5 DE SEPTIEMBRE DE 2025

UNIVERSIDAD TECNOLÓGICA DE AGUASCALIENTES

APLICACIONES WEB

CONCEPTOS GENERALES DEL DESARROLLO WEB

Presentado por:
Edwin Enoc vargas Cruz

Matrícula:
240513

Carrera:
DESM

Grado y Grupo:
4-B-6

Nombre Maestro:
Carlos Fernando Ovalle García

Introducción

La web se ha integrado profundamente en nuestra vida, transformando la forma en la que nos comunicamos, trabajamos, aprendemos y accedemos a servicios esenciales. Prácticamente cualquier actividad, como gestiones bancarias, consultas médicas, educación a distancia, entretenimiento, entre muchas otras dependen directa o indirectamente de aplicaciones y servicios alojados en internet. La tecnología nos hace entender que sus fundamentos son cada vez más relevantes, no solo para especialistas sino para cualquier persona que interactúe con dispositivos digitales.

Detrás de cada página web, plataforma o servicio en línea existe una arquitectura tecnológica compleja. Conceptos como el funcionamiento de internet, el rol de los protocolos de comunicación, la estructura cliente-servidor y los lenguajes de programación web conforman los pilares esenciales que sostienen la experiencia digital actual. Comprender estos elementos nos permite apreciar la ingeniería detrás de interfaces aparentemente simples, y al mismo tiempo, desarrollar una visión más crítica y consciente sobre el ecosistema digital que moldea nuestra realidad.

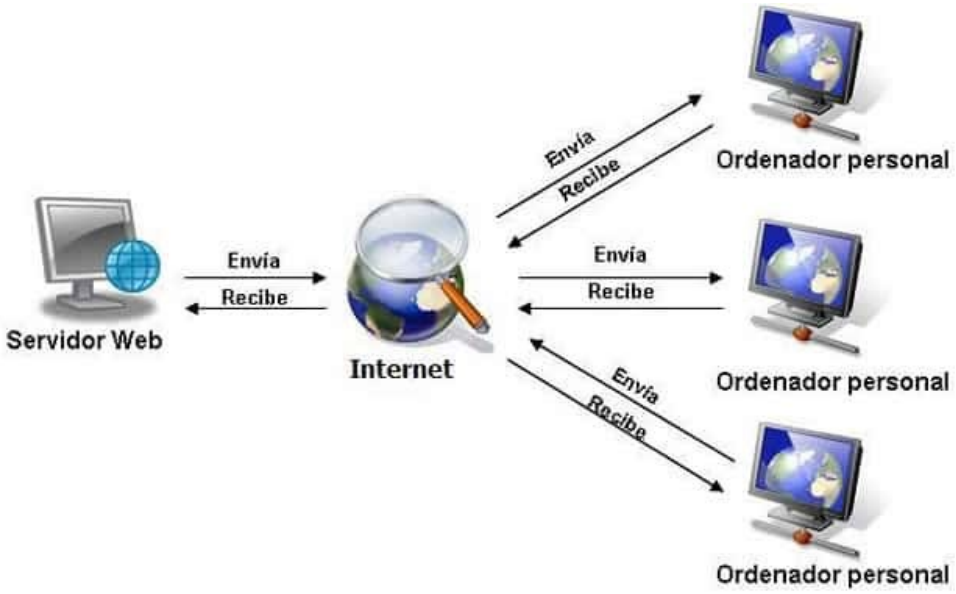
Internet: La Red de Redes (Red Internacional)

El Internet es una infraestructura global de redes de computadoras interconectadas que utilizan un conjunto común de protocolos (TCP/IP) para comunicarse. No es lo mismo que la Web; Internet es la carretera, mientras que la Web es uno de los muchos servicios (como el correo electrónico o la mensajería instantánea) que circulan por ella. Es una red descentralizada, lo que significa que no tiene un núcleo central controlador, haciéndola robusta y resistente a fallos.

Servicios Habilitados: Además de la Web (WWW), Internet permite una miríada de servicios:

- **Email (SMTP/POP/IMAP):** El protocolo SMTP (Simple Mail Transfer Protocol) se encarga del envío, mientras que POP3 e IMAP gestionan la recepción y almacenamiento de correos.
- **Transferencia de Archivos (FTP/SFTP):** FTP (File Transfer Protocol) permite mover archivos entre un cliente y un servidor. SFTP añade una capa de cifrado para mayor seguridad.
- **Mensajería Instantánea y VoIP:** Protocolos como XMPP (Extensible Messaging and Presence Protocol) y SIP (Session Initiation Protocol) permiten comunicación en tiempo real.
- **Acceso Remoto (SSH/Telnet):** SSH (Secure Shell) ofrece una terminal segura para controlar servidores remotos, mientras que Telnet hace lo mismo pero sin cifrado, por lo que está en desuso.

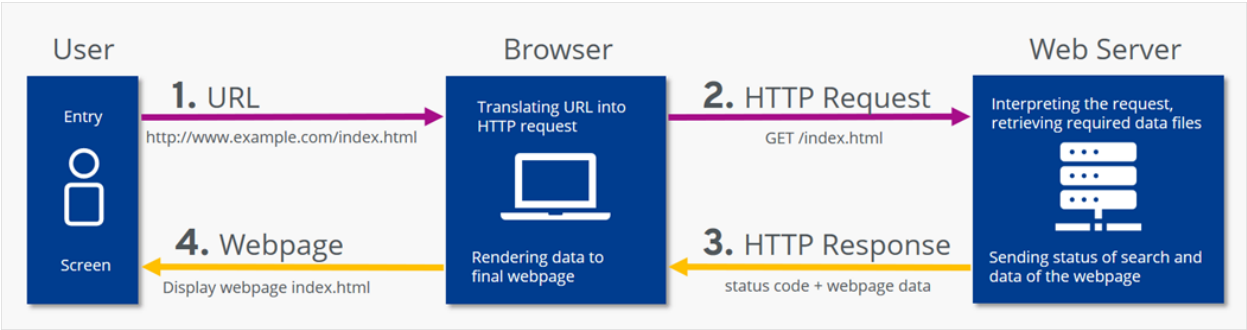
Funcionamiento del Internet



Reglas de Empaquetamiento (Protocolos): Los datos no viajan en bruto. Se fragmentan en paquetes. El protocolo TCP (Transmission Control Protocol) se encarga de dividir el mensaje en paquetes, numerarlos, asegurar su entrega (reenviando los perdidos) y reensamblarlos en el destino en el orden correcto, proporcionando una conexión confiable.

