Aplicaciones Web y Node.js

Conceptos fundamentales y arquitectura

Frontend | Backend | Arquitectura | Protocolos | Patrones de Diseño

Tipos de Aplicaciones

Aplicaciones sin conexión a Internet

No necesitan conexión a Internet o red local para funcionar. Suelen llamarse **aplicaciones de escritorio**.

Ejemplos: procesador de textos, lector de libros electrónicos, reproductor de música o vídeo, videojuegos instalados.

Aplicaciones con conexión a Internet

Necesitan conexión a Internet o red local para funcionar correctamente.

Aplicaciones P2P (peer-to-peer)

Todos los elementos conectados tienen el mismo "rango" y comparten información entre ellos. Base de programas de descarga como archivos torrent.

Aplicaciones cliente-servidor

Conjunto de ordenadores (clientes) se conectan a uno central (servidor) que proporciona información y servicios. Caso de videojuegos online.

Aplicaciones web

Software que se ejecuta en un servidor y se accede a través de un navegador web. El navegador actúa como interfaz entre usuario y servidor.

¿Qué es una Aplicación Web?

Definición

Una **aplicación web** es un tipo de software que se ejecuta en un servidor y se accede a través de un navegador web. El navegador actúa como **interfaz** entre el usuario y el servidor, donde se procesa la lógica de negocio, se accede a bases de datos y se genera la respuesta que el navegador presenta al usuario.

Evolución de las tecnologías web

Con el avance de las tecnologías web, el concepto de aplicación web se ha ampliado. Actualmente, se considera aplicación web a cualquier software desarrollado con tecnologías como **HTML, CSS, JavaScript, PHP**, etc., que se ejecuta en un navegador o en un entorno que lo emula.

WebView

Es un componente que permite **incrustar contenido web** dentro de una aplicación móvil nativa. Es común en aplicaciones como Facebook o Instagram, donde se cargan páginas externas sin abrir el navegador del sistema.

- Ofrece rapidez y cohesión dentro de la app
- Uso más controlado que el navegador tradicional

Tipos de Aplicaciones Web Modernas

Aplicaciones Híbridas

Desarrolladas con **tecnologías web** y empaquetadas como aplicaciones móviles usando herramientas como Ionic o Capacitor.

- Se ejecutan dentro de un WebView
- Acceso a funcionalidades del dispositivo
- Cámara, GPS, notificaciones

Aplicaciones de Escritorio

Utilizan frameworks como **Electron o Tauri** para crear aplicaciones de escritorio que funcionan como programas nativos.

- Combinan motor web con acceso al SO
- Reutilizan código base del frontend
- Mayor consumo de recursos

Ejemplo: Visual Studio Code

Progressive Web Apps

Aplicaciones web que pueden **instalarse desde el navegador** y ofrecer funcionalidades típicas de apps nativas.

- Notificaciones push
- Acceso sin conexión
- Ejecución en segundo plano

Ejemplo: Twitter Lite

Arquitectura de una Aplicación Web

Front-end

Parte que se ejecuta en el **navegador** del usuario. Muestra la interfaz visual y captura la interacción.

- HTML, CSS, JavaScript
- React, Angular, Vue
- Bootstrap, Tailwind

Back-end

Componente que se ejecuta en un **servidor** remoto. Gestiona la lógica de negocio y el acceso a datos.

- PHP, Node.js, Python
- MySQL, MongoDB, Redis
- Apache, Nginx, Express

Comunicación

Interacción mediante **peticiones HTTP/HTTPS**. La información suele enviarse en formato JSON.

- APIs REST o GraphQL
- Formato JSON, XML
- Autenticación y autorización

Secuencia básica de funcionamiento

1. Usuario accede desde navegador (cliente)
2. Cliente envía petición al servidor
3. Servidor procesa solicitud y accede a datos
4. Servidor envía respuesta (JSON, HTML)
5. Cliente interpreta respuesta y muestra información

Funcionamiento de una Aplicación Web

Proceso de Petición-Respuesta

1. Usuario accede desde **navegador**
2. Cliente envía **petición** al servidor
3. Servidor procesa y accede a **datos**
4. Servidor envía **respuesta**
5. Cliente muestra **información**

Ejemplo Práctico: Consulta de Noticias

1. Cliente envía solicitud de **inicio de sesión** al servidor web
2. Tras iniciar sesión, cliente solicita **listado de noticias**
3. Servidor web procesa petición y consulta al **servidor de bases de datos**
4. Servidor de bases de datos responde con **datos solicitados**
5. Servidor web genera **respuesta** (JSON o HTML) y la envía al cliente
6. Cliente muestra los datos y permite nuevas interacciones

Tipos de Arquitectura

Arquitectura de 2 niveles

Servidor web y servidor de bases de datos se ejecutan en la **misma máquina**.

