Temas a tratar:   
  
**HTML (Lenguaje de marcado de hipertexto):** define la estructura del contenido en una página web (encabezados, párrafos, imágenes, enlaces, etc.).

**CSS (Cascading Style Sheets):** Controla el diseño y la presentación de los elementos HTML (colores, fuentes, márgenes, disposición, etc.).

**JavaScript:** Lenguaje de programación que permite agregar interactividad y dinamismo a las páginas web.

**Diseño Responsivo:** Uso de CSS y técnicas como *media queries* para adaptar el sitio a diferentes tamaños de pantalla (móviles, tabletas, computadoras).

**DOM (Document Object Model):** Representación en árbol de los elementos de una página web que JavaScript puede manipular dinámicamente.

**Protocolo HTTP/HTTPS:** Reglas que permiten la comunicación entre navegadores y servidores web.

**Frontend vs. Backend:**

* *Frontend* es la parte visual e interactiva de un sitio web (HTML, CSS, JavaScript).
* *Backend* es la lógica detrás de escena que maneja bases de datos, autenticación y lógica del servidor (PHP, Python, Node.js, etc.).

**APIs (Application Programming Interfaces):** Interfaces que permiten la comunicación entre sistemas o servicios en la web.

HTML (lenguaje de marcado de hipertexto)

**HTML es el lenguaje de marcado utilizado para estructurar y presentar contenido en la web. Funciona a través de etiquetas que definen los diferentes elementos de una página, como encabezados, párrafos, imágenes y enlaces.**

**<!DOCTYPE html>**

**<html lang="es">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Mi Página Web</title>**

**</head>**

**<body>**

**<h1>¡Hola, mundo!</h1>**

**<p>Este es un ejemplo básico de HTML.</p>**

**</body>**

**</html>**

#### **Explicación:**

1. **<!DOCTYPE html>→ Indica que el documento es HTML5.**
2. **<html lang="es">→ Define el idioma del documento.**
3. **<head>→ Contiene metadatos como el título de la página, la codificación de caracteres ( UTF-8 para admitir acentos y caracteres especiales) y la configuración para que sea responsivo ( viewport).**
4. **<title>→ Nombre que aparece en la pestaña del navegador.**
5. **<body>→ Contiene todo el contenido visible de la página.**

**Etiquetas comunes:**

**Encabezados**

**<h1>Título Principal</h1>**

**<h2>Subtítulo</h2>**

**<h3>Sección</h3>**

**Párrafos y texto**

**<p>Este es un párrafo.</p>**

**<strong>Texto en negrita</strong>**

**<em>Texto en cursiva</em>**

**Enlaces**

**<a href="https://www.ejemplo.com">Visita Ejemplo</a>**

**Imágenes**

**<img src="imagen.jpg" alt="Descripción de la imagen">**

**Listas**

**Ordenadas ( <ol>)**

**<ol>**

**<li>Primer ítem</li>**

**<li>Segundo ítem</li>**

**</ol>**

**No ordenados ( <ul>)**

**<ul>**

**<li>Ítem 1</li>**

**<li>Ítem 2</li>**

**</ul>**

**Tablas**

**<table border="1">**

**<tr>**

**<th>Nombre</th>**

**<th>Edad</th>**

**</tr>**

**<tr>**

**<td>Ana</td>**

**<td>25</td>**

**</tr>**

**</table>**

**Formularios**

**<form action="procesar.php" method="POST">**

**<label for="nombre">Nombre:</label>**

**<input type="text" id="nombre" name="nombre">**

**<label for="email">Correo:</label>**

**<input type="email" id="email" name="email">**

**<button type="submit">Enviar</button>**

**</form>**

**HTML y CSS**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**body { background-color: lightblue; text-align: center; }**

**h1 { color: darkblue; }**

**</style>**

**</head>**

**<body>**

**<h1>Hola, mundo con estilo</h1>**

**</body>**

**</html>**

HTML y JAVASCRIPT

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Formulario con JavaScript</title>

