Tabla de contenidos

1. ¿Qué se entiende por Desarrollo Web?

El desarrollo web es el arte y la ciencia de crear aplicaciones que se ejecutan en un navegador web. Involucra la planificación, el diseño, la codificación, las pruebas y el mantenimiento de sitios web. Desde una simple página estática (como un currículum en línea) hasta plataformas complejas y dinámicas (como redes sociales o tiendas de e-commerce), todo es desarrollo web.

Ejemplo: Cuando compras en una tienda online, un desarrollador web ha creado la interfaz que te permite ver los productos (Front-End) y también el sistema que procesa tu pago de forma segura y registra tu pedido (Back-End).

Tipos de Sitios Web

- **Sitios web estáticos**: Contenido fijo que no cambia sin intervención manual
- **Sitios web dinámicos**: Contenido que se genera automáticamente y puede cambiar
- **Aplicaciones web**: Programas que se ejecutan en el navegador
- **Sitios web responsivos**: Se adaptan a diferentes dispositivos y tamaños de pantalla

Categorías de Desarrollo Web

- **Desarrollo Front-End**: La parte visible e interactiva para el usuario
- **Desarrollo Back-End**: La lógica del servidor, bases de datos y APIs
- **Desarrollo Full-Stack**: Combinación de Front-End y Back-End

Industrias que Utilizan Desarrollo Web

- E-commerce y comercio electrónico
- Educación y e-learning
- Entretenimiento y medios
- Servicios financieros y bancarios
- Salud y telemedicina
- Gobierno y servicios públicos
- Redes sociales y comunicación

2. Diferencias entre Front-End, Back-End y Full-Stack

Para entenderlo mejor, sigamos el ejemplo de un formulario de registro en una red social:

- **Front-End (El Lado del Cliente):** Es todo lo que el usuario ve y con lo que interactúa. El desarrollador Front-End se encarga de que el formulario se vea bien, que los campos sean claros y que los botones respondan.
  Ejemplo: Usando HTML, crea el campo para el email y la contraseña. Con CSS, le da color al botón "Registrarse". Con JavaScript, muestra un mensaje de error si la contraseña es demasiado corta, todo esto sin recargar la página.

- **Back-End (El Lado del Servidor):** Es el motor que no se ve. Cuando haces clic en "Registrarse", los datos viajan al servidor. El desarrollador Back-End escribe el código que recibe esa información.
  Ejemplo: El código del servidor (escrito en lenguajes como Python, Java o Node.js) verifica que el email no esté ya registrado, encripta la contraseña para guardarla de forma segura y finalmente almacena los datos en una base de datos. Si todo sale bien, le envía una respuesta al Front-End diciendo "¡Usuario registrado con éxito!".

- **Full-Stack:** Es el perfil profesional que domina ambos mundos. Un desarrollador Full-Stack puede construir el formulario de registro (Front-End), programar la lógica del servidor para que funcione (Back-End) y gestionar la base de datos.

También podemos imaginar una aplicación web como un **restaurante**:

Analogía del restaurante:

```
🏪 RESTAURANTE (Aplicación Web Completa)
├── 👥 Sala del restaurante (Front-End) - Lo que ven los clientes
├── 👨‍🍳 Cocina (Back-End) - Donde se prepara todo
└── 📊 Almacén/Inventario (Base de Datos) - Donde se guardan los ingredientes
```

- **Front-End** = La experiencia del cliente (decoración, menús, meseros)
- **Back-End** = La cocina y logística (chefs, preparación, inventario)
- **Full Stack** = El gerente que entiende tanto la sala como la cocina

Responsabilidades del Desarrollador Front-End

Interfaz de Usuario (UI)

- Diseño visual y layout de la aplicación
- Elementos gráficos, colores, tipografías
- Botones, formularios, menús de navegación
- Responsive design (adaptación a diferentes dispositivos)

Experiencia de Usuario (UX)

- Facilidad de uso e intuitividad
- Navegación fluida entre páginas
- Tiempos de carga optimizados
- Accesibilidad web

Interactividad

- Efectos visuales y animaciones
- Validación de formularios en tiempo real
- Manipulación del DOM (Document Object Model)
- Respuesta a acciones del usuario

Tecnologías Front-End Principales

Lenguajes Base

Tecnologías fundamentales:

Tecnología	Propósito	Ejemplo
HTML	Estructura y contenido	`<button>Enviar</button>`

CSS	Estilos y presentación	`button { color: blue; }`
JavaScript	Lógica e interactividad	`button.addEventListener('click', function)`

Frameworks y Librerías Populares

- **React**: Creado por Facebook, basado en componentes
- **Angular**: Framework completo desarrollado por Google
- **Vue.js**: Framework progresivo, fácil de aprender
- **jQuery**: Librería para manipulación del DOM (menos usado actualmente)

Herramientas de Desarrollo

- **Preprocesadores CSS**: Sass, Less, Stylus
- **Bundlers**: Webpack, Vite, Parcel
- **Package Managers**: npm, yarn
- **Frameworks CSS**: Bootstrap, Tailwind CSS, Material-UI

Flujo de Trabajo Front-End

```
Diseño (Figma/Adobe XD) → HTML/CSS → JavaScript → Testing → Deployment
```

