de correos, etc.

En **PHP**, tenemos [CodeIgniter](#), [Symfony](#) y [Laravel](#).



Figura 9: Tecnologías en el Servidor

PHP Frameworks. Fuente: <https://coderseye.com/best-php-frameworks-for-web-developers/>

En **Java**, [Spring MVC](#), [JSF](#), [Struts](#).

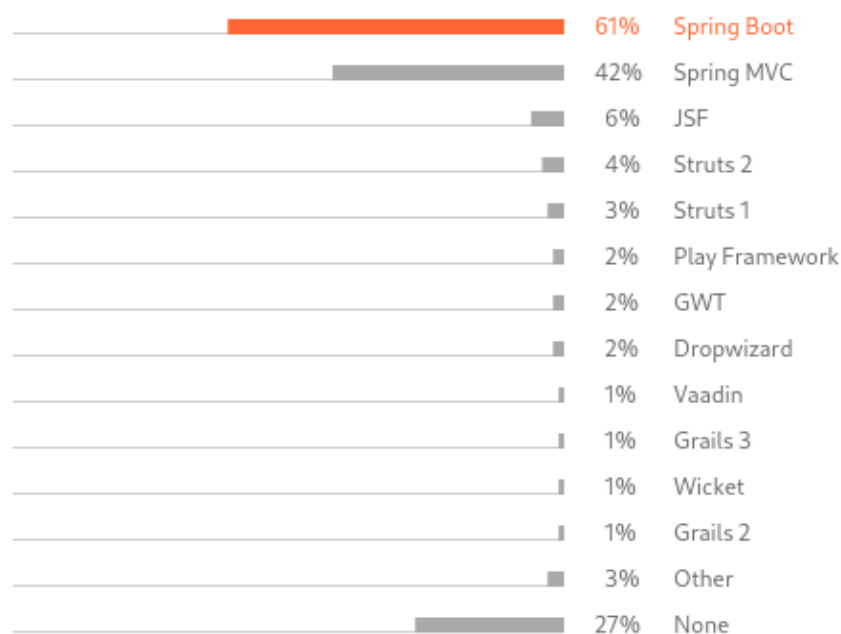


Figura 10: Java Frameworks

Fuente: <https://www.jetbrains.com/lp/devecosystem-2020/java/>

En **javascript**, tenemos [node.js](#) y el framework más utilizado es [Express](#).

- **Bases de datos:** La gran mayoría de las webs necesitan guardar información. Las bases de datos son una parte esencial del desarrollo web. Entre ellas destacan, Oracle, MySQL, Microsoft SQL Server, PostgreSQL. En el informe [DB-Engines Ranking - popularity ranking of database](#)

Rank			DBMS	Data
Oct 2020	Sep 2020	Oct 2019		
1.	1.	1.	Oracle +	Relati
2.	2.	2.	MySQL +	Relati
3.	3.	3.	Microsoft SQL Server +	Relati
4.	4.	4.	PostgreSQL +	Relati
5.	5.	5.	MongoDB +	Docu
6.	6.	6.	IBM Db2 +	Relati
7.	↑ 8.	7.	Elasticsearch +	Searc
8.	↓ 7.	8.	Redis +	Key-v
9.	9.	↑ 11.	SQLite +	Relati
10.	10.	10.	Cassandra +	Wide

[management systems](#) podemos ver la lista completa.

Fuente: <https://db-engines.com/en/ranking>

4.2 Sistemas gestores de contenido (CMS)

Existen aplicaciones web cuya principal funcionalidad es la publicación de contenido: blogs, páginas de empresas, organismos públicos, comercio electrónico, etc.

Todas estas webs tienen mucho en común, prácticamente sólo se diferencian en el contenido y en el aspecto gráfico.

Para desarrollar este tipo de webs, en vez de desarrollar la web con integración de tecnologías de desarrollo se usa una aplicación ya creada que se puede personalizar y adaptar (mayormente vía web).

A las aplicaciones de este tipo se las denomina **Sistemas Gestores de Contenido** (*Content Management Systems*).

Entre las **ventajas** de contar con un CMS están:

1. **Fácilmente actualizable.** La edición del contenido está separada del diseño y funcionalidad del sitio. Esto permite introducir contenido a usuarios con poca formación técnica sin interferir en el diseño.
2. A cada usuario se le pueden asignar permisos de acceso selectivo basados en sus **roles** (por ejemplo, puede permitir que algunos usuarios solo agreguen y edite su propio contenido, mientras que otros le dan acceso universal).
3. **Uso de plantillas totalmente personalizables.** Tanto para el contenido como para el diseño de tu propio sitio web. Esta es una de las partes que más suelen asustar si no se tienen conocimientos. Pero ahora no hace falta ser un experto en diseño.
4. Los componentes básicos de la aplicación como menús, cabeceras, pies de página y laterales se mantienen desde la interfaz de administración.
5. Además, **los mejores CMS incluyen plugins** para poder sacar aún más partido a todas las funcionalidades

Evidentemente, no son todas ventajas. Los principales **inconvenientes** de usar un CMS son:

1. Desafortunadamente, como el código fuente está disponible al público en general y *todas las aplicaciones tienen bugs* de seguridad, hay algunos hackers malévolos por ahí que pueden averiguar cómo entrar en estas plataformas; por lo que la seguridad requerirá precauciones adicionales.
2. Hacer que el sitio web se vea exactamente como queremos puede suponer desafío. Esto es cierto en algunos *frameworks* de CMS más que otros, pero todos presentan un poco más de trabajo para definir un «estilo» propio.

3. El CMS almacena todo por separado, luego lo monta en el momento en que el cliente web solicita una página, lo que significa que pueden ser lentos; sin embargo, esto se puede mitigar mediante el uso de un almacenamiento en [caché](#) fuerte y eficaz y con una [Red de Entrega de Contenidos](#) (*Content Delivery Network* - CDN).
4. Limitaciones de funcionalidad: Hay algunas cosas que no se pueden hacer en un CMS, al menos no sin reescribir parte del código.

