

Guía Introducción a Full Stack

Sitio: [Instituto Superior Politécnico Córdoba](#)
Curso: Programador Full Stack - TSDWAD - 2022
Libro: Guía Introducción a Full Stack

Imprimido por: Ezequiel Maximiliano GIAMPAOLI
Día: jueves, 8 septiembre 2022, 4:13

Tabla de contenidos

1. Introducción

2. Breve Reseña

3. Desarrollo Web Full Stack

4. Arquitectura de Aplicaciones Web

4.1. Arquitectura Web de 3 niveles

4.2. Elementos de la Arquitectura Web

5. Stacks

5.1. .NET + Web Api+ IIS + Sql Server + Angular

5.2. LAMP

5.3. MEAN

5.4. MEER

5.5. BFF

5.6. Otros Stacks

1. Introducción

En esta sección avanzaremos en el concepto de "Arquitectura de las aplicaciones Web", su importancia y su aplicación en el desarrollo.

Objetivos:

- Comprender la importancia de la arquitectura en el desarrollo de aplicaciones web como así también sus elementos.
- Identificar los conceptos pilares de las arquitecturas de aplicaciones web y los stacks más populares.
- Identificar los conceptos y tecnologías propias del frontend y backend, así como de la conexión entre ambas.

2. Breve Reseña

Esta breve reseña está plagada de términos técnicos. La idea es ampliar estos conceptos durante el desarrollo del documento. Para lo cual nos referiremos al glosario disponible al final del mismo.

ARPANET, The Advanced Research Projects Agency Network, fue el primer proyecto de una red **WAN** exitoso, en el año 1969. A partir de allí empieza un crecimiento vertiginoso del uso de la internet.

Pero no fue hasta 1990 que 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 (hoy estándares) que son:

- **HTML** (HyperText Markup Language). Lenguaje de marcado o etiquetado que se emplea para escribir los documentos o páginas web.
- **URL** (Universal Resource Locator). El localizador de recursos uniforme, sistema de localización o direccionamiento de los documentos web.
- **HTTP** (HyperText Transfer Protocol) El lenguaje con el que se comunica el navegador con el servidor web y el que se emplea para 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 hacen peticiones (clientes) y procesos que responden a esas peticiones (servidores). Esta comunicación es posible gracias al protocolo HTTP.

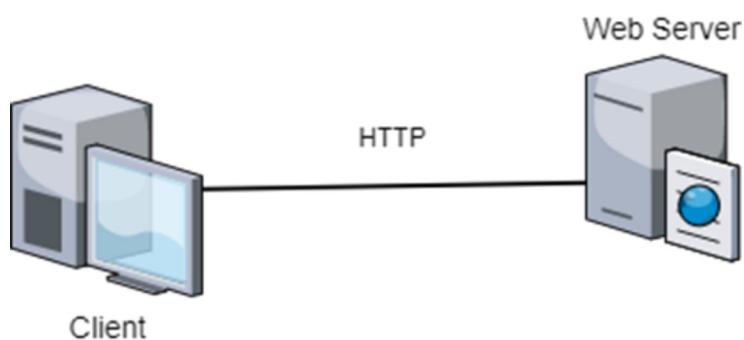


Figura: Arquitectura cliente/servidor básica

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.



A continuación, te invitamos a tomar un cafecito y mirar la conferencia de **Tim Berners-Lee** dónde cuenta en sus propias palabras cómo nació la web.

El futuro de la Web 1/2: Tim Berners-Lee en TED 2009



A continuación se puede observar la evolución de la web hasta 2012 y sus hitos más destacados de los cuales se destacan:

- En 1991 surge **HTTP** definido como “protocolo de red para sistemas de información hipertexto 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, o sea, 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.

3. Desarrollo Web Full Stack

El desarrollo web full stack permite crear **aplicaciones web dinámicas**. Esto se logra en base a los **estándares web** y utilizando tecnologías web que pueden variar dependiendo del stack de desarrollo.

Según la definición previa el desarrollo web full stack tiene por objeto la creación de aplicaciones web dinámicas pero...



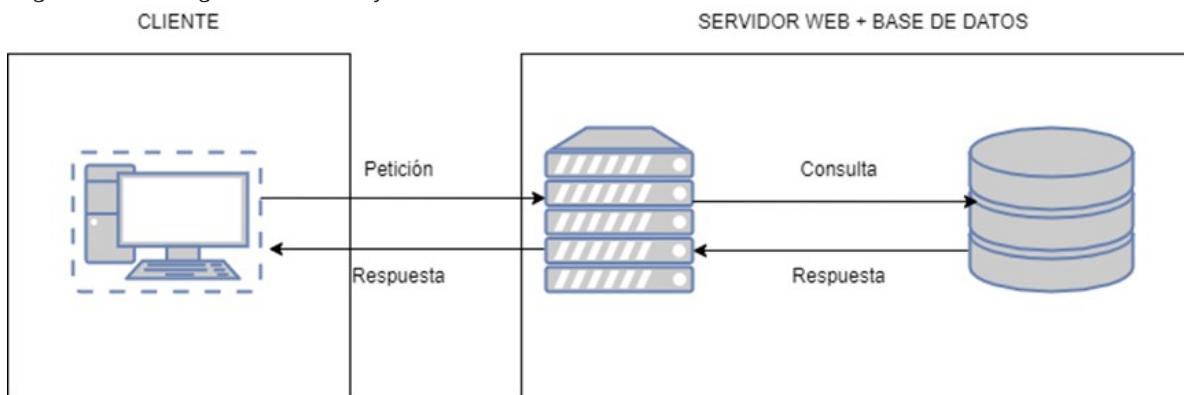
Desafío: ¿Qué entendemos por web dinámica? ¿En qué se diferencia de las web estáticas? Investiga.