- Solución sencilla y económica
- Adecuada para proyectos pequeños
- Limitaciones en seguridad y escalabilidad

Arquitectura de 3 niveles

Servidor web y servidor de bases de datos están **separados físicamente**.

- Mejor distribución de carga
- Mayor rendimiento y seguridad
- Modelo común en entornos empresariales

Arquitecturas Modernas

Microservicios

Funcionalidades como **servicios independientes** con despliegue y escalado separados.

- Despliegue independiente
- Escalabilidad granular
- Menor acoplamiento

Arquitectura Desacoplada

Frontend y **backend** separados, comunicándose mediante API.

- Mayor flexibilidad
- Reutilización multiplataforma
- Equipos especializados

Comparación

Elección según **escala**, **recursos** y **requisitos** del proyecto.

- Microservicios: sistemas complejos
- Desacoplada: multiplataforma
- Combinación según necesidades

Alojamiento Web (Hosting)

Servidor Propio

Desplegar la aplicación en una **máquina física** gestionada directamente por la empresa o equipo de desarrollo.

- Control total sobre el sistema
- Requiere conocimientos avanzados
- Mantenimiento continuo

Hosting Tradicional

Empresas como Ionos, Hostinger u OVH ofrecen **alojamiento web profesional** con distintas modalidades.

- Adaptado al tipo de proyecto
- Diferentes niveles técnicos
- Soporte técnico incluido

Compartido

Varios sitios web comparten un **mismo servidor** físico y sus recursos.

- Opción más económica
- Ideal para proyectos pequeños
- Rendimiento afectado por otros

VPS

Partición **virtual** dentro de un servidor físico, que funciona como un servidor independiente.

- Mejor rendimiento
- Más control y configuración
- Entorno más aislado y seguro

Dedicado

Alquilar un **servidor físico completo**, exclusivo para una única aplicación o cliente.

- Máximo rendimiento
- Control total sobre el sistema
- Requiere experiencia técnica

Plataformas Modernas de Despliegue

Servicios en la nube como **AWS, Azure** o plataformas especializadas como **Vercel y Netlify**, permiten desplegar aplicaciones de forma automatizada, con integración continua y escalado dinámico.

- **Automatización**
- **Integración Continua**
- **Escalado Dinámico**
- **Orientado a APIs**

Nombres de Dominio y URLs

¿Qué es un Nombre de Dominio?

Dirección web **legible y fácil de recordar** asociada a una dirección IP del servidor.

Proceso de Registro

1. Elegir nombre **disponible**
2. Seleccionar extensión (.com, .es, .org)
3. Registrar mediante **agente acreditado**
4. Configurar registros DNS

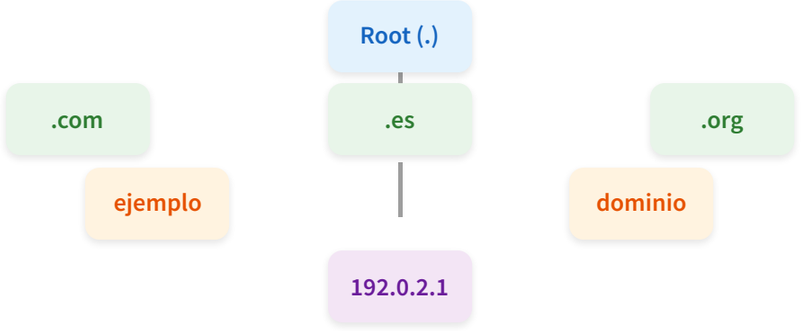
Estructura de una URL

Dirección completa para localizar un **recurso específico** en una aplicación web.

```
protocolo://subdominio.dominio.com/carpeta/pagina?param=valor
```

- **Protocolo:** Reglas de comunicación (http, https)
- **Subdominio:** Organiza servicios dentro del dominio
- **Dominio:** Identifica el servidor
- **Ruta:** Ubicación del archivo en el servidor
- **Parámetros:** Información adicional para contenidos dinámicos

Sistema de Nombres de Dominio (DNS)



Sistema que **traduce nombres de dominio a direcciones IP**, permitiendo el acceso a recursos web sin memorizar direcciones numéricas.