Los principales CMS *libres* propiamente dichos son [WordPress](#), [Drupal](#) y [Joomla](#). Para una estadística más detallada podemos consultar la página de tendencias de [buildwith.com](#)

Top In CMS Usage Distribution in the Top 1 Million Sites

Technology	Websites	%
 WordPress	381,513	38.15
 WP Engine	29,023	2.9
 Drupal	28,625	2.86
 Plesk	26,039	2.6
 Google Search Appliance	20,724	2.07
 Joomla!	16,967	1.7
 Squarespace	14,426	1.44
 cPanel	14,327	1.43
 Blogger	11,186	1.12
 Unbounce	10,750	1.08
 Atlassian Cloud	9,198	0.92
 Wix	9,076	0.91

Figura 11: Top CMS

Fuente: <https://cms2cms.com/uncategorized/what-cms-will-be-your-choice-in-z017-facts-and-figures/>

Existen otros tipos de aplicaciones orientadas a otros ámbitos, como por ejemplo al comercio electrónico. Entre ellas cabe destacar [WooCommerce](#) y [Magento](#) que se utilizan para la creación de tiendas virtuales. Podemos ver una estadística completa en [builtwith.com](#)

Top In eCommerce Usage Distribution in the Top 1 Million Sites





Technology	Websites	%
 WooCommerce Checkout	48,638	4.86
 Shopify	35,800	3.58
 Magento	12,914	1.29
 OpenCart	5,692	0.57

Figura 12: Top eCommerce

4.2.1 Headless CMS Ahora bien, en los últimos años hemos podido presenciar el surgimiento de una nueva clase de CMS, el «headless CMS». Ciertamente, el término traducido al español no resulta muy explicativo («CMS sin cabeza»), por lo que vamos a explicar un poco en qué consisten.

Este tipo de CMS sólo se encargan de generar una interfaz web para introducir los datos. Una vez creada esta interfaz y alimentada con datos, proveen de una API REST, generalmente en JSON, para consultarlos.

Después ya sólo hace falta conectar los datos con:

1. Un front-end para Web
2. Otro para móvil
3. Otro para relojes inteligentes.
4. ...

Parece que no tiene sentido, ¿verdad?

Pues debe tenerlo porque existe un gran ecosistema alrededor de esta tecnología como se puede comprobar en <https://jamstack.org/headless-cms/>

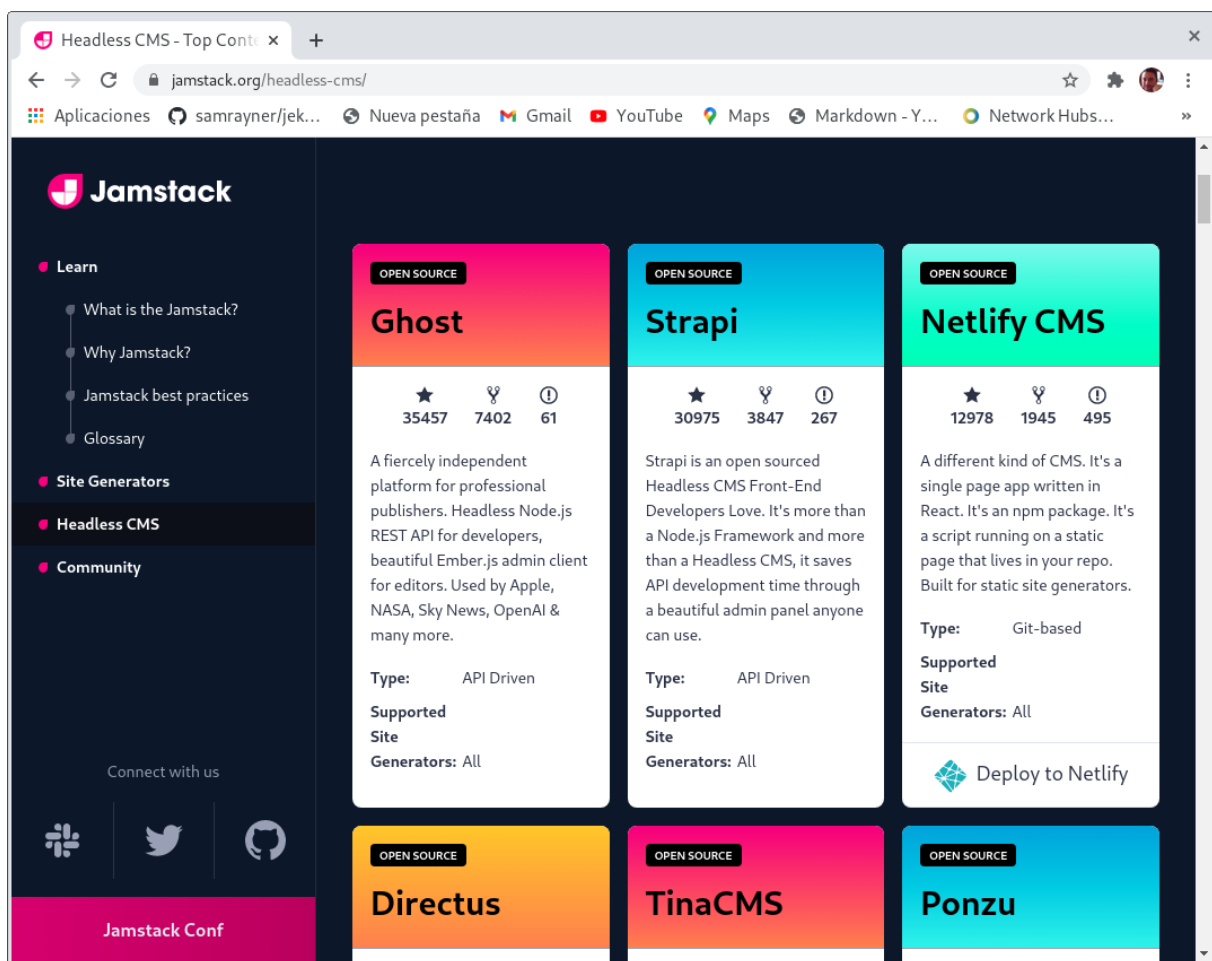


Figura 13: Jamstack

Podéis consultar más información en <https://www.genbeta.com/desarrollo/headless-cms-que-que-se-diferencian-tradicionales>

5 Servidores web y de aplicaciones

Una vez tenemos definida la funcionalidad de nuestra aplicación y las tecnologías de desarrollo que vamos a utilizar, hemos de escoger entre servir las peticiones a nuestra aplicación mediante un servidor web o mediante un servidor de aplicaciones. Esta decisión viene marcada, evidentemente, por el tipo de tecnología que hayamos escogido para implementar el lado del servidor. Si hemos escogido implementar nuestra aplicación con Java, usaremos un servidor de aplicaciones; mientras si la tecnología es PHP o ASP.NET escogeremos un servidor web.