4. Arquitectura de Aplicaciones Web

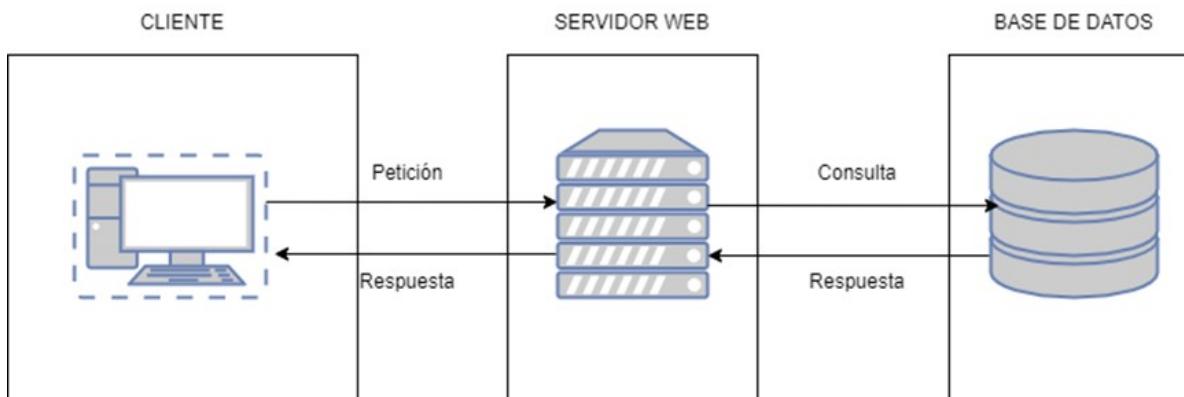
Las aplicaciones web se basan en una arquitectura cliente/servidor. Es decir que, por un lado está el cliente (navegador) y por otro lado el servidor. Existen diferentes variantes de la arquitectura básica según como se implemente.

A continuación enumeramos algunas las arquitecturas más comunes:

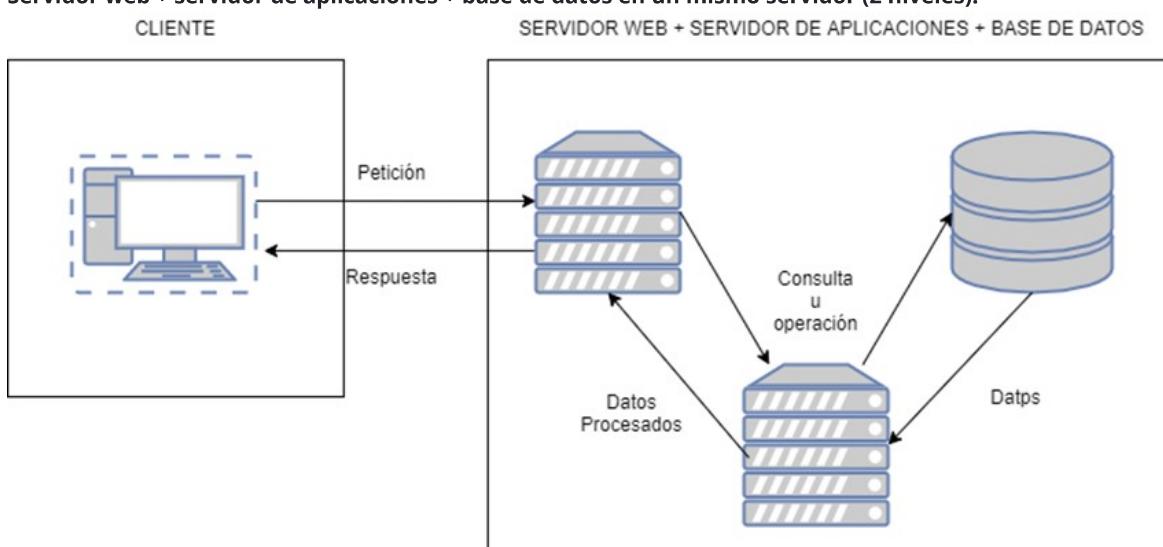
- **Servidor web + base de datos en un mismo servidor (2 niveles).** En este caso el servidor gestiona tanto la lógica de negocio como la lógica de los datos y los datos.



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



- **Servidor web + servidor de aplicaciones + base de datos en un mismo servidor (2 niveles).**



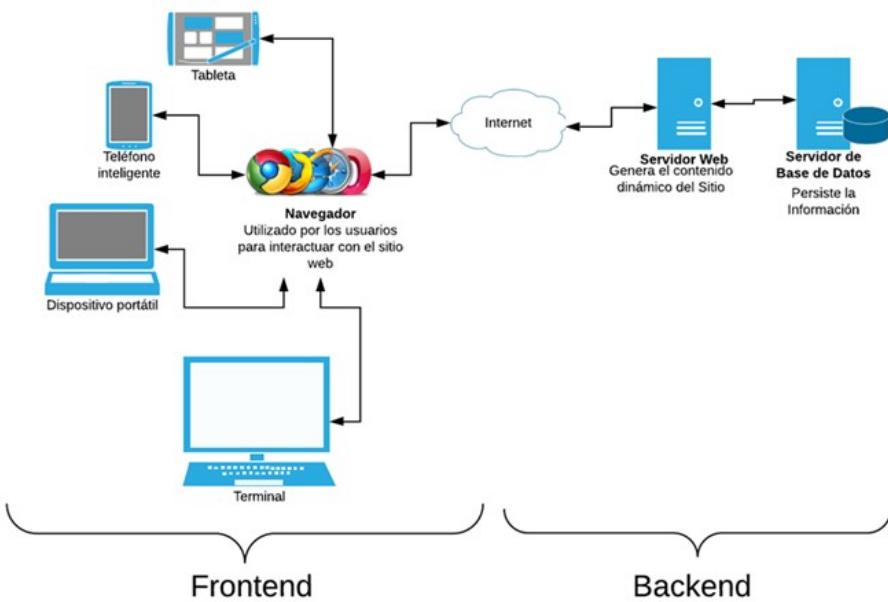
- **Servidor de aplicaciones + base de datos en un mismo servidor (3 niveles).**