Medio Físico (Red): Los paquetes viajan a través de una vasta red de medios físicos: cables de fibra óptica transoceánicos, routers, switches, torres de telefonía móvil (inalámbrica) y satélites. Cada dispositivo conectado a Internet tiene una dirección única llamada dirección IP (Internet Protocol), que actúa como su "domicilio" en la red.

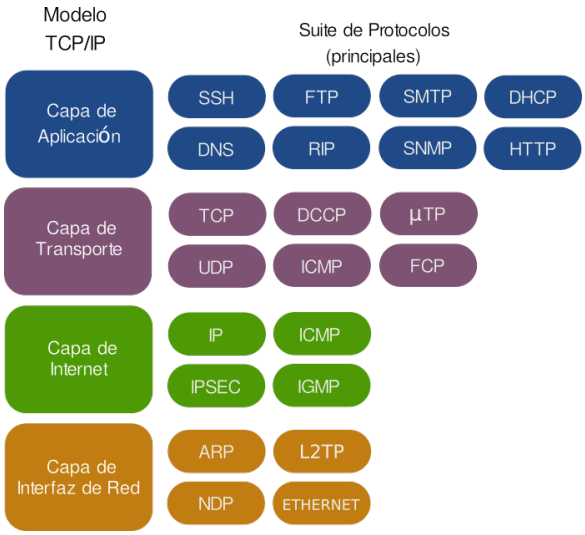
Enrutamiento (Routing): Los routers son dispositivos inteligentes que examinan la dirección IP destino de cada paquete y deciden el mejor camino para que llegue a su destino. Esta decisión se toma dinámicamente en milisegundos, basándose en tablas de enrutamiento y la congestión de la red. El protocolo IP es el responsable de este direccionamiento y enrutamiento.



Familia de Protocolos TCP/IP: El Lenguaje Universal

TCP/IP es el conjunto de reglas que permite la comunicación entre computadoras y dispositivos en Internet. Funciona como un lenguaje universal que garantiza que, sin importar el sistema operativo, la marca o el lugar, todos los equipos puedan entenderse entre sí.

Este modelo organiza la comunicación en capas, donde cada nivel cumple una función específica. La capa de acceso a la red define cómo viajan físicamente los datos; la capa de Internet se encarga de direccionarlos y encontrar la mejor ruta; la capa de transporte asegura que lleguen de manera confiable o rápida, según se use TCP o UDP; y la capa de aplicación reúne los protocolos que usamos día a día, como HTTP para páginas web, SMTP para correos o DNS para traducir direcciones.



TCP (Transmission Control Protocol): Este garantiza la entrega confiable y ordenada de los datos. Establece una conexión virtual entre emisor y receptor ("handshake" de tres vías) antes de enviar datos.

IP (Internet Protocol): Se encarga del direccionamiento lógico y el enrutamiento de los paquetes. Es un protocolo "no confiable" por sí solo (no garantiza la entrega), por eso se combina con TCP.

HTTP (HyperText Transfer Protocol): El protocolo de la capa de aplicación que define cómo se formatean y transmiten los mensajes entre clientes y servidores web. Opera sobre TCP/IP.

DNS (Domain Name System): Traduce los nombres de dominio legibles por humanos (como google.com) en direcciones IP numéricas (como 142.251.42.206) que son las que usan las máquinas para enrutar el tráfico. Es esencialmente la "agenda de teléfonos" de Internet.

Otros Protocolos de la Web:

ARP (Address Resolution Protocol): Traduce direcciones IP a direcciones MAC (identificador físico único de una tarjeta de red).

DHCP (Dynamic Host Configuration Protocol): Asigna automáticamente direcciones IP a los dispositivos que se conectan a una red.

TLS/SSL (Transport Layer Security/Secure Sockets Layer): Protocolos de cifrado que se ejecutan sobre TCP para crear una conexión segura (HTTPS = HTTP + TLS).

DNS (Domain Name System): El Directorio de Internet

El proceso de resolución DNS es crítico y a menudo involucra múltiples pasos:

1. **Caché del Navegador:** El navegador verifica primero su propia caché para ver si ya conoce la IP del dominio
2. **Caché del Sistema Operativo:** Si no está en el navegador, consulta la caché del SO.
3. **Resolver del ISP**: El router envía la consulta a un Resolver DNS (generalmente operado por el Proveedor de Internet - ISP). Este resolver tiene su propia caché.
4. **Servidores Raíz:** Si el Resolver no tiene la respuesta, consulta uno de los 13 grupos de servidores raíz DNS en el mundo. Estos no tienen la respuesta directa, pero dirigen al resolver hacia los servidores TLD (Top-Level Domain) responsables del dominio de primer nivel (como .com, .org).
5. **Servidores TLD:** Los servidores TLD dirigen al resolver hacia los servidores de nombres autoritativos del dominio específico (por ejemplo, los servidores de nombres de google.com).
6. **Servidores de Nombres Autoritativos:** Estos servidores, configurados por el dueño del dominio, finalmente devuelven la dirección IP correcta.
7. **Respuesta y Almacenamiento en Caché**: La IP viaja de vuelta por la cadena, almacenándose en cada paso para acelerar futuras consultas.

WWW (World Wide Web): El Servicio de Información

La World Wide Web (WWW) es un sistema de información creado en 1989 por Tim Berners-Lee que revolucionó la forma en que se accede y comparte datos en Internet. Se basa en el concepto de hipertexto, lo que permite enlazar documentos y navegar de uno a otro de manera sencilla mediante enlaces.

La Web funciona gracias a un modelo cliente-servidor: los navegadores (clientes) solicitan páginas a través del protocolo HTTP/HTTPS, y los servidores web responden enviando los documentos solicitados. Estos documentos pueden incluir texto, imágenes, audio, video y enlaces, lo que convierte a la Web en un espacio interactivo y multimedia..