5.1 Servidor Web

Un servidor Web controla el protocolo HTTP. Cuando el servidor Web recibe una solicitud mediante el **protocolo HTTP**, responde con una respuesta HTTP, como el envío de una página HTML. Para procesar una solicitud, un servidor Web puede responder con una página o imagen HTML estática, enviar una redirección o delegar la generación de respuestas dinámicas a algún otro programa, como secuencias de comandos CGI, PHP, JSP (Servidores Java), servlets, ASPs (Active Server Pages), JavaScripts del lado del servidor, o alguna otra tecnología del lado del servidor.

Cualquiera que sea su propósito, tales programas del lado del servidor generan una respuesta, la mayoría de las veces en HTML, para ver en un navegador Web.

Este modelo de delegación de un servidor Web es bastante simple. Cuando una solicitud entra en el servidor Web, el servidor Web simplemente pasa la solicitud al programa más capaz de manejarla.

El servidor Web no proporciona ninguna funcionalidad más allá de simplemente proporcionar un entorno en el que el programa del lado del servidor puede ejecutar y devolver las respuestas generadas.

Como vemos en el siguiente gráfico de uso, el servidor más usado es [Apache](#), seguido de [Nginx](#). Si desarrollamos una aplicación en ASP.NET, escogeríamos [Internet Information Services \(IIS\)](#) de Microsoft.

Top In Web Server Usage Distribution in the Top 1 Million Sites

Technology	Websites	%
 Apache	347,508	34.75
 nginx	336,791	33.68
 IIS	127,121	12.71
 LiteSpeed	65,292	6.53
 Varnish	35,923	3.59
 BIG-IP	11,540	1.15
 Apache Tomcat Coyote	10,723	1.07
 Phusion Passenger	8,805	0.88
 Rack Cache	8,143	0.81

Figura 14: Uso de servidores web

Fuente: <https://trends.builtwith.com/web-server>



Figura 15: apache



Figura 16: Nginx

5.2 Servidor de aplicaciones

Un servidor de aplicaciones expone la lógica empresarial a las aplicaciones cliente a través de **varios protocolos**, posiblemente incluyendo HTTP.

Aunque un servidor Web se ocupa principalmente del envío de HTML para su visualización en un navegador Web, un servidor de aplicaciones proporciona acceso a la lógica empresarial para su uso por los programas de aplicación del cliente.

Como consecuencia del éxito del lenguaje de programación Java, el término servidor de aplicaciones usualmente hace referencia a un servidor de aplicaciones Java EE, aunque también hay servidores aplicaciones para PHP como [Zend Server](#).

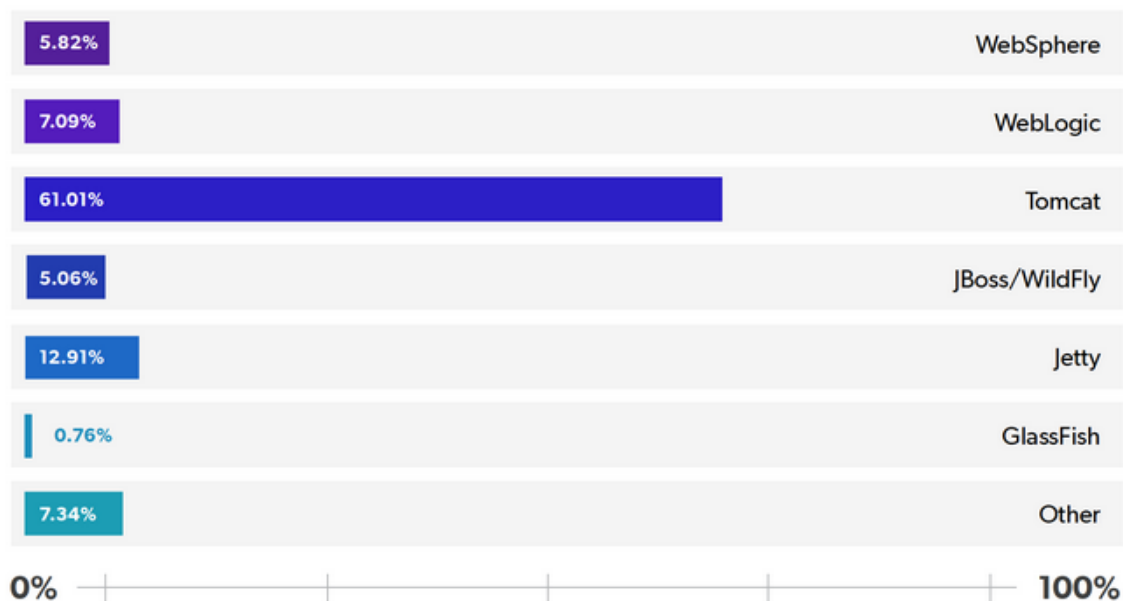


Figura 17: Servidor de aplicaciones

Fuente: <https://www.jrebel.com/blog/2020-java-technology-report#application-server>

6 Despliegue de Aplicaciones web

Cuando se crea una **aplicación web** en nuestro ordenador suele hacerse en un **Entorno de Desarrollo** (*Development Environment*) al que sólo somos capaces de acceder desde ese ordenador.

Esto se conoce como entorno de **desarrollo local o localhost**. Trabajando en localhost, el código de la aplicación se mantiene en nuestro ordenador y será accesible únicamente desde el mismo. Pero para hacer la aplicación web accesible a todo el mundo, se necesita ponerla a disposición en un ordenador público, accesible a través de una URL. A estos ordenadores se les denomina **Servidores Web** y el entorno en el que se despliegan se denomina **Entorno de Producción** (*Production Environment*).

Estos servidores web son los encargados de ejecutar la aplicación web y entregar las páginas a cualquiera que las pida.

El proceso de mover una aplicación web desde localhost a un servidor web se denomina **Despliegue (_Deployment_)**.