- **Servidor de aplicaciones, base de datos y servicios de aplicaciones en diferentes separados en diferentes servidores (4 niveles).**

El objetivo de separar las distintas funcionalidades en distintos servidores es aumentar la escalabilidad y el rendimiento. Por ejemplo el servidor web al ofrecer servicios de http deberá tener una buena conexión a internet mientras que el servidor de base de datos requerirá tener una buena capacidad de almacenamiento y procesamiento. (Sergio Lujan Mora, Arquitectura de aplicaciones web: historia, principios básicos y clientes web)

4.1. Arquitectura Web de 3 niveles

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).



Frontend

El **Frontend** son aquellas tecnologías de desarrollo **web del lado del cliente**, es decir, las que corren en el navegador del usuario y que son básicamente tres: *HTML, CSS y JavaScript*.

El frontend **se enfoca en el usuario**, en todo con lo que puede interactuar y lo que ve mientras navega. Una buena experiencia de usuario, inmersión y usabilidad son algunos de los objetivos que busca un buen desarrollador frontend, y hoy en día hay una gran variedad de **Frameworks**, preprocesadores y librerías que ayudan en esta tarea.

Lenguajes Web Frontend

A pesar de que hay varios lenguajes que se usan en el frontend, nosotros nos basaremos en tres, **HTML, CSS y JavaScript**, aunque HTML y CSS no son lenguajes de programación, no se debe confundir lenguajes de programación como ejemplo JavaScript, ActionScript o Java, con lenguajes de marcado como HTML o lenguaje de hojas de estilo como CSS. También existen otros lenguajes frontend, como por ejemplo ActionScript, Java, Silverlight, VBScript u otros lenguajes XML, pero se usan poco en comparación con HTML, CSS y JavaScript.

HTML es un lenguaje de marcado de los contenidos de un sitio web, es decir, para designar la función de cada elemento dentro de la página: titulares, párrafos, listas, tablas, etc. Es el esqueleto de la web y la base de todo el frontend.

CSS es un lenguaje de hojas de estilo creado para controlar la presentación de la página definiendo colores, tamaños, tipos de letras, posiciones, espaciados, etc.

JavaScript es un lenguaje de programación interpretado que se encarga del comportamiento de una página web y de la interactividad con el usuario.

A parte, junto al cliente también tenemos los frameworks, las librerías, los preprocesadores, los plugins... pero todo gira alrededor de HTML, CSS y JavaScript.

Backend

El **Backend** es aquello que **se encuentra del lado del servidor** y se encarga de *interactuar con bases de datos, verificar maniobras de sesiones de usuarios, montar la página en un servidor y servir todas las vistas creadas por el desarrollador frontend*.

En este caso el número de tecnologías es mucho menos limitado, puesto que la programación backend puede alcanzar *lenguajes como PHP, Python, .NET, Java, etc.*, y las bases de datos sobre las que se trabaja pueden ser SQL, MongoDB, MySQL, entre otras.

La idea de esta abstracción es mantener separadas las diferentes partes de un sistema web o software para tener un mejor control. En pocas palabras, el objetivo es que el frontend recoja los datos y el backend los procese.

Estas dos capas que forman una aplicación web son independientes entre sí (no comparten código), pero intercambian información. Esta división permite que el acceso a las bases de datos solo se haga desde el backend y el usuario no tenga acceso al código de la aplicación, mientras que la programación del lado del cliente permite que el navegador pueda, por ejemplo, controlar dónde el usuario hace clic o acceder a sus ficheros.

Con esta separación de entornos el usuario de una aplicación web lo que hace es, por ejemplo, iniciar sesión escribiendo su usuario y contraseña en un formulario; a continuación, los datos se envían y el backend toma esta información que viene desde el HTML y busca las coincidencias de usuario en la base de datos con una serie de procesos invisibles para el usuario. En este punto, el servidor mandaría un mensaje al frontend dándole acceso (o no) a la aplicación.

Lenguajes Web Backend

Aquí encontramos unos cuantos, por ejemplo, PHP, Python, Rails, Go, C#, Java, Node JS (JavaScript) entre otros. Como vemos, mientras que para el frontend se acostumbra a trabajar solo con tres lenguajes, en el backend hay unos cuantos más. Por suerte, un desarrollador backend no necesita saberlos todos.

Quizás habéis notado que tenemos JavaScript tanto por el lado del cliente como por el lado del servidor. JavaScript se creó originalmente como lenguaje para el frontend, pero los últimos años se ha creado su lugar dentro del backend con NodeJS, un motor que interpreta JavaScript en el servidor sin necesidad de un navegador. Esto no quiere decir que el JavaScript que tenemos en el cliente tenga alguna conexión con el que se encuentra en el servidor: cada uno corre por su parte de manera independiente. El JavaScript del cliente corre en el navegador y no tiene ningún enlace ni ninguna conexión con el que hay en el servidor y no le interesa saber cómo está montada la arquitectura del servidor ni cómo se conecta a la base de datos.