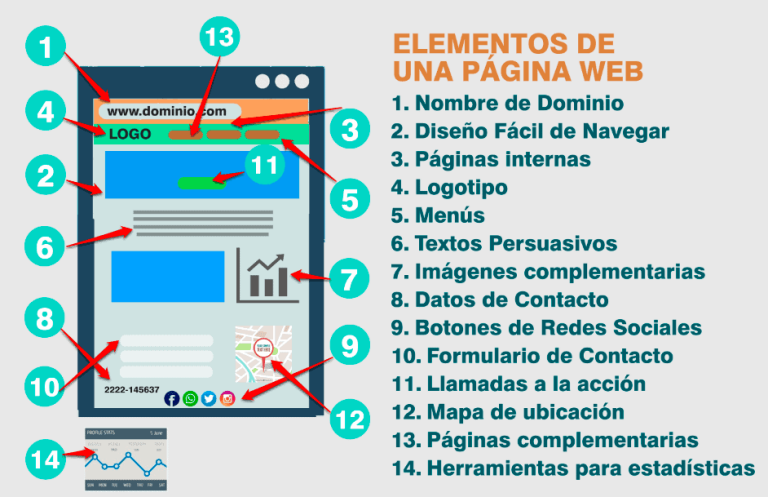
Componentes Fundamentales

URLs (Uniform Resource Locators): La dirección única de cada recurso (página, imagen, video) en la web.

HTTP (Hypertext Transfer Protocol): El protocolo para recuperar esos recursos.

HTML (HyperText Markup Language): El lenguaje de marcado para crear documentos de hipertexto.

Hipertexto: La idea revolucionaria de conectar documentos entre sí mediante hipervínculos (links), creando una "telaraña" de información interconectada.

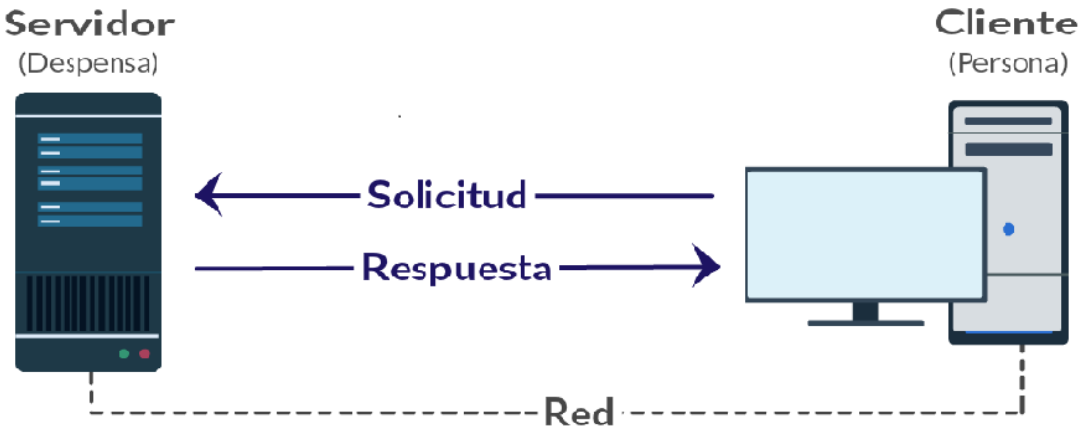


Modelo Cliente-Servidor: La Arquitectura Fundamental de la Web

Esta es la arquitectura de comunicación predominante en la web. Se conforma por los siguientes partes:

Cliente: Es el que solicita servicios o recursos. En la web, el cliente por excelencia es el navegador web (Chrome, Firefox, Safari), pero también pueden ser apps móviles o scripts.

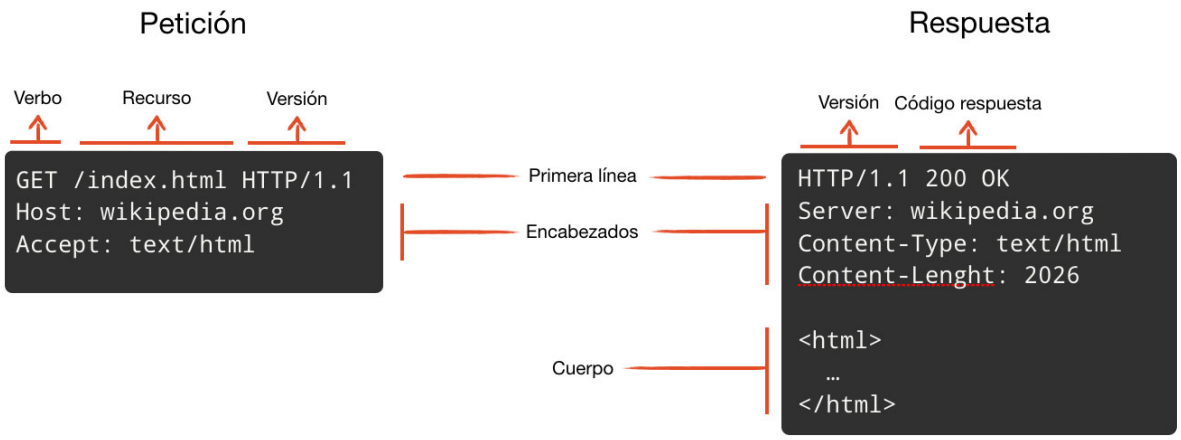
Servidor: Es el que provee el servicio o recurso solicitado. Es una computadora potente (física o virtual) que ejecuta software especializado para atender las peticiones de múltiples clientes simultáneamente. Ejemplos: Servidor Web (Apache, Nginx), Servidor de Base de Datos (MySQL, PostgreSQL).



HTTP (Hypertext Transfer Protocol): La Conversación Web

HTTP es el protocolo de comunicación que permite la transferencia de información en la World Wide Web. Es el encargado de establecer cómo los navegadores y los servidores web se comunican entre sí para enviar y recibir páginas, imágenes, videos y otros recursos.

HTTP es un protocolo sin estado, lo que significa que no guarda información de una conexión a otra; sin embargo, puede complementarse con cookies y sesiones para mantener datos de los usuarios. En su versión segura, HTTPS, añade un cifrado mediante SSL/TLS para proteger la comunicación.