Desde el momento que desplegamos una aplicación en un servidor web tenemos una versión que está públicamente disponible por cualquiera a través de la URL y todavía mantenemos la versión que

corre en nuestro anfitrión local que continuamos retocando y probando. Periódicamente, deberemos actualizar (desplegar) el código al servidor web después de que hayamos probado los nuevos cambios en localhost e implementado procesos **CD/CI** (Continuos Development/Continuos Integration)

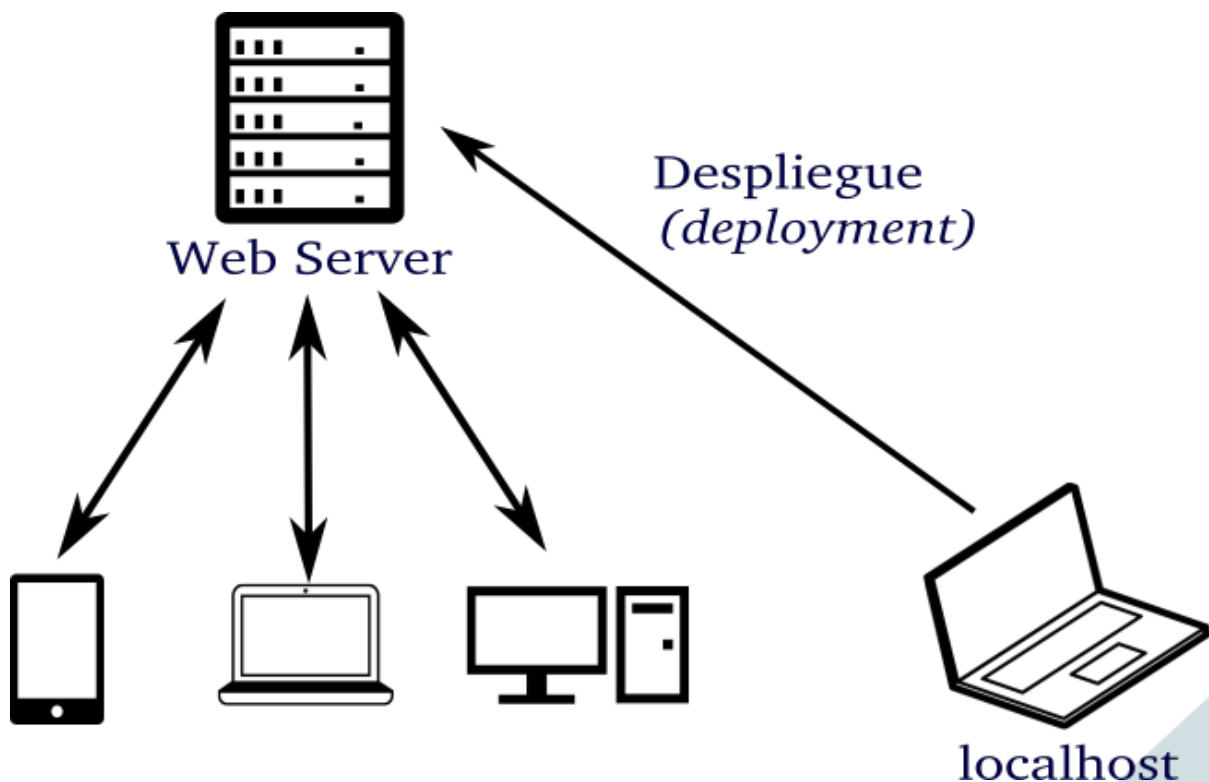


Figura 18: Despliegue

Este proceso no es tan sencillo como parece, sobre todo cuando hablamos de aplicaciones empresariales en las que participan gran cantidad de profesionales tanto en desarrollo como en sistemas, así como multitud de sistemas operativos, servidores de aplicaciones, API's end-points, servidores web, contenedores, orquestadores, etc.

De hacer todo este proceso de forma segura es de lo que trata este módulo profesional.

Un caso real: Storyblocks

Esta es una traducción del artículo [original](#)

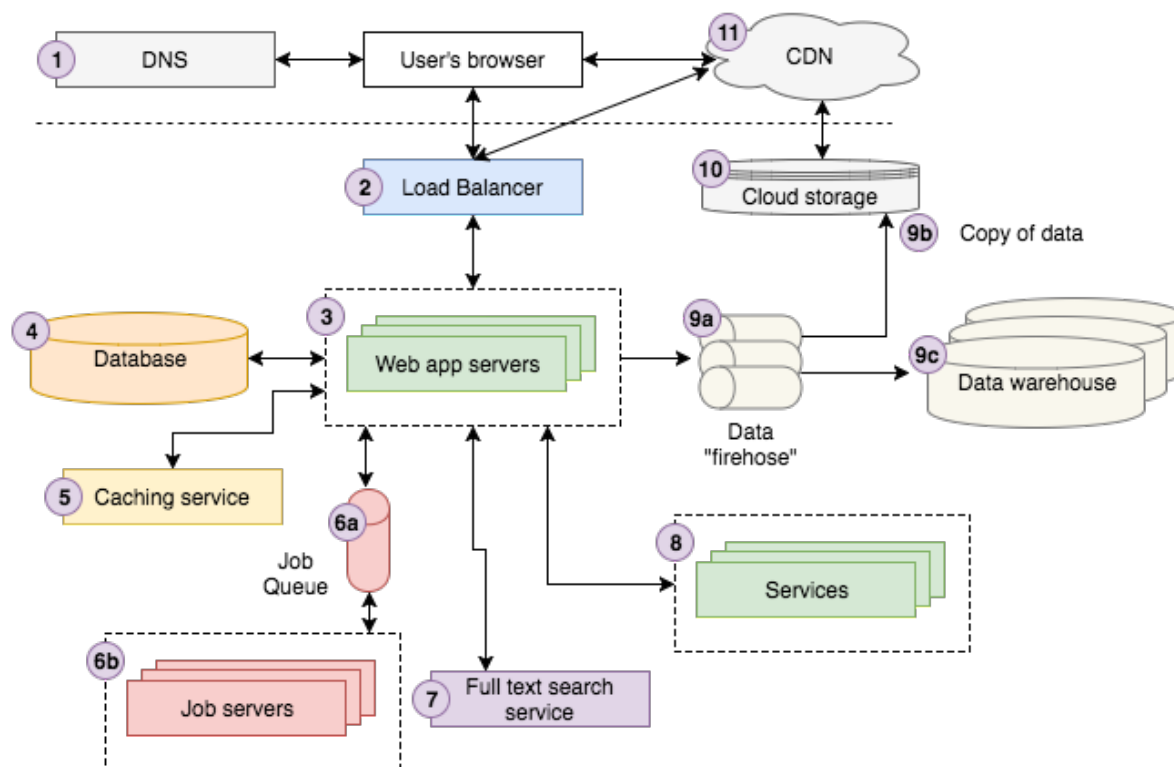


Figura 19: Storybloks

El diagrama de arriba es una representación bastante buena de la arquitectura en [Storyblocks](#). Si no eres un desarrollador web experimentado, es probable que lo encuentres complicado. El recorrido explicado debajo debería ser más accesible antes de profundizar en los detalles de cada componente.