Ahora se puede utilizar el mismo lenguaje en todos los contextos del desarrollo: JavaScript en el cliente de escritorio (DOM), en el cliente móvil (Cordova, React Native), en el servidor (Node.js) o en la BBDD (MongoDB). La posibilidad de trabajar frontend y backend con un mismo lenguaje desde el punto de vista del desarrollador es muy cómoda, especialmente para aquellos que trabajan ambos mundos.

En cuanto a la tecnología, las herramientas que se utilizan en el backend son: editores de código, compiladores, algunos depuradores para revisar errores y seguridad, gestores de bases de datos y algunas otras cosas.

4.2. Elementos de la Arquitectura Web

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

La infraestructura de red

Si bien es cierto que en fase de desarrollo, para probar nuestra aplicación web, no necesitaríamos esta infraestructura, una vez nuestra aplicación se instale en el hosting definitivo, será necesaria una red ethernet y todos los componentes que hacen posible la conectividad de los equipos informáticos. Con esto nos referimos a cables de red, sea utp o fibra óptica, placas de red, sea wifi o cableada, switch, routers, modem, etc. Y estos componentes físicos son necesarios tanto del lado del cliente como del servidor.

Con esto se quiere decir que no es posible implementar una aplicación web sino existe una infraestructura de red preexistente o se diseña e implementa una nueva.

Esta puede ser: internet, intranet o extranet.

Isp

Es el proveedor del servicio de internet.

Cliente Web

Es el navegador web. Ej. Chrome, Safari, Firefox, etc. El cliente se ejecuta en un hardware y hoy sabemos que puede ser desde una pc de escritorio, una notebook, o un dispositivo móvil tal como un teléfono celular o una tablet.

Pero ya no se restringe solo a estos dispositivos sino que podría ser, por ejemplo, un sistema embebido ejecutándose en una SBC (small board computer). Incluso podría no estar ejecutando un navegador convencional, como por ejemplo un reloj inteligente, o un dispositivo vinculado a una máquina de producción seriada como los surgidos de la mano del concepto de Industria 4.0.

Nombre de Dominio

Dicho de forma sencilla, el nombre de dominio (o, simplemente, "dominio") es el nombre de un sitio web. Es decir, es lo que aparece después de "@" en una dirección de correo electrónico o después de "www." en una dirección web. Si alguien te pregunta cómo encontrarte en Internet, normalmente tendrás que decirle tu nombre de dominio (https://domains.google/intl/es_es/learn/web-terms-101/).

Aquí tienes algunos ejemplos de nombres de dominio:

google.com, wikipedia.org, youtube.com

URL

Una URL (o localizador uniforme de recursos) es una dirección web completa que se utiliza para encontrar una página web específica. Mientras que el dominio es el nombre del sitio web, la URL es una dirección que remite a los usuarios a una de las páginas de ese sitio web. Cada URL contiene un nombre de dominio y otros componentes necesarios para localizar una página o un contenido concretos (https://domains.google/intl/es_es/learn/web-terms-101/).

Aquí tienes algunos ejemplos de URLs:

<http://www.google.com>

<https://es.wikipedia.org/wiki/umami>

<https://www.youtube.com/feed/trending>

Sitio web

Aunque una cosa lleve a la otra, comprar un nombre de dominio no implica tener un sitio web. El dominio es el nombre del sitio web, la URL es la forma de encontrarlo y el sitio web es lo que los usuarios ven en su pantalla y con lo que interactúan. Es decir, cuando compres un dominio, habrás adquirido el nombre de tu sitio web, pero te faltará crear el sitio web en cuestión

(https://domains.google/intl/es_es/learn/web-terms-101/).

Servidor DNS

Se ocupa de la administración del espacio de nombres de dominio. Este servidor se encarga de hacer las conversiones de nombres de dominio a direcciones IP. Cuando el cliente realiza una petición web, por ejemplo google.com, una de las primeras acciones del sistema es invocar a un servidor DNS para que le devuelva la dirección IP del/ o de alguno de los servidores de google. Por ejemplo devolverá la ip 172.217.162.14.

Servidor Web o servidor HTTP, es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente.

Contenedor de aplicaciones Web (o servidor de aplicaciones web), módulo que permite la ejecución de aplicaciones web. Por ejemplo, módulo PHP o Python del Servidor Web. Componente ASP o ASPX de IIS. Servidor o Contenedor de Aplicaciones Web Java: Tomcat, Weblogic, Websphere, JBoss, Geronimo, etc.

Servidor de Bases de Datos. Estos son contenedores de bases de datos que permiten organizar y administrar los datos que deben permanecer en un medio de almacenamiento permanente. Resuelven problemas de seguridad, mecanismos de comunicación, concurrencia, inconsistencias de los datos, respaldo, entre otros. Hay varios tipos de bases de datos, por ejemplo las relaciones que organizan los datos en forma de tablas, en filas y columnas. Otro tipos son los orientados a objetos u orientados a documentos donde el concepto de tablas se cambia por el de colección con formatos similares a “json”.

JavaScript Object Notation (JSON) es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. Es comúnmente utilizado para transmitir datos en aplicaciones web (por ejemplo: enviar algunos datos desde el servidor al cliente, así estos datos pueden ser mostrados en páginas web, o vice versa) (<https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/JSON>).