Estructura de una Petición (Request):

Línea de solicitud: Contiene el método (GET, POST, etc.), la URL del recurso y la versión de HTTP.

Headers: Metadatos sobre la solicitud (e.g., User-Agent - qué navegador es, Accept - qué tipos de contenido entiende, Cookie - datos de sesión).

Body (Opcional): Donde se envían los datos, principalmente en peticiones POST o PUT.

Estructura de una Respuesta (Response):

Línea de estado: Contiene el código de estado HTTP (e.g., 200 OK, 404 Not Found, 500 Internal Server Error).

Headers: Metadatos sobre la respuesta (e.g., Content-Type - tipo de dato devuelto, Set-Cookie - instruir al navegador que guarde una cookie).

Body: El contenido solicitado en si mismo (e.g., el código HTML, una imagen, datos JSON).

Métodos HTTP: Las funciones de la Web

Los métodos de HTTP son las acciones que indican al servidor qué operación se desea realizar sobre un recurso de la Web. Cada solicitud que un navegador o aplicación envía a un servidor incluye un método que define cómo debe manejarse esa petición. Existen los siguientes:

GET: Solicita una representación de un recurso. No debe tener efectos secundarios (no debe cambiar el estado del servidor). Los datos se envían en la URL (query string). Es cacheable y queda en el historial.

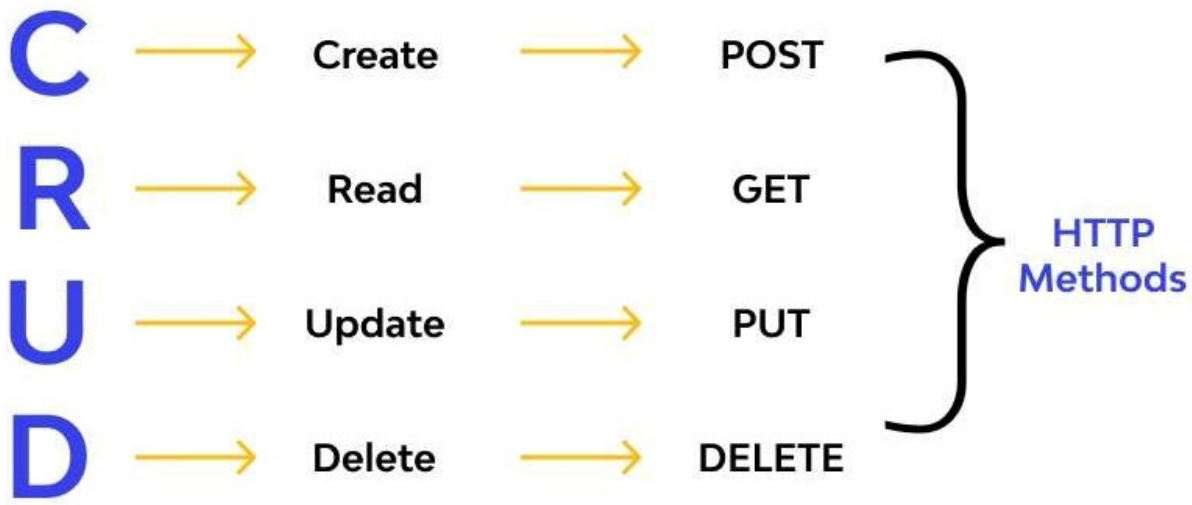
POST: Envía datos para que sean procesados por el recurso. Sí puede tener efectos secundarios (crear un nuevo usuario, hacer una compra). Los datos se envían en el cuerpo de la petición. No es cacheable y no queda en el historial. Es más seguro para datos sensibles.

PUT: Reemplaza todas las representaciones actuales del recurso de destino con los datos de la petición. Se usa para actualizar recursos completos.

DELETE: Borra el recurso especificado.

HEAD: Similar a GET, pero el servidor solo devuelve los headers, sin el body. Útil para verificar si un recurso existe o ha sido modificado sin descargarlo.

OPTIONS: Describe las opciones de comunicación para el recurso de destino (qué métodos HTTP soporta).



URL (Uniform Resource Locator): La Dirección de Todo Recurso

Una URL es la dirección única que identifica y localiza un recurso en la Web, como una página, una imagen, un archivo o un video. Funciona como el “domicilio” de un recurso en Internet, indicando a los navegadores dónde encontrarlo.

Estructura Desglosada **(scheme://[user:password@]host[:port]/path?query#fragment):**

Scheme (Esquema): Indica el protocolo a usar (e.g., http, https, ftp, mailto).

user:password@: Credenciales de autenticación (raro y poco seguro en URLs).

host: El nombre de dominio o dirección IP del servidor.

:port: El puerto TCP en el que "escucha" el servidor (por defecto es 80 para HTTP y 443 para HTTPS)

Path (Ruta): Especifica la ubicación del recurso en el servidor, similar a una ruta de archivos (e.g., /products/phones/index.html).

Query String (Cadena de Consulta): Datos adicionales para el recurso, precedidos por ? y formados por pares clave=valor separados por & (e.g., ?category=phones&sort=price).

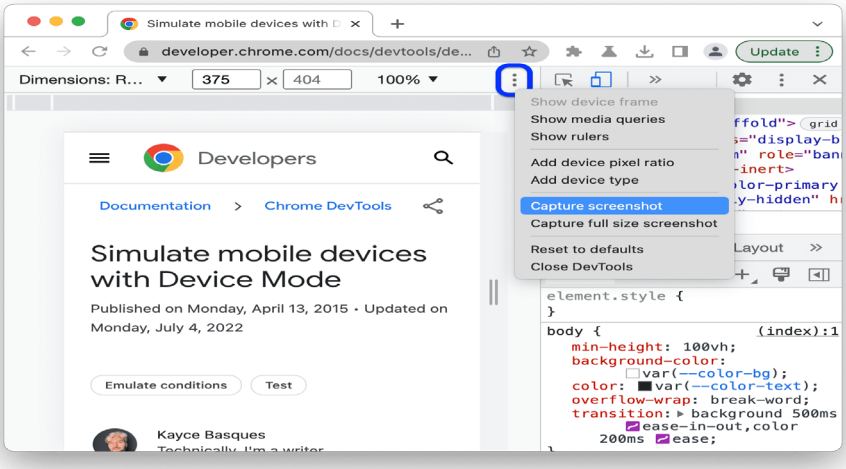
Fragment: Un identificador interno dentro del recurso mismo (e.g., #chapter-1). No se envía al servidor; lo usa el navegador para desplazarse dentro de la página.