Ejemplo Práctico: Página de Login

HTML: Estructura

```
<!-- HTML: Estructura -->
<div class="login-form">
    <h2>Iniciar Sesión</h2>
    <form>
        <input type="email" placeholder="Email" required>
        <input type="password" placeholder="Contraseña" required>
        <button type="submit">Entrar</button>
    </form>
</div>
```

CSS: Estilos

```
/* CSS: Estilos */
.login-form {
    max-width: 400px;
    margin: 0 auto;
    padding: 20px;
    box-shadow: 0 2px 10px rgba(0,0,0,0.1);
}

button {
    background-color: #007bff;
    color: white;
    padding: 10px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}
```

JavaScript: Interactividad

```
// JavaScript: Interactividad
document.querySelector('form').addEventListener('submit', function(e) {
    e.preventDefault();
    const email = document.querySelector('input[type="email"]').value;
    const password = document.querySelector('input[type="password"]').value;

    // Validación básica
    if (!email || !password) {
        alert('Por favor completa todos los campos');
        return;
    }

    // Enviar datos al back-end
    login(email, password);
});
```

Responsabilidades del Desarrollador Back-End

Lógica del Servidor

- Procesamiento de datos y reglas de negocio
- Autenticación y autorización de usuarios
- Validación de datos del lado del servidor
- Gestión de sesiones y cookies

Base de Datos

- Diseño de esquemas de base de datos
- Operaciones CRUD (Create, Read, Update, Delete)
- Optimización de consultas
- Backup y recuperación de datos

APIs y Servicios Web

- Creación de APIs RESTful
- Integración con servicios de terceros
- Manejo de peticiones HTTP
- Documentación de APIs

Seguridad

- Protección contra vulnerabilidades comunes
- Encriptación de datos sensibles
- Configuración de servidores seguros
- Implementación de HTTPS

Tecnologías Back-End Principales

Lenguajes de Programación

- **JavaScript (Node.js)**: Permite usar JS en el servidor
- **Python**: Django, Flask, FastAPI
- **PHP**: Laravel, Symfony
- **Java**: Spring Boot
- **C#**: .NET Core
- **Ruby**: Ruby on Rails
- **Go**: Gin, Echo

Bases de Datos

- **Relacionales (SQL)**: MySQL, PostgreSQL, SQL Server
- **No Relacionales (NoSQL)**: MongoDB, Redis, Cassandra
- **En memoria**: Redis, Memcached

Servidores Web

- **Apache HTTP Server**

- **Nginx**
- **IIS (Internet Information Services)**

Servicios en la Nube

- **AWS**: Amazon Web Services
- **Google Cloud Platform (GCP)**
- **Microsoft Azure**
- **Heroku**, **Vercel**, **Netlify**

Ejemplo Práctico: API de Login

Node.js + Express: Endpoint de login

```
// Node.js + Express: Endpoint de login
const express = require('express');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const User = require('./models/User');

const app = express();

app.post('/api/login', async (req, res) => {
    try {
        const { email, password } = req.body;

        // Buscar usuario en la base de datos
        const user = await User.findOne({ email });
        if (!user) {
            return res.status(401).json({ error: 'Usuario no encontrado' });
        }

        // Verificar contraseña
        const validPassword = await bcrypt.compare(password, user.password);
        if (!validPassword) {
            return res.status(401).json({ error: 'Contraseña incorrecta' });
        }

        // Generar token JWT
        const token = jwt.sign(
            { userId: user._id, email: user.email },
            process.env.JWT_SECRET,
            { expiresIn: '24h' }
        );

        res.json({
            message: 'Login exitoso',
            token,
            user: { id: user._id, email: user.email, name: user.name }
        });

    } catch (error) {
        res.status(500).json({ error: 'Error interno del servidor' });
    }
});
```

Habilidades de un Full Stack Developer

Competencias Técnicas

- Dominio de lenguajes Front-End (HTML, CSS, JavaScript)
- Conocimiento de frameworks Front-End (React, Angular, Vue)
- Experiencia con lenguajes Back-End (Node.js, Python, etc.)
- Manejo de bases de datos (SQL y NoSQL)
- Conocimientos de DevOps básicos

Competencias Complementarias

- Comprensión de UX/UI design
- Conocimientos de arquitectura de software
- Gestión de proyectos
- Comunicación efectiva con equipos multidisciplinarios

Ventajas del Full Stack

Para el Desarrollador

- **Versatilidad profesional**: Mayor empleabilidad
- **Visión completa**: Entiende toda la aplicación
- **Autonomía**: Puede crear productos completos
- **Mejor comunicación**: Entre equipos front y back

Para las Empresas

- **Costo-efectivo**: Un desarrollador para múltiples tareas
- **Flexibilidad**: Puede cubrir diferentes necesidades
- **Startups**: Ideal para equipos pequeños
- **Prototipado rápido**: MVP (Minimum Viable Product)

Desventajas del Full Stack

Desafíos

- **Conocimiento superficial**: "Jack of all trades, master of none"
- **Mantenerse actualizado**: Muchas tecnologías que seguir
- **Complejidad**: Aplicaciones grandes requieren especialización
- **Burnout**: Responsabilidades múltiples pueden agotar