Proceso de una Petición Web 2.0

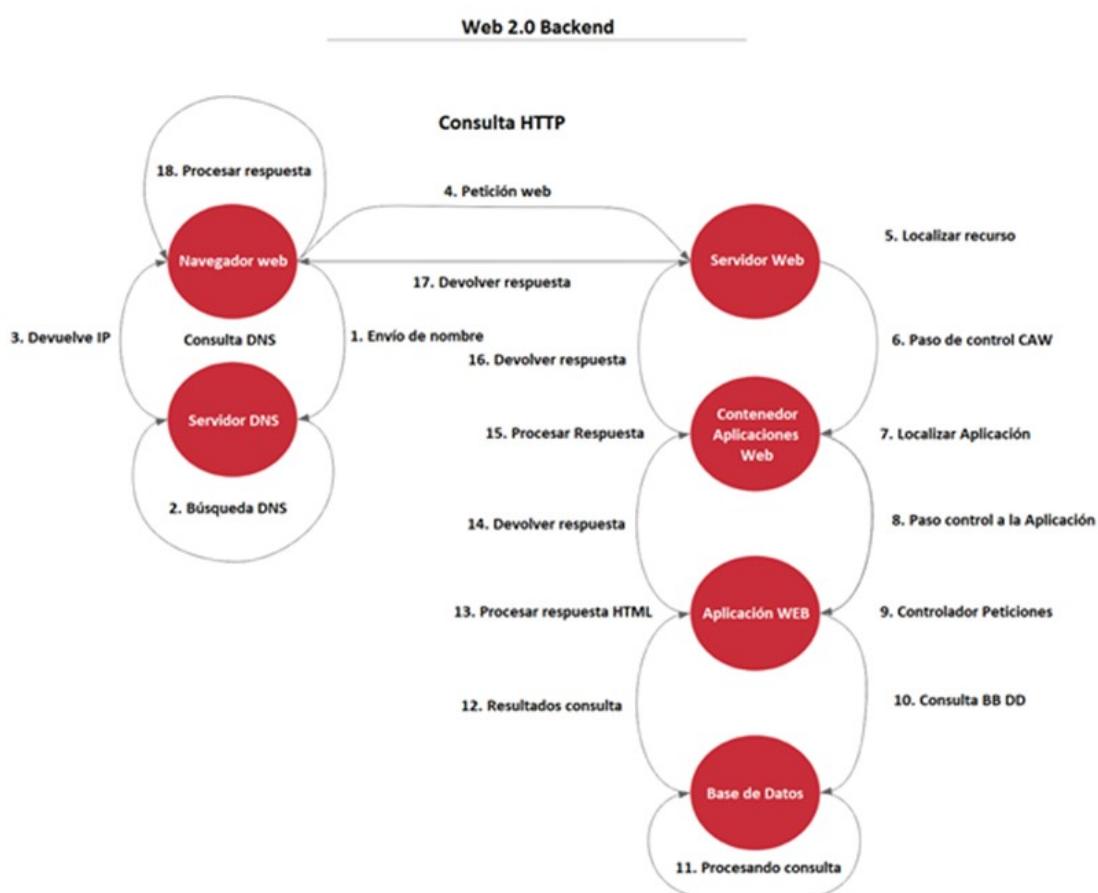


Figura: Proceso de petición web 2.0 Recuperado de: <https://cursosdedesarrollo.com/2019/11/arquitectura-web-2-0-dinamica-en-el-servidor/>

1. "Cliente Web: Solicita la resolución de nombres al servidor DNS. Por ejemplo: google.com"
2. Servidor DNS: Recibe y trata la solicitud. Una vez recibida la petición realiza las consultas necesarias para resolver y obtener la dirección IP.
3. Servidor DNS: Devuelve al navegador Web la dirección IP que corresponde al Servidor Web.
4. Cliente Web: Conecta con el servidor web mediante la dirección IP y el puerto. Realiza la petición mediante una URL (Método GET) o un formulario (Método POST). Dicha solicitud incluye: la dirección IP del servidor web, el puerto del servidor web, URL y parámetros.
5. Servidor Web: Control de Acceso, Análisis de la petición y localización del recurso. Como detecta que es el acceso a un fichero o ruta de aplicación tiene que traspasar el control al Contenedor de aplicaciones Web
6. Paso de la petición del servidor web al contenedor de aplicaciones web
7. El contenedor analiza la petición y en base a la ruta traspasa el control a la aplicación web.
8. Paso del control de la petición desde el CAW a la aplicación.
9. La aplicación recibe la petición y decide qué hacer en base a ella, es decir, elegir la funcionalidad que se encargará de gestionar esa petición, normalmente en base a la ruta, el método HTTP y los parámetros de entrada por URL. Una vez elegida ejecutará esa funcionalidad.
10. La aplicación realiza una petición SQL a la base de datos.
11. La Base de Datos recibe la petición SQL y la procesa realizando los cambios que tenga que hacer, si corresponde.
12. Una vez procesada la petición devuelve los datos a la aplicación web, normalmente un conjunto de datos. Ej. los 10 últimos clientes.
13. La aplicación web recibe estos datos y tiene que generar una salida, normalmente HTML, donde estructura el contenido de los datos devueltos por la BBDD en etiquetas HTML.
14. La aplicación web devuelve una respuesta al Contenedor de Aplicaciones Web
15. El contenedor procesa la respuesta, para controlar la ejecución de la aplicación por si esta falla.
16. El Contenedor de Aplicaciones Web devuelve el fichero al servidor web.
17. El servidor Web devuelve los datos dentro de la respuesta HTTP al navegador web.
18. Cliente Web: Presenta (renderiza) el contenido HTML resultante.
19. Repite los pasos 4-18 para obtener los ficheros relacionados: CSS, JS, Imágenes, etc."