Chrome DevTools: El Escáner de la Web

Las Herramientas de Desarrollo del navegador (F12) son indispensables. La pestaña Network (Red) permite:

- **Monitorizar en Tiempo Real:** Ver todas las peticiones HTTP que realiza la página al cargarse.
- **Analizar cada Petición:** Inspeccionar los headers de solicitud y respuesta, los métodos HTTP, los códigos de estado, los tiempos de carga y el tamaño de los recursos.

- **Simular Condiciones:** Emular dispositivos móviles, throttling de red (conexiones lentas) y deshabilitar la caché.
- **Debuggear:** Identificar recursos que fallan al cargar (404), cuellos de botella en el rendimiento y problemas de seguridad (contenido mixto HTTP/HTTPS).



2. Aplicaciones Web: De Páginas Estáticas a Software Complejo

Una aplicación web es un programa de software al que se accede a través de un navegador web y que se ejecuta principalmente en un servidor remoto.

Las aplicaciones web han evolucionado desde simples páginas HTML estáticas hasta SPAs (Single Page Applications) complejas (como Gmail o Google Maps) que ofrecen una experiencia de usuario fluida y similar a una aplicación de escritorio, todo dentro del navegador. Esta evolución ha sido impulsada por avances en JavaScript (ES6+), APIs del navegador y frameworks frontend.

Modelo de 3 Capas: Separación de Responsabilidades

Las aplicaciones web modernas suelen organizarse en una arquitectura de 3 capas que permite mantener el código ordenado, escalable y más fácil de mantener:

Capa de Presentación (Front-end):

Tecnologías: HTML, CSS, JavaScript.

Responsabilidad: Es la interfaz de usuario (UI). Se encarga de cómo se ve y se siente la aplicación, captura las interacciones del usuario (clics, formularios) y las envía a la capa de lógica. En aplicaciones modernas, gran parte de la lógica de presentación se ejecuta en el cliente usando frameworks como React, Angular o Vue.js.

Capa de Lógica de Negocio (Back-end):

Tecnologías: PHP, Python (Django, Flask), Ruby (Ruby on Rails), Java (Spring), Node.js, .NET.

Responsabilidad: Es el cerebro de la aplicación. Contiene las reglas del negocio, procesa las peticiones del front-end, realiza cálculos, valida datos y se comunica con la capa de datos. Aquí es donde reside la mayor parte de la programación server-side (del ado del servidor).

Capa de Datos:

Tecnologías: Sistemas de Gestión de Bases de Datos (SGBD) como MySQL, PostgreSQL, MongoDB, SQL Server.

Responsabilidad: Almacenar, recuperar y manipular los datos de la aplicación de manera persistente y eficiente. El back-end ejecuta consultas (SQL o NoSQL) contra esta capa.

Tipos de Aplicaciones Web

Basadas en Navegador (Tradicionales): La aplicación se entrega y ejecuta completamente en el navegador. El servidor principalmente envía datos (usualmente en formato JSON) a petición del cliente.

Basadas en Cliente (Rich Clients): Requieren la instalación de un software en el dispositivo del usuario (ej: Adobe Connect, Spotify desktop). Este cliente se comunica con servidores web remotos para sus funciones.

Aplicaciones Móviles (Apps Nativas/Híbridas): Aunque son instalables, muchas son en realidad "web views" empaquetadas (híbridas) o consumen datos de APIs RESTful proporcionadas por servidores web, por lo que su back-end es esencialmente una aplicación web.

Servicios Web (APIs): Aplicaciones sin interfaz gráfica cuyo propósito es exponer endpoints (URLs) para que otras aplicaciones (web, móviles, de escritorio) consuman datos y funcionalidades. Son la columna vertebral de la integración moderna entre sistemas. Los estilos arquitectónicos más comunes son REST y GraphQL.

Ventajas de las Aplicaciones Web

Accesibilidad Universal: Acceso desde cualquier dispositivo con navegador y conexión a internet, en cualquier lugar del mundo.

Actualización Centralizada: Al actualizar la aplicación en el servidor, todos los usuarios acceden inmediatamente a la nueva versión sin necesidad de instalar nada.

Multiplataforma: Funcionan independientemente del sistema operativo del cliente (Windows, macOS, Linux, Android, iOS).

Menos Requisitos del Lado del Cliente: No consumen espacio de almacenamiento ni potencia de procesamiento significativa en el dispositivo del usuario.

Facilidad de Distribución: La distribución se reduce a compartir una URL.

Retos y Consideraciones Críticas

La Seguridad Es el reto más grande. Amenazas comunes incluyen:

- **Inyección SQL:** Insertar código SQL malicioso a través de entradas de usuario no validadas para manipular la base de datos
- **XSS (Cross-Site Scripting):** Inyectar scripts maliciosos en páginas web vistas por otros usuarios.
- **CSRF (Cross-Site Request Forgery):** Engañar a un usuario autenticado para que ejecute acciones no deseadas en una web.
- **Manejo Inseguro de Sesiones:** Permitir que tokens de sesión sean robados o secuestrados.
- **Contramedidas:** Validación y sanitización de entradas, uso de consultas preparadas, políticas de CORS, HTTPS obligatorio, tokens CSRF, hash seguro de contraseñas (con salt, usando algoritmos como bcrypt).
- **Compatibilidad entre Navegadores**: Aunque los estándares web han mejorado, aún existen diferencias sutiles en la interpretación de CSS y JavaScript entre navegadores. Herramientas como Babel (para JS) y prefixers automáticos (para CSS) ayudan a mitigar esto.
- **Manejo de Estado (Sesiones y Cookies):** Dado que HTTP es sin estado, las aplicaciones web necesitan mecanismos para "recordar" a un usuario entre peticiones.
- **Rendimiento y Escalabilidad**: Una aplicación debe cargarse rápido y ser capaz de manejar desde unos pocos usuarios hasta millones de ellos simultáneamente. Esto implica optimización de bases de datos, caching (en el servidor, en el navegador, con CDNs), y arquitecturas escalables (balanceo de carga, microservicios).
- **SEO (Search Engine Optimization):** Para aplicaciones que dependen de tráfico orgánico, es crucial que su contenido sea fácilmente rastreable e indexable por los motores de búsqueda. Las SPAs presentan desafíos aquí que se solucionan con técnicas como Server-Side Rendering (SSR)/Renderización de contenido del lado del servidor o Static Site Generation (SSG)/ Generación Estática del Sito.