Comparación Detallada de Perfiles

5.1 Tabla Comparativa

```
Comparación de especializaciones:
```

Aspecto	Front-End	Back-End	Full Stack
Enfoque Principal	Experiencia de Usuario	Lógica de Negocio	Aplicación Completa
Tecnologías Principales	HTML, CSS, JS, React	Node.js, Python, Bases de Datos	Todo lo anterior
Responsabilidades	UI/UX, Interactividad	APIs, Seguridad, Datos	Ambas partes
Colaboración	Diseñadores, UX	DevOps, Arquitectos	Todos los equipos
Salario Promedio	$40k-80k	$50k-90k	$60k-100k
Dificultad de Aprendizaje	Media	Media-Alta	Alta

Trayectorias de Carrera

Especialización Front-End

```
Junior Frontend → Frontend → Senior Frontend → Lead Frontend → Frontend Architect
                                        ↓
                                UX/UI Designer ← → Product Manager
```

Especialización Back-End

```
Junior Backend → Backend → Senior Backend → Lead Backend → Solutions Architect
                                          ↓
                                DevOps Engineer ← → Data Engineer
```

Trayectoria Full Stack

```
Full Stack Junior → Full Stack → Senior Full Stack → Tech Lead → CTO/Founder
                                                  ↓
                                      Product Owner ← → Engineering Manager
```

3. La Triada: HTML, CSS y JavaScript

Analogía de una Casa

Imaginemos que construir una página web es como construir una casa:

Analogía de la casa:

```
🏠 PÁGINA WEB (Casa Completa)
├── 🏗️ HTML (Estructura) - Los cimientos, paredes y habitaciones
├── 🎨 CSS (Presentación) - La decoración, colores y muebles
└── ⚡ JavaScript (Comportamiento) - La electricidad, plomería y automatización
```

- **HTML** = La estructura básica y el contenido
- **CSS** = La apariencia visual y el diseño
- **JavaScript** = La funcionalidad e interactividad

Principio de Separación de Responsabilidades

Esta separación permite:

- **Mantenibilidad**: Cada lenguaje tiene su propósito específico
- **Escalabilidad**: Cambios en uno no afectan necesariamente a los otros
- **Colaboración**: Diferentes especialistas pueden trabajar en paralelo
- **Reutilización**: Componentes pueden ser reutilizados en diferentes contextos

3.1. El Lenguaje de Marcado de Hipertexto (HTML)

¿Qué es HTML?

HTML (HyperText Markup Language) es el esqueleto de una web. Usa "etiquetas" para darle estructura y significado al contenido. No le dice a la web cómo debe verse, sino qué es cada cosa.

Ejemplo concreto de código HTML:

```
<h1>Este es un título principal</h1>
<p>Este es un párrafo de texto. Contiene información importante.</p>
<a href="https://www.mindhub.la">Visita MindHub</a>
```

Aquí, `<h1>` le dice al navegador "esto es un título de nivel 1". `<p>` le dice "esto es un párrafo". Y `<a>` le dice "esto es un enlace (ancla) a otro sitio".

🎯 El Rol del Navegador

El navegador (Chrome, Firefox, etc.) es un traductor. Su trabajo es leer el código HTML, CSS y JavaScript que le envía un servidor y renderizarlo (dibujarlo) como la página visual que conocemos. Sigue las instrucciones de las etiquetas HTML para saber cómo ordenar y mostrar el contenido. Sin un navegador, una página web sería solo un archivo de texto plano.

🏢 ¿Qué es la W3C?

El **World Wide Web Consortium (W3C)** es como la "ONU" de la web. Es un organismo internacional que crea los estándares que aseguran que la web funcione para todos de la misma manera. ¿Por qué es importante? Porque gracias a sus estándares, una página web se ve y funciona de forma similar en Chrome, Firefox o Safari, y es accesible para personas con discapacidades (por ejemplo, recomendando que las imágenes tengan texto alternativo para los lectores de pantalla).

✨ Evolución del HTML hacia HTML5

HTML ha mejorado con los años. Antes de **HTML5**, para estructurar una página se usaban etiquetas genéricas como `<div id="header">` o `<div id="footer">`. Esto funcionaba, pero no tenía significado.

Ejemplo concreto: HTML5 introdujo **etiquetas semánticas** que describen su contenido:

- `<header>`: Para la cabecera (logo, menú principal).
- `<nav>`: Para la barra de navegación principal.
- `<main>`: Para el contenido principal de la página.
- `<article>`: Para un bloque de contenido independiente, como una noticia.
- `<footer>`: Para el pie de página (copyright, enlaces de contacto).

Usar estas etiquetas no solo organiza mejor el código, sino que también ayuda a los motores de búsqueda (como Google) y a las tecnologías de asistencia a entender mejor la estructura de tu página.