Un usuario busca en Google «Strong Beautiful Fog And Sunbeams In The Forest». El primer resultado es [Storyblocks](#), nuestro sitio principal de fotos y vectores. El usuario hace clic en el resultado que redirige su navegador a la página de detalles de la imagen. Entre bambalinas, el navegador del usuario envía una solicitud a un servidor DNS para buscar cómo contactar a Storyblocks y luego envía la solicitud.

La solicitud llega a nuestro balanceador de carga (**load balancer**), que elige aleatoriamente uno de los 10 servidores web que tenemos para ejecutar el sitio en ese momento para procesar la solicitud. El servidor web busca información sobre la imagen de nuestro servicio de almacenamiento en caché (**caching service**) y recupera los datos restantes de la base de datos. Observamos que el perfil de color de la imagen aún no se ha calculado, por lo que enviamos un trabajo de «perfil de color» a nuestra cola de trabajos (**job queue**), que nuestros servidores de trabajo procesarán de forma asincrónica, actualizando la base de datos de forma adecuada con

los resultados.

A continuación, intentamos encontrar fotos similares enviando una solicitud a nuestro servicio de búsqueda de texto (**full text service**) completo utilizando el título de la foto como entrada. El usuario está registrado en Storyblocks como miembro, por lo que buscamos la información de su cuenta en nuestro servicio de cuenta (**account service**). Finalmente, lanzamos un evento de visualización de página a nuestra firehose de datos (**data firehose**) para registrarlo en nuestro sistema de almacenamiento en la nube (**cloud storage**) y finalmente cargarlo en nuestro almacén de datos (**data warehouse**), que los analistas usan para ayudar a responder preguntas sobre el negocio.

El servidor ahora muestra la vista como HTML y la envía al navegador del usuario, pasando primero a través del equilibrador de carga. La página contiene activos de Javascript y CSS que cargamos en nuestro sistema de almacenamiento en la nube, que está conectado a nuestro **CDN**, por lo que el navegador del usuario contacta al CDN para recuperar el contenido. Por último, el navegador muestra visiblemente la página para que el usuario la vea.

A continuación, te guiaré a través de cada componente, proporcionando una introducción de «101» a cada uno que te proporcionará un buen modelo mental para pensar a través de la arquitectura web en el futuro.

1 DNS

DNS significa «Servidor de nombres de dominio» (**Domain Name Server**) y es una tecnología troncal que hace posible la red mundial. En el nivel más básico, DNS proporciona una búsqueda de clave / valor desde un nombre de dominio (por ejemplo, google.com) a una dirección IP (por ejemplo, 85.129.83.120), que es necesaria para que tu computadora enrute una solicitud al servidor. Analizando los números de teléfono, la diferencia entre un nombre de dominio y una dirección IP es la diferencia entre «llamar a John Doe» y «llamar al 201-867-5309». Al igual que necesitabas una guía telefónica para buscar el número de John en los viejos tiempos, necesita DNS para buscar la dirección IP de un dominio. Entonces, puedes pensar en DNS como el directorio telefónico de Internet.

2 Balanceador de carga (load balancer)

Antes de profundizar en los detalles sobre el equilibrio de carga, debemos dar un paso atrás para analizar el escalado horizontal/vertical de la aplicación. ¿Qué son y cuál es la diferencia? De una manera muy sencilla, la escala **horizontal** significa que escala agregando **más máquinas** a su grupo de recursos mientras que la escala **vertical** significa que escala agregando **más potencia** (por ejemplo, CPU, RAM) a una máquina existente.

En el desarrollo web, (casi) siempre quieres escalar horizontalmente porque, para simplificar, las cosas se rompen. Los servidores se bloquean aleatoriamente. Las redes se degradan. Centros de datos completos ocasionalmente se desconectan. Tener más de un servidor te permite planificar las interrupciones para que su aplicación continúe ejecutándose. En otras palabras, tu aplicación es «tolerante a fallos» (**fault tolerant**). En segundo lugar, la escala horizontal te permite acoplar al menos partes diferentes de tu aplicación backend (servidor web, base de datos, servicio X, etc.) ejecutando cada uno de ellos en servidores diferentes. Por últimos, puede alcanzar una escala en la que ya no sea posible escalar verticalmente. No hay una computadora en el mundo lo suficientemente grande como para hacer todos los cálculos de su aplicación. Piensa en la plataforma de búsqueda de Google como un ejemplo por excelencia, aunque esto se aplica a las empresas a escalas mucho más pequeñas. Storyblocks, por ejemplo, ejecuta de 150 a 400 instancias de [AWS EC2](#) en cualquier momento dado. Sería un desafío proporcionar toda la potencia de cálculo a través de escalamiento vertical.

Ok, de vuelta a los balanceadores de carga. Son la salsa mágica que hace que el escalado horizontal sea posible. Enrutan las solicitudes entrantes a uno de los muchos servidores de aplicaciones que suelen ser clones / imágenes espejo entre sí y envían la respuesta desde el servidor de la aplicación al cliente. Cualquiera de ellos debe procesar la solicitud de la misma manera, por lo que solo se trata de distribuir las solicitudes en el conjunto de servidores para que ninguno de ellos esté sobrecargado.

Eso es. Conceptualmente, los equilibradores de carga son bastante sencillos. Seguramente hay más complejidad internamente pero no hay necesidad de sumergirse para nuestra versión 101.

El siguiente artículo, profundiza en los [balanceadores de carga](#).

3 Servidores de aplicaciones web

En un nivel alto, los servidores de aplicaciones web son relativamente simples de describir. Ejecutan la lógica de negocio principal que maneja la solicitud de un usuario y envía de vuelta HTML al navegador del usuario. Para hacer su trabajo, normalmente se comunican con una variedad de infraestructura de back-end, como bases de datos, capas de almacenamiento en caché, colas de trabajos, servicios de búsqueda, otros microservicios, colas de datos / registro y más. Como se mencionó anteriormente, normalmente se tienen al menos dos y muchas veces más, conectados a un balanceador de carga para procesar las solicitudes de los usuarios.