Bases de Datos: El Almacén Persistente

Una base de datos es un sistema que permite almacenar, organizar y gestionar información de manera estructurada para que pueda ser consultada y actualizada de forma eficiente. Su principal ventaja es la persistencia, es decir, los datos se mantienen guardados incluso cuando la aplicación o el dispositivo se apagan.

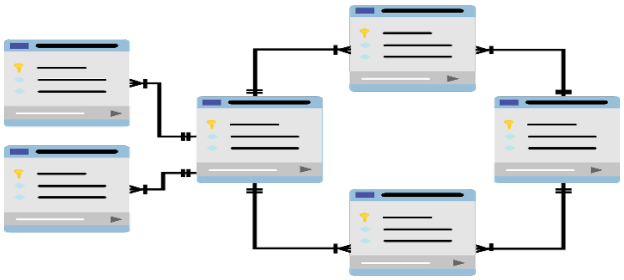
Bases de Datos Relacionales (SQL):

Estructura: Datos organizados en tablas con filas y columnas. Las tablas se relacionan entre sí mediante claves foráneas.

Lenguaje: SQL (Structured Query Language) para consultas.

Ejemplos: MySQL, PostgreSQL, SQL Server, SQLite.

Ventajas: Robustez, consistencia de datos (ACID: Atomicidad, Consistencia, Aislamiento, Durabilidad), madurez.

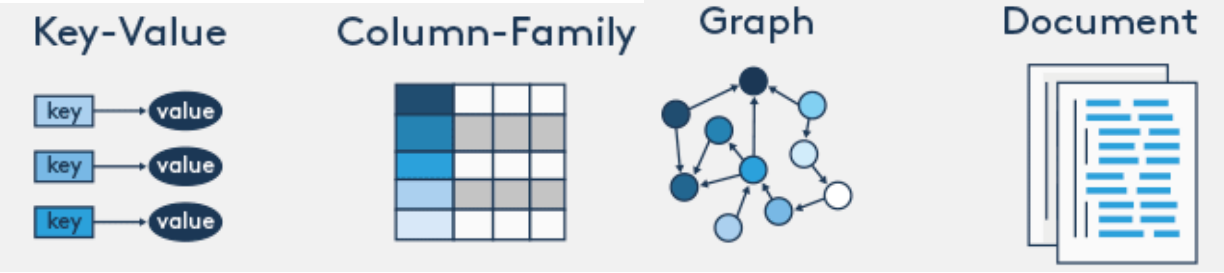


Bases de Datos No Relacionales (NoSQL):

Estructura: Más flexible, diferente tipo de persistencia. Pueden ser documentos (JSON-like), clave-valor, grafos, columnares.

Ejemplos: MongoDB (documentos), Redis (clave-valor en memoria), Cassandra (columnar).

Ventajas: Escalabilidad horizontal, flexibilidad de esquema, alto rendimiento para ciertos casos de uso (e.g., grandes volúmenes de datos no estructurados).



AJAX (Asynchronous JavaScript and XML)

Aunque el nombre incluye XML, hoy en día JSON es el formato predominante.

Cómo Funciona: En lugar de recargar toda la página para actualizar un pequeño fragmento de data, JavaScript (usando la Fetch API o anteriormente XMLHttpRequest) realiza una petición HTTP asíncrona al servidor en segundo plano. El servidor responde (generalmente con JSON), y JavaScript procesa esa respuesta y actualiza dinámicamente solo la parte necesaria del DOM.

Impacto: Esto crea una experiencia de usuario mucho más fluida y responsive, similar a una aplicación de escritorio. Es la base técnica de las Single Page Applications (SPAs).

Web Stacks Comunes: Conjuntos de Tecnologías

Un "stack" es una combinación de tecnologías usadas para construir una aplicación completa (front-end, back-end, base de datos).

LAMP: El stack tradicional. Linux (SO), Apache (servidor web), MySQL (BD), PHP (lenguaje).

MEAN/MERN: Stacks modernos basados en JavaScript. MongoDB (BD), Express.js (framework web para Node.js), Angular o React (framework front-end), Node.js (entorno de ejecución back-end).

JAMstack: Arquitectura moderna basada en JavaScript, APIs y Markup pre-renderizado. Los sitios se construyen previamente y se sirven desde una CDN, son muy rápidos y seguros. Usa herramientas como Gatsby, Next.js, o Hugo.

4. Herramientas: El Taller del Desarrollador Web

Editores de Código/IDEs (Entornos de Desarrollo Integrado):

Visual Studio Code: El editor más popular actualmente. Gratuito, extensible, con soporte excelente para debugging, control de versiones integrado y miles de extensiones.

WebStorm: Un IDE potente y pago de JetBrains, muy completo para desarrollo JavaScript.

Sublime Text: Editor ligero y muy rápido.

Características Clave: Resaltado de sintaxis, autocompletado inteligente (IntelliSense), debugging integrado, terminal integrada, control de versiones integrado (Git), soporte para múltiples lenguajes, plugins y herramientas adicionales de desarrolladores.

Control de Versiones (Git): Es absolutamente esencial. Git permite rastrear todos los cambios en el código a lo largo del tiempo, revertir a estados anteriores, crear ramas para experimentar con nuevas features y colaborar con otros desarrolladores sin pisar el trabajo de cada uno. GitHub, GitLab y Bitbucket son plataformas de hosting para repositorios Git que facilitan la colaboración y el despliegue continuo (CI/CD).