Características Principales de HTML

Lenguaje de Marcado

- No es un lenguaje de programación
- Utiliza etiquetas (tags) para marcar contenido
- Estructura jerárquica y anidada
- Basado en estándares del W3C

Semántica

- Las etiquetas tienen significado específico
- Mejora la accesibilidad web
- Optimiza el SEO (Search Engine Optimization)
- Facilita el mantenimiento del código

Estructura Básica de HTML

Estructura básica de una página HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mi Primera Página Web</title>
</head>
<body>
    <header>
        <h1>Bienvenido a Mi Sitio Web</h1>
        <nav>
            <ul>
                <li><a href="#inicio">Inicio</a></li>
                <li><a href="#acerca">Acerca de</a></li>
                <li><a href="#contacto">Contacto</a></li>
            </ul>
        </nav>
    </header>

    <main>
        <section id="inicio">
            <h2>Página Principal</h2>
            <p>Este es el contenido principal de la página.</p>
        </section>

        <aside>
            <h3>Barra Lateral</h3>
            <p>Contenido complementario.</p>
        </aside>
    </main>

    <footer>
        <p>© 2025 Mi Sitio Web. Todos los derechos reservados.</p>
    </footer>
</body>
</html>
```

Elementos HTML Esenciales

Elementos Estructurales

Elementos estructurales principales:

Etiqueta	Propósito	Ejemplo
`<header>`	Encabezado de página o sección	`<header><h1>Título</h1></header>`
`<nav>`	Navegación principal	`<nav><ul><li>Menu</li></ul></nav>`
`<main>`	Contenido principal	`<main><article>Contenido</article></main>`
`<section>`	Sección temática	`<section><h2>Tema</h2></section>`
`<article>`	Contenido independiente	`<article><h3>Artículo</h3></article>`
`<aside>`	Contenido lateral	`<aside>Información adicional</aside>`
`<footer>`	Pie de página	`<footer>Copyright 2025</footer>`

Elementos de Contenido

Etiqueta	Propósito	Ejemplo

`<h1>-<h6>`	Encabezados jerárquicos	`<h1>Título Principal</h1>`
`<p>`	Párrafos de texto	`<p>Este es un párrafo.</p>`
`<a>`	Enlaces (hipervínculos)	`<a href="url">Enlace</a>`
`<img>`	Imágenes	`<img src="imagen.jpg" alt="Descripción">`
`<ul>, <ol>, <li>`	Listas	`<ul><li>Elemento</li></ul>`
`<table>`	Tablas de datos	`<table><tr><td>Celda</td></tr></table>`

Elementos de Formulario

Ejemplo de formulario HTML:

```
<form action="/enviar" method="POST">
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre" required>

    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>

    <label for="mensaje">Mensaje:</label>
    <textarea id="mensaje" name="mensaje" rows="4"></textarea>

    <button type="submit">Enviar</button>
</form>
```

HTML5 y Características Modernas

APIs Nativas de HTML5

- **Geolocation API**: Obtener ubicación del usuario
- **Local Storage**: Almacenamiento local en el navegador
- **Canvas**: Dibujo y gráficos 2D/3D
- **Web Workers**: Procesamiento en segundo plano
- **Drag and Drop**: Arrastrar y soltar elementos

Elementos Multimedia

Video y audio en HTML5:

```
<!-- Video -->
<video controls width="640" height="480">
    <source src="video.mp4" type="video/mp4">
    <source src="video.webm" type="video/webm">
    Tu navegador no soporta video HTML5.
</video>

<!-- Audio -->
<audio controls>
    <source src="audio.mp3" type="audio/mpeg">
    <source src="audio.ogg" type="audio/ogg">
    Tu navegador no soporta audio HTML5.
</audio>
```

3.2. CSS

¿Qué es CSS?

CSS es el lenguaje utilizado para describir la **presentación** y el **diseño** de documentos HTML. Controla cómo se ven y se organizan los elementos en la pantalla.

Características Principales de CSS

Separación de Contenido y Presentación

- HTML se enfoca en estructura y contenido
- CSS se enfoca únicamente en la apariencia
- Permite reutilizar estilos en múltiples páginas
- Facilita el mantenimiento y actualizaciones

Cascada y Herencia

- Los estilos se aplican en cascada (de general a específico)
- Los elementos heredan propiedades de sus padres
- Especificidad determina qué estilos se aplican
- `!important` puede forzar la aplicación de un estilo

Sintaxis y Selectores CSS

Sintaxis Básica

```
selector {
    propiedad: valor;
    otra-propiedad: otro-valor;
}
```

Tipos de Selectores

Ejemplos de selectores CSS:

```css
/* Selector de elemento */
h1 {
    color: blue;
    font-size: 2em;
}

/* Selector de clase */
.mi-clase {
    background-color: yellow;
    padding: 20px;
}

/* Selector de ID */
#mi-id {
    border: 2px solid red;
    margin: 10px;
}

/* Selector descendente */
nav ul li {
    list-style: none;
    display: inline-block;
}

/* Selector de pseudo-clase */
a:hover {
    color: red;
    text-decoration: underline;
}

/* Selector de atributo */
input[type="email"] {
    border: 1px solid blue;
}
```

Propiedades CSS Fundamentales

Tipografía

Propiedades de texto:

```css
.texto {
    font-family: 'Arial', sans-serif;
    font-size: 16px;
    font-weight: bold;
    line-height: 1.5;
    text-align: center;
    text-decoration: underline;
    color: #333333;
}
```

Box Model (Modelo de Caja)

Modelo de caja CSS:

```css
.caja {
    /* Contenido */
    width: 300px;
    height: 200px;

    /* Padding (relleno interno) */
    padding: 20px;

    /* Border (borde) */
    border: 2px solid #000;

    /* Margin (margen externo) */
    margin: 15px;

    /* Background */
    background-color: lightblue;
}
```

Layout y Posicionamiento

Posicionamiento de elementos:

```css
/* Display */
.contenedor {
    display: flex; /* o grid, block, inline, etc. */
}

/* Positioning */
.posicionado {
    position: relative; /* o absolute, fixed, sticky */
    top: 10px;
    left: 20px;
    z-index: 100;
}

/* Float (método tradicional) */
.flotante {
    float: left;
    clear: both;
}
```

3.5 CSS Moderno: Flexbox y Grid

Flexbox para Layout Unidimensional

Ejemplo de Flexbox:

```css
.flex-container {
    display: flex;
    justify-content: space-between;
    align-items: center;
    flex-direction: row;
}

.flex-item {
    flex: 1; /* Crecimiento flexible */
    flex-shrink: 0; /* No se encoge */
}
```

CSS Grid para Layout Bidimensional

Ejemplo de CSS Grid:

```
.grid-container {
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
    grid-template-rows: auto 1fr auto;
    gap: 20px;
    height: 100vh;
}

.grid-item {
    grid-column: span 2;
    grid-row: 1 / 3;
}
```

Responsive Design con Media Queries

Media queries para diseño responsive:

```
/* Estilos base (mobile-first) */
.container {
    width: 100%;
    padding: 10px;
}

/* Tablet */
@media (min-width: 768px) {
    .container {
        width: 750px;
        margin: 0 auto;
    }
}

/* Desktop */
@media (min-width: 1024px) {
    .container {
        width: 1000px;
    }
}

/* Desktop grande */
@media (min-width: 1200px) {
    .container {
        width: 1170px;
    }
}
```

3.3. Javascript

¿Qué es JavaScript?

JavaScript es un lenguaje de programación que permite agregar **interactividad** y **comportamiento dinámico** a las páginas web. Es el único lenguaje de programación que funciona nativamente en los navegadores web.

Características Principales de JavaScript

Lenguaje de Programación Completo

- Variables, funciones, objetos, arrays
- Estructuras de control (if, for, while)
- Programación orientada a objetos
- Programación funcional

Interpretado y Dinámico

- No necesita compilación previa
- Tipado dinámico (las variables pueden cambiar de tipo)
- Ejecución en tiempo real
- Flexibilidad para prototipado rápido

Event-Driven (Basado en Eventos)

- Responde a acciones del usuario (clicks, teclas, movimiento)
- Maneja eventos del navegador (carga de página, resize)
- Programación asíncrona con callbacks y promises

4.3 Sintaxis Básica de JavaScript

Variables y Tipos de Datos

Declaración de variables:

```
// Declaración de variables
let nombre = "Juan";           // String
const edad = 25;               // Number
var esEstudiante = true;       // Boolean
let hobbies = ["leer", "nadar"]; // Array
let persona = {                // Object
    nombre: "Ana",
    edad: 30
};

// Null y undefined
let valorNulo = null;
let valorIndefinido;
```

Funciones

Diferentes formas de declarar funciones:

```
// Función tradicional
function saludar(nombre) {
    return "Hola, " + nombre + "!";
}

// Función arrow (ES6+)
const saludarModerno = (nombre) => {
    return `Hola, ${nombre}!`; // Template literals
};

// Función arrow simplificada
const sumar = (a, b) => a + b;

// Llamada a funciones
console.log(saludar("María"));
console.log(sumar(5, 3));
```

Estructuras de Control

Condicionales y bucles:

```
// Condicionales
if (edad >= 18) {
    console.log("Es mayor de edad");
} else if (edad >= 13) {
    console.log("Es adolescente");
} else {
    console.log("Es niño");
}

// Switch
switch (dia) {
    case 1:
        console.log("Lunes");
        break;
    case 2:
        console.log("Martes");
        break;
    default:
        console.log("Otro día");
}

// Bucles
for (let i = 0; i < 5; i++) {
    console.log("Iteración: " + i);
}

// Bucle for...of (arrays)
for (let hobby of hobbies) {
    console.log("Hobby: " + hobby);
}
```

Manipulación del DOM

Selección de Elementos

Diferentes formas de seleccionar elementos:

```
// Seleccionar por ID
const titulo = document.getElementById('mi-titulo');

// Seleccionar por clase
const botones = document.getElementsByClassName('boton');

// Seleccionar por etiqueta
const parrafos = document.getElementsByTagName('p');

// Selectores CSS (modernos)
const primerBoton = document.querySelector('.boton');
const todosBotones = document.querySelectorAll('.boton');
```

Modificación de Contenido y Atributos

Manipulación de elementos HTML:

```
// Cambiar contenido de texto
titulo.textContent = "Nuevo título";
titulo.innerHTML = "<strong>Título en negrita</strong>";

// Modificar atributos
const imagen = document.querySelector('img');
imagen.src = "nueva-imagen.jpg";
imagen.alt = "Nueva descripción";