Debes saber que las implementaciones del servidor de aplicaciones requieren elegir un idioma específico (Node.js, Ruby, PHP, Scala, Java, C # .NET, etc.) y un marco web MVC para ese idioma (Express for Node.js, Ruby on Rails , Play para Scala, Laravel para PHP, Spring para Java, etc.). Sin embargo, profundizar en los detalles de estos lenguajes y frameworks está más allá del alcance de este artículo.

4 Servidores de bases de datos

Cada aplicación web moderna aprovecha una o más bases de datos para almacenar información. Las bases de datos proporcionan formas de definir sus estructuras de datos, insertar nuevos datos, encontrar datos existentes, actualizar o eliminar datos existentes, realizar cálculos entre los datos, y más. En la mayoría de los casos, los servidores de aplicaciones web hablan directamente con uno, al igual que los servidores de trabajo. Además, cada servicio backend puede tener su propia base de datos que está aislada del resto de la aplicación.

Aunque estoy evitando una inmersión profunda en tecnologías particulares para cada componente de la arquitectura, les estaría haciendo un flaco favor sin mencionar el siguiente nivel de detalle para las bases de datos: SQL y NoSQL.

SQL, significa «Structured Query Language» y se inventó en la década de 1970 para proporcionar una forma estándar de consulta de conjuntos de datos relacionales que era accesible para una amplia audiencia. Las bases de datos SQL almacenan datos en tablas que están vinculadas entre sí a través de ID comunes, generalmente enteros. Veamos un ejemplo simple de almacenamiento de información histórica de direcciones para los usuarios. Es posible que tengas dos tablas, usuarios y direcciones de usuario, vinculados entre sí por la identificación del usuario. La siguiente imagen muestra para una versión simplista. Las tablas están vinculadas porque la columna *user_id* en *user_addresses* es una «clave externa» a la columna *id* en la tabla de *users*.

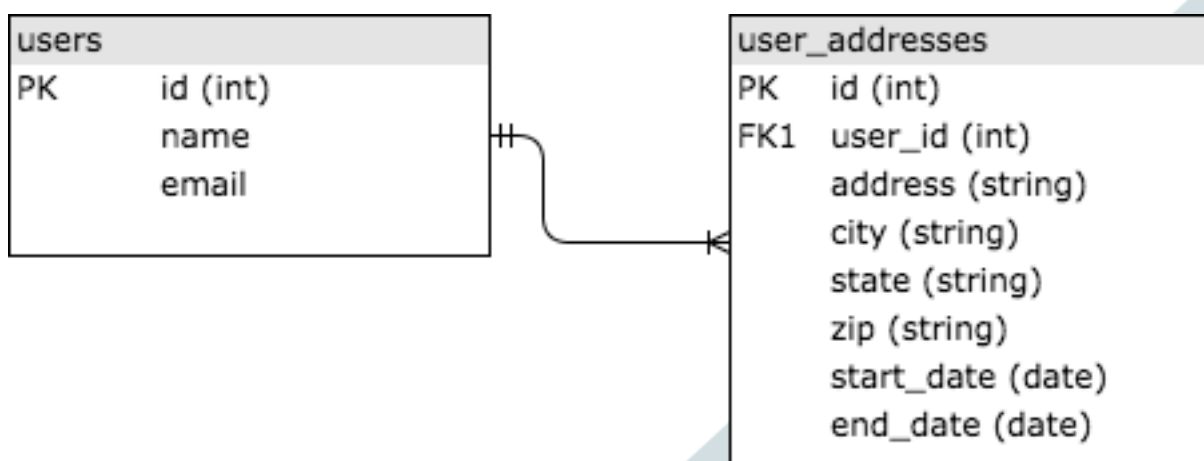


Figura 20: SQL

Si no sabes mucho sobre SQL, te recomiendo que consultes un tutorial como el que puedes encontrar en Khan Academy [aquí](#). Es omnipresente en el desarrollo web, por lo que al menos querrás saber los conceptos básicos para poder diseñar una aplicación de forma adecuada.

NoSQL, que significa «No SQL», es un conjunto de tecnologías de bases de datos más reciente que

ha surgido para manejar cantidades masivas de datos que pueden ser producidos por aplicaciones web de gran escala (la mayoría de las variantes de SQL no se escalan horizontalmente muy bien y solo pueden escalar verticalmente hasta un cierto punto). Si no sabes nada acerca de NoSQL, te recomiendo comenzar con algunas presentaciones de alto nivel como estas:

- <https://www.w3resource.com/mongodb/nosql.php>
- <http://www.kdnuggets.com/2016/07/seven-steps-understanding-nosql-databases.html>
- <https://www.mongodb.com/presentations/back-to-basics-webinar-1-introduction-to-nosql>

También debes tener en cuenta que, en general, la industria se está alineando con SQL como una interfaz incluso para las bases de datos NoSQL, por lo que realmente deberías aprender SQL si no lo sabes. Casi no hay forma de evitarlo estos días.

5 Servicio de caché

Un servicio de almacenamiento en caché proporciona un almacén de datos de clave/valor simple que permite guardar y buscar información en un tiempo cerca de $O(1)$ (esta notación describe el tiempo en ejecutar un proceso en función de los datos. En este caso, el 1 indica que es un tiempo constante, independientemente de la cantidad de claves que haya en el conjunto). Las aplicaciones generalmente aprovechan los servicios de almacenamiento en caché para guardar los resultados de cálculos costosos, de modo que es posible recuperar los resultados de la memoria caché en lugar de volver a calcularlos la próxima vez que se necesiten. Una aplicación puede almacenar en caché los resultados de una consulta de base de datos, llamadas a servicios externos, HTML para una URL determinada y muchas más. Aquí hay algunos ejemplos de aplicaciones del mundo real:

- Google almacena en caché los resultados de búsqueda para consultas de búsqueda comunes como «perro» o «Taylor Swift» en lugar de volver a computarlos cada vez
- Facebook guarda en caché gran parte de la información que ves cuando inicias sesión, como datos de publicaciones, amigos, etc. Lee [aquí](#) un artículo detallado sobre la tecnología de caché de Facebook.
- Storyblocks almacena en caché el resultado HTML de la representación de React del lado del servidor, resultados de búsqueda, resultados de escritura anticipada y más.

Las dos tecnologías de servidor de caché más extendidas son **Redis** y **Memcache**.