Librerías y Frameworks: Conjuntos de código preescrito que aceleran el desarrollo y promueven las mejores prácticas.

4.1 Principales Frameworks y Librerias

Front-end (Lado del Cliente):

React (Biblioteca): Desarrollada por Facebook. Se centra en la creación de interfaces de usuario mediante componentes reutilizables. Usa un DOM virtual para un rendimiento eficiente.

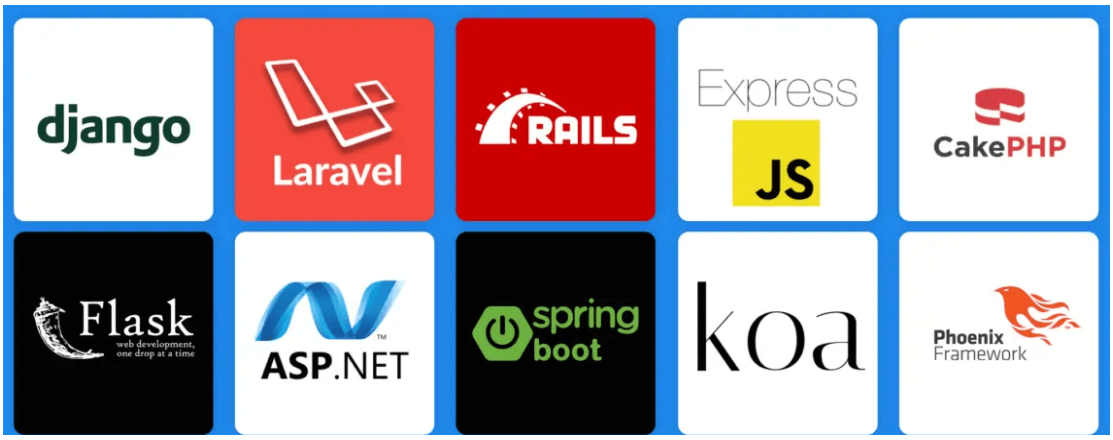
Angular (Framework): Desarrollado por Google. Un framework completo tipo "todo-en-uno" que incluye manejo de rutas, inyección de dependencias, y herramientas para construir aplicaciones complejas.

Vue.js (Framework Progresivo): Puede usarse incrementalmente, desde mejorar HTML hasta construir SPAs complejas. Es conocido por su curva de aprendizaje suave y su documentación excelente.

jQuery (Biblioteca): Fue dominante para simplificar la manipulación del DOM y AJAX, pero su uso ha disminuido con los estándares modernos de JavaScript y los frameworks.

Back-end (Lado del Servidor):

Frameworks principals: Laravel (PHP) Django (Python), Express (Node.js), Ruby on Rails (Ruby), Spring (Java).



Herramientas de Construcción y Empaquetado (Build Tools):

npm / yarn: Gestores de paquetes para JavaScript. Permiten instalar y gestionar dependencias (librerías) de un proyecto.

Webpack / Vite / Parcel: "Bundlers". Toman todos los archivos de un proyecto (JS, CSS, imágenes) y los empaquetan y optimizan para producción (minificación, tree-shaking - eliminar código no usado, transpilación con Babel para compatibilidad con navegadores antiguos), su propósito es mejorar el rendimiento y velocidad de la página web, del lado del cliente y del servidor.

5. Principios de Desarrollo de Software: Filosofías para un Código Mejor

KISS (Keep It Simple, Stupid/Mantenlo estúpidamente simple): La simplicidad debe ser una meta clave. Un diseño simple es más fácil de mantener, debuggear y entender. Se debe evitar la complejidad innecesaria y la "sobre-ingeniería".

DRY (Don't Repeat Yourself/No te repitas siempre): "Toda pieza de conocimiento debe tener una representación única, inequívoca y autoritativa dentro de un sistema." Evitar la duplicación de código o lógica reduce los errores y facilita los cambios. Si una regla de negocio cambia, solo deberías tener que modificarla en un lugar.

YAGNI (You Ain't Gonna Need It/Tu no lo vas a necesitar): No agregues funcionalidad hasta que sea estrictamente necesario. Implementar features "por si acaso" añade complejidad innecesaria, aumenta el costo de desarrollo y mantenimiento, y puede que nunca se use.

6. Formatos de Texto, Codificación y Expresiones Regulares

Codificación de Caracteres (Character Encoding): Es el mapa que traduce bytes (números) en caracteres legibles para humanos.

ASCII: El estándar original, solo soportaba 128 caracteres (inglés básico).

UTF-8: El estándar dominante en la web hoy. Es una codificación de Unicode que puede representar casi todos los caracteres de todos los idiomas del mundo. Es retrocompatible con ASCII. Siempre se debe usar UTF-8 en proyectos web modernos. Se declara en HTML con <meta charset="UTF-8"> y debe coincidir con la codificación en la que se guarda realmente el archivo en el editor.

JSON (JavaScript Object Notation) y XML (eXtensible Markup Language)

Son formatos de texto plano para estructurar e intercambiar datos en la web.

JSON: Derivado de JavaScript, es más ligero y fácil de leer y escribir para humanos y máquinas. Es el formato de facto para las APIs web modernas.

```
{
  "nombre": "Ana",
  "edad": 30,
  "hobbies": ["leer", "correr"],
  "direccion": {
    "ciudad": "Ciudad de México",
    "pais": "México"
  }
}
```

XML: Más verboso y estricto. Usa etiquetas personalizables.

```
<persona>
  <nombre>Ana</nombre>
  <edad>30</edad>
  <hobbies>
    <hobbie>leer</hobbie>
    <hobbie>correr</hobbie>
  </hobbies>
  <direccion ciudad="Ciudad de México" pais="México"/>
</persona>
```

Usos Comunes:

1. **Validación:** Verificar que un email, número de teléfono o contraseña cumple con un formato específico.
2. **Búsqueda y Reemplazo:** Encontrar todas las ocurrencias de un patrón en un texto y potencialmente reemplazarlas.
3. **Extracción de Datos:** Parsear logs o documentos para extraer información específica.