// Trabajar con clases CSS
titulo.classList.add('destacado');
titulo.classList.remove('oculto');
titulo.classList.toggle('activo');

// Modificar estilos directamente
titulo.style.color = "red";
titulo.style.fontSize = "24px";
```

Manejo de Eventos

Event listeners para interactividad:

```
// Event listener básico
const boton = document.querySelector('#mi-boton');
boton.addEventListener('click', function() {
    alert('¡Botón clickeado!');
});

// Event listener con arrow function
boton.addEventListener('click', () => {
    console.log('Click registrado');
});

// Event listener con función externa
function manejarClick(evento) {
    console.log('Elemento clickeado:', evento.target);
    evento.preventDefault(); // Prevenir comportamiento por defecto
}

boton.addEventListener('click', manejarClick);

// Múltiples tipos de eventos
const input = document.querySelector('#mi-input');
input.addEventListener('focus', () => console.log('Input enfocado'));
input.addEventListener('blur', () => console.log('Input desenfocado'));
input.addEventListener('change', () => console.log('Valor cambiado'));
```

Programación Asíncrona

Promises

Manejo de operaciones asíncronas:

```
function obtenerDatos() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const exito = Math.random() > 0.5;
            if (exito) {
                resolve("Datos obtenidos exitosamente");
            } else {
                reject("Error al obtener datos");
            }
        }, 2000);
    });
}

// Usar promise con .then()
obtenerDatos()
    .then(resultado => console.log(resultado))
    .catch(error => console.error(error));
```

Async/Await (ES2017)

Sintaxis moderna para código asíncrono:

```
async function obtenerYProcesarDatos() {
    try {
        const datos = await obtenerDatos();
        console.log("Datos recibidos:", datos);

        const datosProcesados = await procesarDatos(datos);
        console.log("Datos procesados:", datosProcesados);

        return datosProcesados;
    } catch (error) {
        console.error("Error:", error);
    }
}

obtenerYProcesarDatos();
```

4. Importancia de dominar los fundamentos

Base Sólida para Frameworks

- **React**: Requiere sólido conocimiento de JavaScript ES6+
- **Angular**: Necesita comprensión profunda de TypeScript y CSS
- **Vue.js**: Se beneficia del entendimiento de HTML, CSS y JS vanilla

Debugging y Troubleshooting

- Los problemas complejos a menudo se resuelven a nivel fundamental
- Entender cómo funciona el navegador internamente
- Optimización de rendimiento requiere conocimiento profundo
- Mantenimiento de código legacy

Adaptabilidad Tecnológica

- Los frameworks cambian, los fundamentos permanecen
- Facilita el aprendizaje de nuevas tecnologías
- Permite tomar decisiones técnicas informadas
- Comprensión de trade-offs y limitaciones

5. El entorno de desarrollo

¿Por qué es Crucial un Buen Entorno?

Un **entorno de desarrollo** bien configurado es como tener las herramientas adecuadas para cualquier trabajo especializado. Determina:

- **Productividad**: Menos tiempo configurando, más tiempo creando
- **Calidad del código**: Detección automática de errores y mejores prácticas
- **Colaboración**: Estándares consistentes en equipos de trabajo
- **Aprendizaje**: Feedback inmediato y recursos de ayuda integrados

Componentes de un Entorno Profesional

```
🛠️ ENTORNO DE DESARROLLO WEB
├── 💻 Editor de Código (VS Code)
├── 🔧 Extensiones y Plugins
├── 🌐 Navegador con DevTools
├── 📁 Organización de Archivos
├── 🎨 Herramientas de Diseño
└── 🔄 Sistema de Control de Versiones (Git)
```

5.1. Instalar Visual Studio Code

¿Por qué Visual Studio Code?

Visual Studio Code (VS Code) es actualmente el editor más popular para desarrollo web por varias razones:

Ventajas Principales

- **Gratuito y Open Source**: Sin costo, desarrollado por Microsoft
- **Multiplataforma**: Windows, macOS, Linux
- **Extensible**: Miles de extensiones disponibles
- **Integración Git**: Control de versiones integrado
- **IntelliSense**: Autocompletado inteligente
- **Debugging integrado**: Debug de JavaScript y otros lenguajes
- **Terminal integrado**: No necesitas cambiar de aplicación

Comparación con Otros Editores

Editor	Pros	Contras	Ideal para
VS Code	Gratuito, extensible, rápido	Consume RAM	Principiantes y profesionales
Sublime Text	Muy rápido, ligero	De pago, menos extensiones	Edición rápida
Atom	Muy personalizable	Lento, discontinuado	Ya no recomendado
WebStorm	Muy potente, IDE completo	Caro, pesado	Proyectos grandes

Instalación de Visual Studio Code

Pasos de Instalación

1. **Descargar**:
   - Visita code.visualstudio.com
   - Selecciona tu sistema operativo
   - Descarga la versión estable
2. **Instalar**:
   - **Windows**: Ejecutar el .exe descargado
   - **macOS**: Arrastrar VS Code a la carpeta Aplicaciones
   - **Linux**: Usar el gestor de paquetes o descargar .deb/.rpm
3. **Configuración inicial**:
   - Elegir tema (Dark+ es popular para programación)
   - Sincronizar configuración con cuenta de Microsoft/GitHub (opcional)
   - Configurar idioma (español disponible)

Primera Configuración

settings.json - Configuraciones básicas recomendadas