(<https://cursosdedesarrollo.com/2019/11/arquitectura-web-2-0-dinamica-en-el-servidor/>)

En resumen, si hay algo que nos queda claro entonces, es el potencial que ha tenido y tiene la web, su capacidad de escalabilidad, por lo que todos los mercados, incluso la cultura y el arte, se manifiestan con gran libertad.

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.

5. Stacks

Lo que denominamos **stack tecnológico**, o también denominado **stack de soluciones** o **ecosistema de datos**, es un conjunto de todas las herramientas tecnológicas utilizadas para construir y ejecutar una sola aplicación, en este caso web. 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.

Me surge de manera natural pensar cuales son los stacks usan Netflix, Whatsapp, Instagram...

Uno de los stacks o pila de tecnologías más utilizado por los desarrolladores es el que se conoce por **LAMP**: Linux, Apache, MySQL y PHP. Cualquier web hecha con Wordpress, Drupal o Prestashop, por ejemplo, están hechas sobre estos cuatro pilares.

Pero se pueden hacer las variaciones que se crean convenientes, puesto que muchas de estas tecnologías son intercambiables por otras similares. Por ejemplo, NginX en lugar de Apache, PostgreSQL en lugar de MySQL o Ruby on Rails en lugar de PHP.

Otro stack muy utilizado es el llamado **MEAN**, que se compone de MongoDB, Express, Angular y NodeJS. A diferencia del conjunto anterior, esta pila de trabajo busca entregar la mayor cantidad de carga de procesamiento al cliente pero requiere una forma muy diferente de pensar las cosas.

También existe un equivalente en Microsoft que sería Windows + Microsoft IIS + .NET + SQL Server.

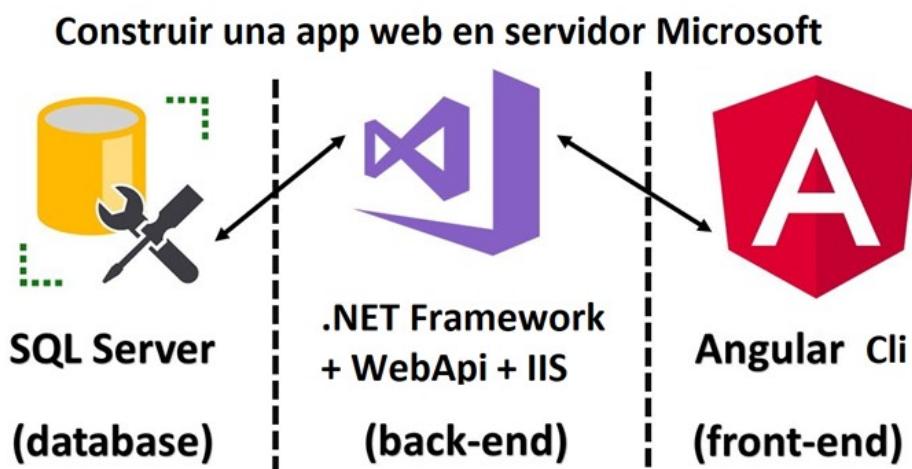
Stack populares

- **LAMP**: Linux - Apache - MySQL - PHP
- **LEMP**: Linux - Nginx - MySQL - PHP
- **MEAN**: MongoDB - Express - AngularJS - Node.js
- **Django**: Python - Django - MySQL
- **Ruby on Rails**: Ruby - SQLite - Rails
- **.NET**: .NET + WebApi + IIS - SQL Server

5.1. .NET + Web Api+ IIS + Sql Server + Angular

Microsoft tiene su propio servidor web que se llama Internet Information Server, IIS, y su servidor de base de datos, que se llama Microsoft SQL Server.

Afortunadamente nos provee con sus versiones “community”, o sea de uso libre.



Por lo tanto esta arquitectura quedaría conformada por:

- IIS: servidor web.
- Lenguaje en el backend: cualquiera de la familia .Net: C#, VB .Net, Asp.net, etc.
- Servidor de base de datos: SQL Server.
- Frontend Framework: Angular Cli.

Importante agregar que, **Microsoft .Net Core es** la plataforma de desarrollo de Microsoft más moderna con las siguientes características generales:

- de código fuente abierto
- multiplataforma: esto significa que el proyecto desarrollado se puede desplegar (deploy), o sea, poner en producción, en un servidor linux, macOS u otros sistemas operativos. No tiene la dependencia del sistema operativo Windows.
- de alto rendimiento.
- para la creación de todo tipo de aplicaciones.
- es modular, usando el sistema de paquetes **NuGet**.

Probablemente la mejor variación de la arquitectura mostrada antes sea mudar de .Net Framework a .Net Core.

Para más información sobre .Net Core, consulta los siguientes enlaces:

- <https://openwebinars.net/blog/que-es-net-core/>
- <https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0>

5.2. LAMP

LAMP es el acrónimo de Linux, Apache, MySQL y PHP, es decir las tecnologías que conforman esta plataforma de código abierto para el desarrollo del backend.

- Linux, el sistema operativo;
- Apache, el servidor web;
- MySQL/MariaDB, el motor de bases de datos;
- PHP, el lenguaje de programación.