Ejemplo**: /\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b/i** es un patrón (imperfecto pero común) para buscar direcciones de email en un texto.

7. W3C y la Importancia de los Estándares Web

El World Wide Web Consortium (W3C), fundado por Tim Berners-Lee, es la organización internacional que desarrolla los estándares abiertos para la Web.

Objetivo: "Guiar la Web hacia su máximo potencial" asegurando su crecimiento a largo plazo, su interoperabilidad y su accesibilidad para todos.

Proceso: Los estándares pasan por varias etapas (Working Draft, Candidate Recommendation, Proposed Recommendation, W3C Recommendation) con amplia revisión de la comunidad.

Estándares Clave que Mantiene: HTML, CSS, XML, SVG, Web Accessibility Initiative (WAI).

Ventajas de Seguir los Estándares

Interoperabilidad: Una página web se ve y funciona de manera consistente en diferentes navegadores y dispositivos.

Accesibilidad: Los estándares incorporan pautas para que las personas con discapacidades puedan usar la web (lectores de pantalla, navegación por teclado)

Mantenibilidad: El código estándar es más limpio, más estructurado y más fácil de entender y modificar.

Rendimiento: Los navegadores pueden procesar y renderizar código estándar de manera más eficiente.

SEO: Los motores de búsqueda pueden rastrear e indexar el contenido de un sitio estándar con mucha mayor facilidad.

Escalabilidad: El código está preparado para evolucionar con la web, evitando la obsolescencia.

Desventajas de Seguir los Estándares

Complejidad inicial: Aprender y aplicar todos los estándares correctamente requiere más tiempo y esfuerzo al principio.

Soporte desigual en navegadores: No todos los navegadores interpretan los estándares exactamente igual, lo que a veces obliga a usar soluciones adicionales o "parches" para que todo funcione bien en todos lados.

Mayor tiempo de desarrollo: Hacer las cosas "como deben ser" (especialmente accesibilidad y semántica) puede tomar más tiempo que usar soluciones rápidas pero desordenadas.

Limitaciones creativas percibidas: A veces los estándares pueden sentir que limitan opciones de diseño o funcionalidad en comparación con métodos no estándar o plugins obsoletos.

Rendimiento en casos muy específicos: En situaciones extremadamente particulares, una solución hecha "a medida" podría ser ligeramente más rápida que la estándar, aunque esto es raro y usualmente no vale la pena el trade-off.

Curva de aprendizaje constante: Los estándares web evolucionan rápido (HTML5, CSS3, nuevas APIs de JS), lo que obliga a los desarrolladores a estar aprendiendo continuamente.

Conclusión

El internet nació como una herramienta especializada y se transformó en un ecosistema global indispensable, la web ha redefinido nuestra forma en que nos comunicamos, creamos y nos relacionamos. Hoy en día sostiene empresas, impulsa innovaciones, facilita educación a distancia y mantiene conectadas a personas en todo el mundo. Necesitamos comprender sus bases técnicas, desde los protocolos que permiten el intercambio de datos hasta la arquitectura que hace posibles aplicaciones cada vez más complejas.

Por ello, este conocimiento nos prepara para el futuro, como profesionistas y como personas. Permite a las personas no solo usar tecnología de manera más consciente y segura, sino también participar en su desarrollo, en su evolución y criticarla con fundamento e imaginar nuevas formas de mejorarla.

Por último, la web sigue cambiando. Surgen y mueren estándares, herramientas y paradigmas, pero los fundamentos, las bases, permanecen como base sólida sobre la que se construye el futuro digital. Entenderlos es el primer paso para cualquier persona que desee no solo habitar ese futuro, sino también darle forma.

Fuentes de información

Cloudflare. (s.f.). *¿Cómo funciona Internet?* Cloudflare. https://www.cloudflare.com/es-es/learning/network-layer/how-does-the-internet-work/

Mozilla Developer. (s.f.). *¿Cómo funciona Internet?* MDN Web Docs. https://developer.mozilla.org/es/docs/Learn_web_development/Howto/Web_mechanics/How_does_the_Internet_work

Ryte. (s.f.). *W3C*. Ryte Wiki. https://es.ryte.com/wiki/W3C

Arsys. (2025, 17 de junio). *Todo sobre la arquitectura cliente-servidor*. Arsys Blog. https://www.arsys.es/blog/todo-sobre-la-arquitectura-cliente-servidor

Mozilla Developer. (s.f.). *Descripción general de HTTP*. MDN Web Docs. https://developer.mozilla.org/es/docs/Web/HTTP/Guides/Overview

Mozilla Developer. (s.f.). *¿Qué es una URL?* MDN Web Docs. https://developer.mozilla.org/es/docs/Learn_web_development/Howto/Web_mechanics/What_is_a_URL

IBM. (s.f.). *Arquitectura de tres capas*. IBM. https://www.ibm.com/es-es/topics/three-tier-architecture

ESIC. (2024, junio). *¿Qué son las aplicaciones web?* ESIC. https://www.esic.edu/rethink/tecnologia/que-son-las-aplicaciones-web-c

Hiberus. (2023, 26 de mayo). *Tipos de bases de datos: cuáles hay y por qué es importante elegirlos bien*. Hiberus. https://www.hiberus.com/crecemos-contigo/tipos-de-bases-de-datos-cuales-hay-y-por-que-es-importante-elegirlos-bien/

freelancermap. (2023, 25 de enero). *Tech stack tecnológico: qué es y por qué es importante*. freelancermap. https://www.freelancermap.com/blog/es/tech-stack-tecnologico/

Kinsta. (2025, 11 de febrero). *Mejores herramientas de desarrollo web*. Kinsta. https://kinsta.com/es/blog/herramientas-desarrollo-web/

HubSpot. (2025, 14 de julio). *Framework de desarrollo web*. HubSpot Blog. https://blog.hubspot.es/website/framework-desarrollo-web

Ahierro. (s.f.). *Principios KISS, DRY y YAGNI*. Blog Ahierro. https://blog.ahierro.es/principios-kiss-dry-y-yagni/

Amazon Web Services. (s.f.). *La diferencia entre JSON y XML*. AWS. https://aws.amazon.com/es/compare/the-difference-between-json-xml/