```
{
    "editor.fontSize": 14,
    "editor.tabSize": 2,
    "editor.wordWrap": "on",
    "editor.minimap.enabled": true,
    "editor.formatOnSave": true,
    "editor.codeActionsOnSave": {
        "source.fixAll": true
    },
    "files.autoSave": "onFocusChange",
    "workbench.colorTheme": "Dark+ (default dark)",
    "terminal.integrated.fontSize": 12,
    "emmet.includeLanguages": {
        "javascript": "javascriptreact"
    }
}
```

Interfaz de VS Code

Explorar la Interfaz

```
📋 INTERFAZ DE VS CODE
├── 🗂️ Explorador de archivos (Ctrl+Shift+E)
├── 🔍 Búsqueda global (Ctrl+Shift+F)
├── 🌿 Control de versiones Git (Ctrl+Shift+G)
├── 🐛 Debug y ejecución (Ctrl+Shift+D)
├── 🧩 Extensiones (Ctrl+Shift+X)
├── 📝 Editor principal (centro)
├── 🖥️ Terminal integrado (Ctrl+`)
└── 🔗 Barra de estado (inferior)
```

Atajos de Teclado Esenciales

Acción	Windows/Linux	macOS	Descripción
Paleta de comandos	`Ctrl+Shift+P`	`Cmd+Shift+P`	Acceso a todas las funciones

Acción	Windows/Linux	macOS	Descripción
Abrir archivo rápido	`Ctrl+P`	`Cmd+P`	Buscar y abrir archivos
Nuevo archivo	`Ctrl+N`	`Cmd+N`	Crear archivo nuevo
Guardar	`Ctrl+S`	`Cmd+S`	Guardar archivo actual
Buscar en archivo	`Ctrl+F`	`Cmd+F`	Buscar texto en archivo
Reemplazar	`Ctrl+H`	`Cmd+Option+F`	Buscar y reemplazar
Comentar línea	`Ctrl+/`	`Cmd+/`	Comentar/descomentar
Duplicar línea	`Shift+Alt+↓`	`Shift+Option+↓`	Duplicar línea actual
Mover línea	`Alt+↑/↓`	`Option+↑/↓`	Mover línea arriba/abajo

Extensiones Esenciales para Desarrollo Web

Extensiones Fundamentales

Live Server

```
Nombre: Live Server
ID: ritwickdey.liveserver
Función: Servidor local con recarga automática
```

¿Por qué es esencial?

- Servidor web local instantáneo
- Recarga automática al guardar cambios
- Evita problemas de CORS en desarrollo
- Preview inmediato de cambios

Cómo usar:

1. Click derecho en archivo HTML
2. Seleccionar "Open with Live Server"
3. Se abre automáticamente en el navegador
4. Los cambios se reflejan inmediatamente

Prettier - Code Formatter

```
Nombre: Prettier - Code formatter
ID: esbenp.prettier-vscode
Función: Formateo automático de código
```

Configuración recomendada:

```
{
    "prettier.singleQuote": true,
    "prettier.semi": true,
    "prettier.tabWidth": 2,
    "prettier.trailingComma": "es5",
    "prettier.printWidth": 80
}
```

Auto Rename Tag

```
Nombre: Auto Rename Tag
ID: formulahendry.auto-rename-tag
Función: Renombra automáticamente etiquetas HTML emparejadas
```

Bracket Pair Colorizer 2

```
Nombre: Bracket Pair Colorizer 2
ID: coenraads.bracket-pair-colorizer-2
Función: Colorea pares de corchetes/paréntesis
```

HTML CSS Support

```
Nombre: HTML CSS Support
ID: ecmel.vscode-html-css
Función: Autocompletado CSS en archivos HTML
```

Extensiones para HTML/CSS

Emmet (Incorporado)

Emmet viene integrado en VS Code y acelera enormemente la escritura de HTML/CSS:

Ejemplo de Emmet:

```
<!-- Escribir: div.container>ul>li*5>a -->
<!-- Presionar Tab, genera: -->
<div class="container">
    <ul>
        <li><a href=""></a></li>
        <li><a href=""></a></li>
        <li><a href=""></a></li>
        <li><a href=""></a></li>
        <li><a href=""></a></li>
    </ul>
</div>
```

Comandos Emmet más útiles:

```
html:5                  <!-- Estructura HTML5 básica -->
div.clase#id            <!-- <div class="clase" id="id"></div> -->
ul>li*3                 <!-- Lista con 3 elementos -->
img[src alt]            <!-- <img src="" alt=""> -->
a{Click me}             <!-- <a href="">Click me</a> -->
div.row>div.col*4       <!-- Bootstrap grid rápido -->
```

CSS Peek

```
Nombre: CSS Peek
ID: pranaygp.vscode-css-peek
Función: Ver definiciones CSS desde HTML
```

Extensiones para JavaScript

ESLint