6 Colas de trabajos y servidores

La mayoría de las aplicaciones web necesitan realizar un trabajo asíncrono entre bastidores que no está directamente asociado con la respuesta a la solicitud de un usuario. Por ejemplo, Google necesita

rastrear e indexar todo Internet para poder devolver resultados de búsqueda. No hace esto cada vez que buscas. En cambio, rastrea la web de forma asincrónica, actualizando los índices de búsqueda a lo largo del camino.

Si bien hay diferentes arquitecturas que permiten que se realice un trabajo asíncrono, la más ubicua es lo que llamaré la arquitectura de «cola de trabajos». Consta de dos componentes: una cola de «trabajos» que deben ejecutarse y uno o más servidores de trabajos (a menudo llamados «trabajadores») que ejecutan los trabajos en la cola.

Las colas de trabajos almacenan una lista de trabajos que deben ejecutarse de forma asincrónica. Las más simples son las colas de primero en entrar, primero en salir (**FIFO: First In First Out**), aunque la mayoría de las aplicaciones terminan necesitando algún tipo de sistema de colas de prioridad. Cuando la aplicación necesita que se ejecute un trabajo, ya sea en algún tipo de programa regular o según lo determinen las acciones del usuario, simplemente agrega el trabajo adecuado a la cola.

Storyblocks, por ejemplo, aprovecha una cola de trabajos para potenciar una gran parte del trabajo detrás de escena requerido para respaldar nuestros mercados. Ejecutamos trabajos para codificar vídeos y fotos, procesar CSV para el etiquetado de metadatos, agregar estadísticas de usuario, enviar correos electrónicos de restablecimiento de contraseña y más. Comenzamos con una cola FIFO simple aunque nos actualizamos a una cola de prioridad para garantizar que las operaciones urgentes como el envío de correos electrónicos de restablecimiento de contraseña se completaran lo antes posible.

Los servidores de trabajos procesan trabajos. Realizan una consulta en la cola de trabajos para determinar si hay trabajo por hacer y, si los hay, extraen un trabajo de la cola y lo ejecutan. Los idiomas subyacentes y las opciones de frameworks son tan numerosos como para los servidores web, por lo que no profundizaré en los detalles de este artículo.

Por ejemplo, en PHP se puede utilizar [Bernard](#) y se pueden implementar colas por ejemplo con [beanstalkd](#) o [RabbitMQ](#)

7 Servicio de búsqueda de texto completo

Muchas, si no la mayoría, de las aplicaciones web admiten algún tipo de función de búsqueda en la que un usuario proporciona una entrada de texto (a menudo llamada «consulta o query») y la aplicación devuelve los resultados más «relevantes». La tecnología que alimenta esta funcionalidad se conoce como «[búsqueda de texto completo](#)» (**full-text search**), que aprovecha un [índice invertido](#) para buscar rápidamente documentos que contienen las palabras clave de consulta.

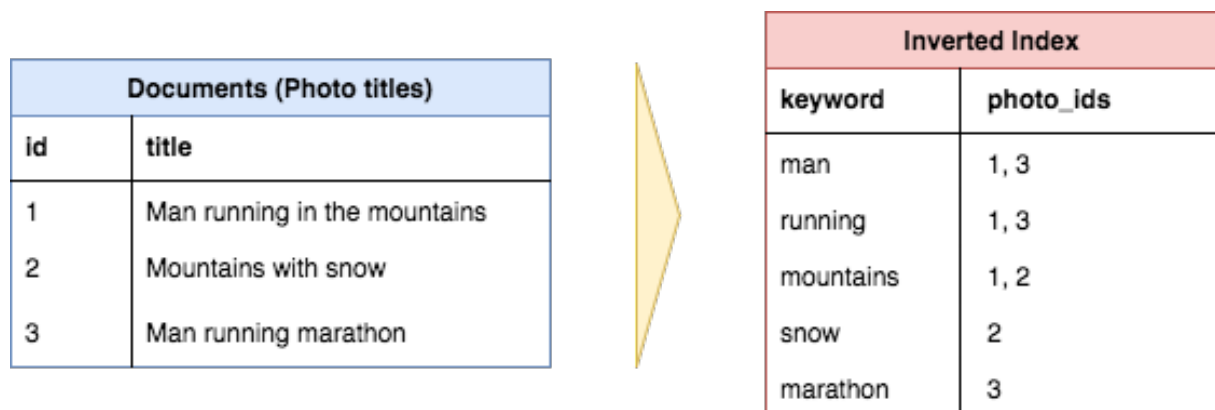


Figura 21: Texto completo

El ejemplo muestra cómo tres títulos de documentos se convierten en un índice invertido para facilitar la búsqueda rápida desde una palabra clave específica a los documentos con esa palabra clave en el título. Ten en cuenta que las palabras comunes como «en», «el», «con», etc. (llamadas palabras de finalización - **stop words**-) generalmente no se incluyen en un índice invertido.

Si bien es posible realizar búsquedas de texto completo directamente desde algunas bases de datos (por ejemplo, MySQL admite la búsqueda de texto completo), es típico ejecutar un «servicio de búsqueda» separado que computa y almacena el índice invertido y proporciona una interfaz de consulta. La plataforma de búsqueda de texto completo más popular en la actualidad es [Elasticsearch](#), aunque hay otras opciones como [Sphinx](#) o [Apache Solr](#).

8 Servicios

Una vez que una aplicación alcanza cierta escala, es probable que existan ciertos «servicios» que se diseñen para ejecutarse como aplicaciones separadas. No están expuestos al mundo externo, pero la aplicación y otros servicios interactúan con ellos. Storyblocks, por ejemplo, tiene varios servicios operacionales y planificados:

- El servicio de cuenta almacena datos de usuario en todos nuestros sitios, lo que nos permite ofrecer oportunidades de venta cruzada y crear una experiencia de usuario más unificada.
- El servicio de contenido almacena metadatos para todo nuestro contenido de vídeo, audio e imagen. También proporciona interfaces para descargar el contenido y ver el historial de descargas.
- El servicio de pago proporciona una interfaz para facturar tarjetas de crédito de clientes.
- El servicio HTML → PDF proporciona una interfaz simple que acepta HTML y devuelve un documento PDF correspondiente.