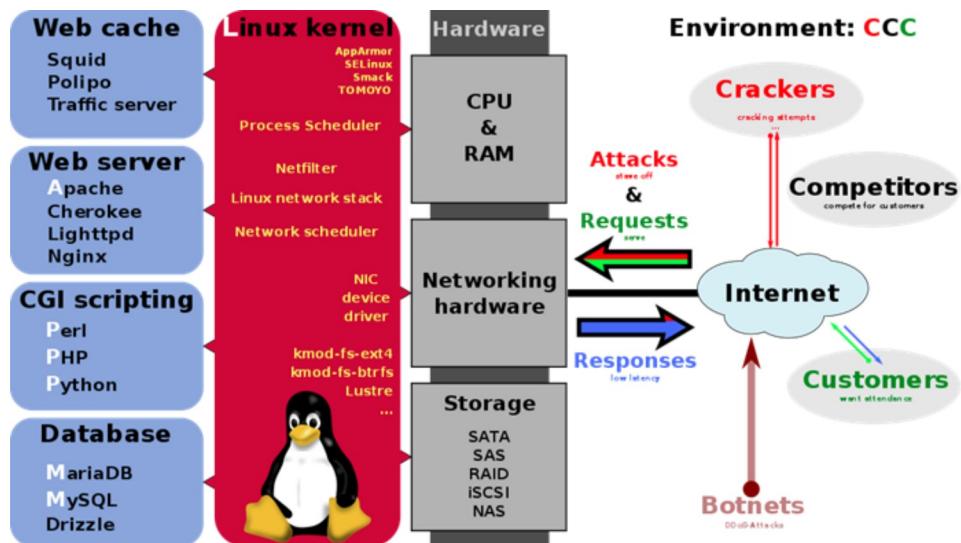


Figura: Arquitectura LAMP, <https://es.wikipedia.org/wiki/LAMP>

De este stack se pueden desprender varias variaciones como mencionamos anteriormente, incluso cambiando la marca del servidor de Apache a Nginx, los motores de bases de datos, por ejemplo de MySQL a PostgreSQL.

Esta arquitectura tiene una particularidad, en principio es la más antigua y el formato de aplicaciones predominante se denomina "**Multi Page Application o MPA**" o aplicación de múltiples páginas.

Esto hace referencia a Arquitecturas Web Clásicas en donde uno dispone de múltiples páginas HTML y cada una carga diferentes contenidos. Es decir cada página muestra su contenido y se conecta mediante links con las demás y todas son enviadas desde el servidor.

5.3. MEAN

MEAN es una plataforma de desarrollo full-stack en JavaScript, es decir, es el conjunto de tecnologías necesarias para el desarrollo de todas las capas de una aplicación web con JavaScript. Está compuesto fundamentalmente por cuatro tecnologías, MongoDB, Express, Angular y Node.js.

Cada subsistema del MEAN stack es de código abierto y de uso gratuito. A continuación se describen estas tecnologías.

- **MongoDB:** es un sistema de base de datos NoSQL, que almacena los datos en estructuras o "documentos", los cuales están definidos con la notación JSON (Notación simple de objeto tipo JavaScript), lo que permite una rápida manipulación y transferencia de los datos. La mayor característica de esta plataforma es su escalabilidad, lo que significa que puede aumentar en forma considerable la cantidad de datos que almacena sin que esto afecte su funcionamiento en general.
- **ExpressJS:** este framework está escrito en JavaScript para Node.js, es decir es un módulo de NodeJS y como tal funciona sobre esta plataforma; este módulo ofrece los métodos suficientes en JavaScript, para poder manejar las solicitudes o peticiones que se hacen por medio de los métodos del protocolo HTTP (GET, POST, etc.). También ofrece un sistema simple de enrutamiento, que dentro del mean stack es aprovechado en el back-end o en el lado del servidor.
- **AngularJS:** es un framework que facilita la manipulación del DOM ('Modelo de Objetos del Documento' o 'Modelo en Objetos para la Representación de Documentos), y por lo tanto en el mean stack es la plataforma que se usa para trabajar en el front end. Este framework permite crear una gran variedad de efectos, de una forma sencilla, reduciendo contundentemente la cantidad de código, lo que permite que sea mucho más sencillo de mantener.
- **NodeJS:** Es un entorno de ejecución o runtime para JavaScript construido con el motor de JavaScript V8 de Chrome y es la plataforma encargada del funcionamiento del servidor. Funciona totalmente con JavaScript, un lenguaje de programación que en un principio era dedicado a correr en el lado del cliente, pero su uso se ha ampliado considerablemente en todos los aspectos de un sitio web.



Figura: Arquitectura MEAN, <https://funnyfrontend.com/installar-mongodb-y-uso-de-comandos-basicos/stack-mean-esquema/>

Esta arquitectura tiene una particularidad, en principio es la más moderna y el formato de aplicaciones predominante se denomina "**Single Page Application o SPA**" o aplicación de una sola página.

Single Page Applications es una arquitectura en la que el servidor únicamente dispone de una página y carga todos los estilos y todos los formatos en el cliente. A partir de ese momento, el cliente pide únicamente datos al servidor y va renderizando diferentes componentes en el navegador que existen en la mega página.

5.4. MEER

"MERN" es el acrónimo de cuatro componentes:

- **MongoDB:** definido en MEAN
- **Express:** definido en MEAN
- **React:** React (también llamada React.js o ReactJS) es una librería Javascript de código abierto diseñada para crear interfaces de usuario (frontend) y facilita el desarrollo de todo tipo de aplicaciones.
-
- **Node.js:** definido en MEAN

Es importante agregar que React es una librería de software libre desarrollada inicialmente por Facebook pero hoy mantenida por una creciente comunidad de desarrolladores.