```
Nombre: ESLint
ID: dbaeumer.vscode-eslint
Función: Linting para JavaScript
```

JavaScript (ES6) Code Snippets

```
Nombre: JavaScript (ES6) code snippets
ID: xabikos.javascriptsnippets
Función: Snippets para JavaScript moderno
```

Snippets útiles:

```
// Escribir 'clg' + Tab
console.log();

// Escribir 'fre' + Tab
for (const item of object) {

}

// Escribir 'afn' + Tab
const name = (params) => {

}
```

Extensiones de Productividad

Material Icon Theme

```
Nombre: Material Icon Theme
ID: pkief.material-icon-theme
Función: Iconos para diferentes tipos de archivos
```

GitLens

```
Nombre: GitLens — Git supercharged
ID: eamodio.gitlens
Función: Potencia las capacidades Git de VS Code
```

TODO Highlight

```
Nombre: TODO Highlight
ID: wayou.vscode-todo-highlight
Función: Resalta comentarios TODO, FIXME, etc.
```

Better Comments

```
Nombre: Better Comments
ID: aaron-bond.better-comments
Función: Mejora la visualización de comentarios
```

```
// ! Comentario de alerta en rojo
// ? Comentario de pregunta en azul
// TODO: Comentario de tarea en naranja
// * Comentario destacado en verde
```

5.2. Herramientas de Desarrollo del Navegador

¿Qué son las DevTools?

Las **Developer Tools** (DevTools) son herramientas integradas en los navegadores web que permiten a los desarrolladores inspeccionar, debuggear y optimizar sitios web.

4.2 Acceso a DevTools

Métodos de Acceso

- **F12**: Atajo universal en la mayoría de navegadores
- **Ctrl+Shift+I**: Alternativa común
- **Click derecho → "Inspeccionar elemento"**: Inspección directa
- **Menú del navegador → Herramientas de desarrollador**

4.3 Tabs Principales de DevTools

1. Elements (Elementos)

Propósito: Inspeccionar y modificar HTML y CSS en tiempo real

Funcionalidades principales:

```
<!-- Puedes modificar HTML directamente -->
<div class="container">
    <!-- Doble click para editar -->
    <h1>Título modificable en vivo</h1>
</div>
```

Casos de uso:

- Inspeccionar estructura HTML de cualquier página
- Modificar CSS en tiempo real para probar cambios
- Ver qué estilos se están aplicando a cada elemento
- Debuggear problemas de layout y responsive design

Ejercicio práctico:

1. Abre cualquier sitio web
2. Presiona F12
3. Click derecho en un elemento → "Inspeccionar"
4. Modifica el texto del elemento
5. Cambia colores en la sección Styles

Console (Consola)

Propósito: Ejecutar JavaScript y ver mensajes de error/debug

Comandos básicos:

```
// Información básica de la página
console.log("Hola desde la consola");
document.title
document.URL
window.location.href

// Seleccionar elementos (similar a querySelector)
$('#miId')          // Equivale a document.getElementById('miId')
$$('.miClase')      // Equivale a document.querySelectorAll('.miClase')

// Limpiar consola
clear()

// Inspeccionar objetos
console.table(['manzana', 'banana', 'cereza']);
```

Sources (Fuentes)

Propósito: Debuggear JavaScript con breakpoints

Funcionalidades:

- Ver código fuente de archivos JavaScript
- Establecer breakpoints (puntos de parada)
- Step through code (ejecutar paso a paso)
- Inspeccionar variables en tiempo de ejecución

Cómo usar breakpoints:

1. Abrir tab Sources
2. Navegar al archivo JavaScript
3. Click en número de línea para crear breakpoint
4. Recargar página o ejecutar función
5. Usar controles de debugging (Step over, Step into, Continue)

Network (Red)

Propósito: Monitorear todas las peticiones HTTP de la página

Información que proporciona:

- Archivos descargados (HTML, CSS, JS, imágenes)
- Tiempo de carga de cada recurso
- Tamaño de archivos
- Códigos de estado HTTP
- Headers de peticiones y respuestas

Ejercicio de Network:

1. Abrir tab Network
2. Recargar página (Ctrl+R)
3. Observar todos los archivos que se descargan
4. Click en cualquier archivo para ver detalles
5. Identificar el archivo más pesado

Performance (Rendimiento)

Propósito: Analizar el rendimiento de la página

Métricas importantes:

- Tiempo de carga inicial
- First Contentful Paint (FCP)
- Largest Contentful Paint (LCP)
- Cumulative Layout Shift (CLS)

Application (Aplicación)

Propósito: Gestionar almacenamiento local y cookies

Incluye:

- Local Storage
- Session Storage
- Cookies
- IndexedDB
- Service Workers

Responsive Design Testing

Device Mode

```
Activar: Ctrl+Shift+M o icono de dispositivo móvil
```

Funcionalidades:

- Simular diferentes tamaños de pantalla
- Dispositivos predefinidos (iPhone, iPad, etc.)
- Orientación portrait/landscape
- Throttling de red para simular conexiones lentas

Breakpoints comunes para testing:

- **Mobile**: 320px - 767px
- **Tablet**: 768px - 1023px
- **Desktop**: 1024px+