9 Datos

Hoy, las compañías viven y mueren en base en que saben aprovechar los datos. Casi todas las aplicaciones actualmente, una vez que alcanzan cierta escala, aprovechan un flujo de datos o tubería (**data pipeline**) para garantizar que los datos se puedan recopilar, almacenar y analizar. Una tubería típica tiene tres etapas principales:

1. La aplicación envía datos, generalmente eventos sobre las interacciones del usuario, a los datos «firehose», que proporciona una interfaz de transmisión para ingerir y procesar los datos. A menudo, los datos brutos se transforman o aumentan y pasan a otro firehose. [AWS Kinesis](#) y [Kafka](#) son las dos tecnologías más comunes para este propósito.
2. Los datos brutos (**raw data**), así como los datos finales transformados/aumentados se guardan en el almacenamiento en la nube. AWS Kinesis proporciona una configuración llamada «firehose» que hace que guardar los datos sin formato en su almacenamiento en la nube (S3) sea extremadamente fácil de configurar.
3. Los datos transformados / aumentados a menudo se cargan en un almacén de datos para su análisis. Usamos AWS Redshift, al igual que una porción grande y en crecimiento del mundo de las startups, aunque las compañías más grandes a menudo usan Oracle u otras tecnologías de almacenamiento patentadas. Si los conjuntos de datos son lo suficientemente grandes, puede ser necesaria una tecnología NoSQL MapReduce similar a [Hadoop](#) para el análisis.

10 Almacenamiento en la nube

«El almacenamiento en la nube es una forma simple y escalable de almacenar, acceder y compartir datos a través de Internet» según AWS. Puedes usarlo para almacenar y acceder más o menos a cualquier cosa que almacenes en un sistema de archivos local con los beneficios de poder interactuar con él a través de una API RESTful a través de HTTP. La oferta S3 de Amazon es de lejos el almacenamiento en la nube más popular disponible en la actualidad y en el que confiamos extensamente aquí en Storyblocks para almacenar nuestros activos de vídeo, foto y audio, nuestro CSS y Javascript, nuestros datos de eventos de usuario y mucho más.

11 CDN

CDN significa «Content Delivery Network» y la tecnología proporciona una forma de servir activos (**assets**) como HTML estático, CSS, Javascript e imágenes en la web mucho más rápido que servirlos desde un solo servidor de origen. Funciona distribuyendo el contenido a través de muchos servidores de todo el mundo para que los usuarios terminen descargando activos desde su servidor más cercano en lugar del servidor de origen. Por ejemplo, en la imagen siguiente, un usuario en España solicita una

página web desde un sitio con servidores de origen en Nueva York, pero los activos estáticos para la página se cargan desde un servidor «**edge**» de CDN en Inglaterra, evitando muchas peticiones HTTP transatlánticas lentas.

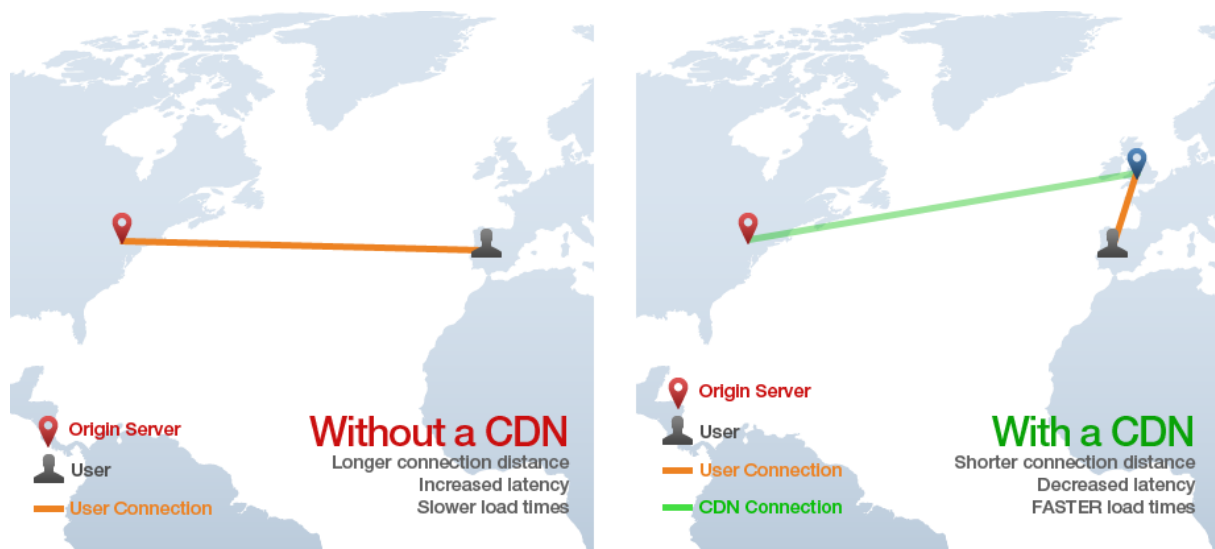


Figura 22: CDN

12 Desde el punto de vista de la seguridad

Todos estos servicios son potencialmente atacables por un hacker. Sin duda alguna es seguro que van a tener algún fallo de seguridad en algún momento de su ciclo de vida. Y en este diagrama no se habla de la infraestructura necesaria para implementar el ciclo **CD/CI**.

Seguro que ahora ya comprendemos la necesidad de securizar todo nuestro proceso de puesta en producción.

Adaptado de los siguientes materiales

Antonio LaTorre (atorre@fi.upm.es)

- http://laurel.datsi.fi.upm.es/_media/docencia/asignaturas/daw/pub/2015_2016/daw-tema1.1.pdf
- http://laurel.datsi.fi.upm.es/_media/docencia/asignaturas/daw/pub/2015_2016/daw-tema1.2.pdf

Adobe. Aspectos básicos de las aplicaciones web

<https://helpx.adobe.com/es/dreamweaver/using/web-applications.html>

App server, Web server: What's the difference? | JavaWorld

<https://www.javaworld.com/article/2077354/learn-java/app-server-web-server-what-s-the-difference.html>

Otros materiales

<https://www.hongkiat.com/blog/static-site-generators/>

<https://www.yeeply.com/blog/6-tipos-desarrollo-de-aplicaciones-web/>

<https://www.sitepoint.com/front-end-tooling-trends-2017/>

<http://www.antevenio.com/blog/2017/06/los-6-mejores-gestores-de-contenido-cms>

<http://glasscanopy.com/website-development-pros-cons-using-cms/>

<https://www.netlify.com/blog/2020/04/14/what-is-a-static-site-generator-and-3-ways-to-find-the-best-one/>

<https://engineering.videoblocks.com/web-architecture-101-a3224e126947>

Imagen de cabecera obtenida en:

Vector de Dibujos animados creado por Vector de Tecnología creado por upklyak - www.freepik.es