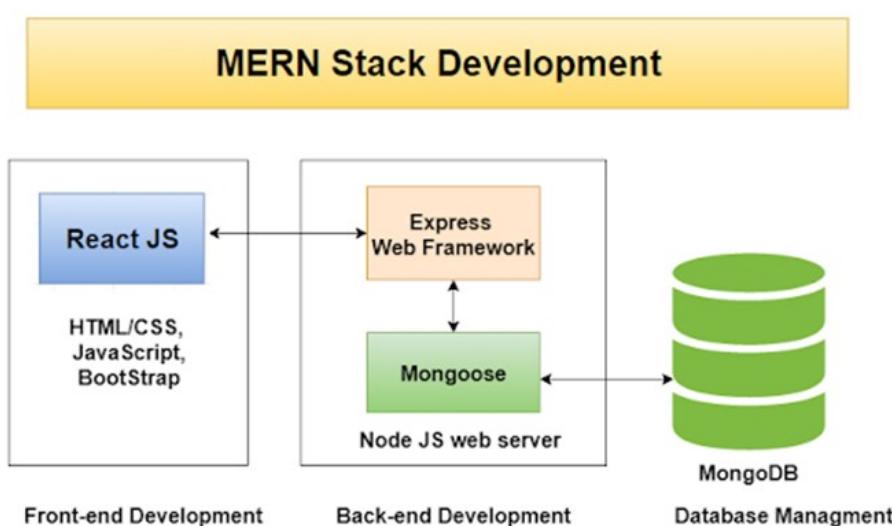


Figura: Arquitectura MEER, https://medium.com/@blockchain_simplified/what-is-mern-stack-9c867dbad302

Es importante observar que, esta arquitectura es similar a MEAN, solo que cambia Angular por React.

5.5. BFF

Se puede usar una arquitectura Backend for Frontend (BFF) para crear backends para aplicaciones web o móviles orientadas 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 **dependen** unos de otros.”^[1]

Una aplicación monolítica es una unidad cohesiva de código.^[2]

Este code pattern ayuda a los equipos 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.

Descripción

Una arquitectura de microservicio permite a los equipos iterar rápidamente y desarrollar tecnología para escalar rápidamente. La arquitectura Backend for Frontend (BFF) es un tipo de patrón creado con microservicios. El componente principal de este patrón es una aplicación que conecta el frontend de su aplicación con el backend. Este “Code Pattern BFF” lo ayudará a crear ese componente de acuerdo con las mejores prácticas de IBM.

Este patrón de código lo ayudará a:

- Crear el patrón de arquitectura Backend for Frontend (BFF)
- Generar una aplicación en Node.js, Swift o Java
- Generar una aplicación con archivos para implementar en Kubernetes, Cloud Foundry o DevOps Pipeline
- Generar una aplicación con archivos para monitoreo y seguimiento distribuido
- Conectar a servicios provisionados

También facilita el seguimiento de un modelo de programación Cloud Native que utiliza las mejores prácticas de IBM para el desarrollo de aplicaciones BFF. Verá cosas como casos de prueba, chequeo de funcionamiento y métricas en cada lenguaje de programación.

Si hace clic en **Desarrollar en IBM Cloud** en la parte superior del Code Pattern, podrá aprovisionar dinámicamente los servicios de Cloud. Esos servicios se inicializarán automáticamente en su aplicación generada. Adicione un servicio administrado de MongoDB, un servicio de Watson o pruebas automáticas en un pipeline de DevOps personalizado.

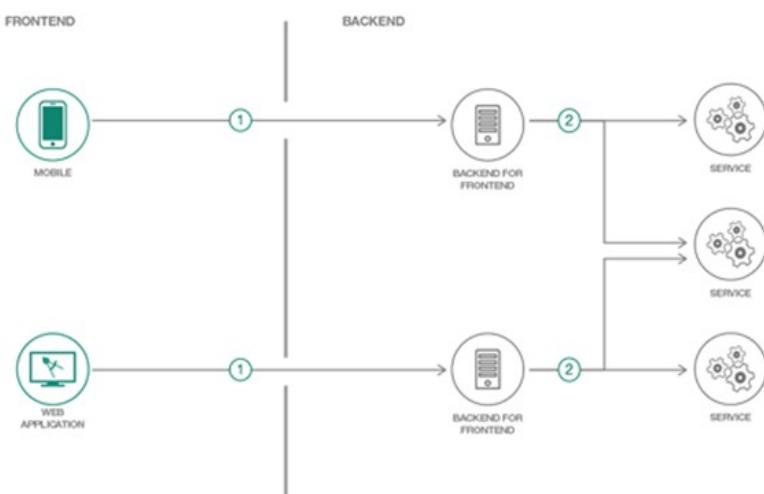


Figura: Arquitectura BFF, <https://developer.ibm.com/es/patterns/create-backend-for-frontend-application-architecture>

^[1] Modularidad, Acoplamiento y Cohesión, <https://www.disrupciontecnologica.com/acoplamiento-y-cohesion/>

[2] Sistemas monolíticos, <https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/monolitico>

5.6. Otros Stacks

Hay otros stacks, y probablemente surjan nuevos en el futuro cercano.

Una solución completa involucra no solo el stack, que en ocasiones conlleva la inversión en las correspondientes licencias, sino también las consideraciones referidas al costo de hardware, entre ellos el del servidor físico.



Desafíos: De acuerdo a los stack definidos previamente, unos proveen la construcción de aplicaciones SPA (Single Page Application) y otros MPA (Multiple Page Application). Investiga: *¿Cuál es la diferencia entre ellos? ¿Cuáles son las ventajas y desventajas de cada uno? ¿Cuándo convendrá utilizar aplicaciones SPA y cuándo aplicaciones MPA?*