- Estructura jerárquica
- Base de datos distribuida
- Seguridad y redundancia
- Resolución mediante caché

Protocolos de Internet

Protocolo Base: TCP/IP

Todas las comunicaciones en Internet se basan en **TCP/IP**, compuesto por:

- **TCP**: Divide información en paquetes y asegura entrega ordenada
- **IP**: Identifica dispositivos mediante direcciones IP

HTTP

Protocolo estándar para **transferencia de páginas web**. Funciona bajo esquema petición-respuesta.

- Cliente solicita recurso
- Servidor envía recurso
- Datos viajan sin cifrar

HTTPS

Versión **segura** de HTTP. Añade capa de cifrado mediante TLS, evitando interceptación de datos.

- Cifrado de datos
- Obligatorio para información sensible
- Certificado digital requerido

SMTP

Protocolo utilizado para **enviar correos electrónicos** entre servidores.

- Comunicación servidor a servidor
- Basado en comandos texto
- Gestiona direcciones y rutas

POP3 / IMAP / FTP

Protocolos para **recibir correos** (POP3/IMAP) y **transferir archivos** (FTP).

- POP3: Descarga correos al dispositivo
- IMAP: Sincronización entre dispositivos
- FTP: Transferencia de archivos en servidor

Funcionamiento de HTTP/HTTPS

Peticiones HTTP (Requests)

Mensaje que el **cliente envía al servidor** para solicitar un recurso.

- **URL del recurso** solicitado
- **Cabeceras de petición**: información sobre el navegador, idioma, tipo de contenido aceptado
- **Cuerpo (opcional)**: datos como valores de formulario o archivos

Respuestas HTTP (Responses)

Mensaje estructurado que el **servidor envía al cliente** como respuesta a una petición.

- **Código de estado**: indica el resultado de la petición
- **Cabeceras de respuesta**: tipo de contenido, tamaño, fecha de modificación
- **Contenido**: página HTML, archivo, imagen o mensaje de error

2xx	Respuesta satisfactoria (200 OK)
3xx	Redirección (301 Moved)
4xx	Error del cliente (404 Not Found)
5xx	Error del servidor (500 Error)

Análisis de Tráfico HTTP

Las herramientas de desarrolladores de los navegadores permiten **analizar en tiempo real** el intercambio de peticiones y respuestas HTTP entre el navegador y un servidor web.

1. Abrir herramientas de desarrolladores (F12 o Ctrl+Shift+I)
2. Seleccionar la pestaña **Network** para ver el tráfico de red
3. Identificar códigos de estado, cabeceras y contenido transferido
4. Analizar tiempos de carga, tamaños y otros parámetros de rendimiento

Sistema de Nombres de Dominio (DNS)

El DNS (Domain Name System) es el sistema que **traduce nombres de dominio a direcciones IP** y viceversa. Este proceso, llamado resolución DNS, es fundamental para el funcionamiento de Internet, permitiendo a los usuarios acceder a recursos mediante nombres fáciles de recordar en lugar de direcciones numéricas.

Espacio de Nombres

Estructura **jerárquica** en forma de árbol invertido donde cada nodo contiene registros de recursos.

- Nodo raíz (.)
- Top Level Domains (TLDs): .es, .com, .org
- Dominios de segundo nivel: gob.es
- Subdominios: www.educacion.gob.es

Servidores de Nombres

Servidores encargados de **mantener y proporcionar** información del espacio de nombres o dominios.

- **Autoritativos**: almacenan información completa para una o varias zonas
  - Servidor primario: guarda versiones definitivas
  - Servidor secundario: guarda copia actualizada
- **Caché**: almacenan temporalmente respuestas

Resolvers

Rutina o componente del sistema operativo que actúa como **intermediario** entre las aplicaciones del usuario y los servidores DNS.

- Gestiona peticiones de resolución de nombres
- Almacena resultados en caché local
- Se comunica con servidor DNS configurado
- Realiza consultas recursivas o iterativas

Proceso de Resolución DNS

1. Cliente solicita resolución de **www.ejemplo.com**
2. Resolver consulta caché local
3. Si no está en caché, consulta a **servidor DNS local**
4. Servidor local consulta jerarquía DNS (raíz → TLD → autoritativo)
5. Respuesta con dirección IP se envía al cliente

Patrones de Diseño Software

Un patrón de diseño es un conjunto de **buenas prácticas y directrices estructurales** que proporcionan una solución reutilizable a problemas comunes en el desarrollo de software. Facilitan la comprensión del sistema, la colaboración entre desarrolladores y la creación de estructuras modulares y escalables.

MVC

Modelo-Vista-Controlador: divide el sistema en tres componentes con responsabilidades claramente definidas.

- **Modelo**: representa los datos y la lógica de negocio
- **Vista**: muestra información al usuario y recoge su interacción
- **Controlador**: intermediario entre modelo y vista

Ventajas

- Separación clara de responsabilidades
- Permite dividir trabajo entre perfiles profesionales

MVVM

Modelo-Vista-VistaModelo: sustituye el controlador por un ViewModel que facilita la comunicación bidireccional.

- **Modelo**: igual que en MVC, representa los datos
- **Vista**: interfaz que interactúa directamente con el usuario
- **ViewModel**: transforma datos del modelo para la vista

Ventajas

- Comunicación bidireccional (data binding)
- Ideal para Single Page Applications (SPA)

MOVE & MVP

Modelo-Operación-Vista-Evento y **Modelo-Vista-Presentador**: alternativas que reorganizan el controlador.

- **MOVE**: divide el controlador en eventos y operaciones
- **MVP**: cada vista tiene su propio presentador
- Ambos mantienen el modelo y la vista similares a MVC

Ventajas

- MOVE: mejor organización de acciones simultáneas
- MVP: facilita pruebas automatizadas

Recursos y Lenguajes

Recursos Necesarios

Para implantar una aplicación web se requiere tanto **hardware** como **software** en ambos lados de la arquitectura:

- **Lado cliente**: equipo con navegador web o aplicación móvil/escritorio
- **Lado servidor**: equipo con servidor web, servidor de aplicaciones y base de datos

Lenguajes del Entorno Cliente

Se ejecutan en el **navegador del usuario** y permiten la interacción directa con la aplicación.

HTML | CSS | JavaScript

- **Frameworks/Librerías**: React, Angular, Vue
- **Herramientas de estilo**: Sass, Bootstrap, Tailwind
- **Mejoras del código**: TypeScript (tipado estático)

Lenguajes del Entorno Servidor

Se ejecutan en el **servidor web** y permiten realizar operaciones como consultas a bases de datos, manejo de archivos o gestión de sesiones.

PHP | Node.js | Python | Java

- **Bases de datos**: MySQL, PostgreSQL, MongoDB, Redis
- **Servidores**: Apache, Nginx, Express
- **Herramientas útiles**: ORMs como Prisma o Sequelize

Conclusiones

Conceptos Clave

Las aplicaciones web modernas requieren una comprensión profunda de múltiples componentes que trabajan en conjunto: desde la **arquitectura** hasta los **protocolos** de comunicación, pasando por el **sistema DNS** y los **patrones de diseño** que estructuran el código.

- Separación clara entre front-end y back-end
- Importancia de HTTPS y seguridad
- Diversidad de plataformas y dispositivos

Importancia de una Buena Arquitectura

Una arquitectura bien diseñada es fundamental para desarrollar aplicaciones **escalables**, **mantenibles** y **seguras**. La elección del patrón de diseño adecuado, la correcta organización de los componentes y la implementación de buenas prácticas de desarrollo son factores determinantes para el éxito a largo plazo de cualquier proyecto web.

- Mejor rendimiento y escalabilidad
- Facilita el trabajo en equipo
- Simplifica mantenimiento y evolución

Tendencias Futuras en Aplicaciones Web

El desarrollo web continúa evolucionando rápidamente, con nuevas tecnologías y enfoques que transforman la forma en que creamos y experimentamos las aplicaciones en línea.

Microfrontends

Descomposición de interfaces en componentes independientes y desplegables

Integración IA

Asistentes inteligentes y generación de contenido automatizada

WebAssembly

Ejecución de código nativo en el navegador con alto rendimiento