</head>

<body>

<form id="miFormulario">

<label for="nombre">Nombre:</label>

<input type="text" id="nombre" name="nombre">

<label for="email">Correo:</label>

<input type="email" id="email" name="email">

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

</form>

<script>

document.getElementById("miFormulario").addEventListener("submit", function(event) {

event.preventDefault(); // Evita el envío del formulario

// Capturar los valores de los inputs

const nombre = document.getElementById("nombre").value;

const email = document.getElementById("email").value;

// Mostrar los valores en la consola o procesarlos

console.log("Nombre:", nombre);

console.log("Correo:", email);

// Aquí puedes agregar lógica adicional, como enviar los datos con fetch

});

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Formulario con JavaScript</title>

</head>

<body>

<form id="miFormulario">

<label for="nombre">Nombre:</label>

<input type="text" id="nombre" name="nombre">

<label for="email">Correo:</label>

<input type="email" id="email" name="email">

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

</form>

<script>

document.getElementById("miFormulario").addEventListener("submit", function(event) {

event.preventDefault(); // Evita el envío del formulario

// Capturar los valores de los inputs

const nombre = document.getElementById("nombre").value;

const email = document.getElementById("email").value;

// Mostrar los valores en la consola o procesarlos

console.log("Nombre:", nombre);

console.log("Correo:", email);

// Aquí puedes agregar lógica adicional, como enviar los datos con fetch

});

</script>

</body>

</html>

### 

### 

### 

CSS

El **CSS (Cascading Style Sheets)** es un lenguaje de estilos que se usa para controlar la apariencia y el diseño de documentos HTML. Permite definir cómo se ven los elementos en una página web, incluyendo colores, tamaños, márgenes, fuentes y posiciones.

### **Características principales de CSS:**

1. **Separación de contenido y diseño**: HTML estructura el contenido, y CSS lo estiliza.
2. **Cascada y herencia**: Los estilos pueden aplicarse desde distintos niveles (inline, interno y externo) y pueden heredarse entre elementos.
3. **Flexibilidad y reutilización**: Un solo archivo CSS puede dar estilo a múltiples páginas HTML.

### **Formas de aplicar CSS:**

**CSS en línea (inline)**: Se aplica directamente en los elementos HTML.  
 html  
  
<h1 style="color: blue;">Hola, CSS</h1>

**CSS interno (en el <head> de HTML)**: Se escribe dentro de <style>.  
 html  
  
<style>

h1 { color: blue; }

</style>

**CSS externo (archivo .css)**: Se enlaza con HTML usando <link>.  
 html  
  
<link rel="stylesheet" href="styles.css">

En styles.css:  
 css  
  
h1 {

color: blue;

font-size: 24px;

1. }

### **Colores y Fondos**

* color → Define el color del texto.
* background-color → Define el color de fondo.
* background-image → Establece una imagen de fondo.
* background-size → Ajusta el tamaño de la imagen de fondo.
* background-position → Define la posición de la imagen de fondo.
* opacity → Define la transparencia (0 es transparente, 1 es opaco).

### **🖋 Texto y Tipografía**

* font-family → Define la fuente del texto.
* font-size → Tamaño del texto.
* font-weight → Grosor del texto (normal, bold, lighter, bolder, 100-900).
* font-style → Define si el texto es normal o itálico (normal, italic, oblique).
* text-align → Alinea el texto (left, center, right, justify).
* text-decoration → Agrega subrayado, tachado, etc. (none, underline, line-through).
* text-transform → Cambia mayúsculas/minúsculas (uppercase, lowercase, capitalize).
* letter-spacing → Espaciado entre letras.
* line-height → Altura de línea.

### **📏 Dimensiones y Tamaño**

* width → Ancho del elemento (px, %, vw).
* height → Alto del elemento.
* max-width / min-width → Ancho máximo y mínimo.
* max-height / min-height → Alto máximo y mínimo.

### **📦 Márgenes y Espaciado**

* margin → Espacio exterior alrededor del elemento.
* padding → Espacio interior dentro del elemento.
* border → Agrega bordes (border: 2px solid black;).
* border-radius → Bordes redondeados.
* box-shadow → Agrega sombra a los elementos.

### **📌 Posicionamiento y Diseño**

* display → Controla el tipo de caja (block, inline, flex, grid, none).
* position → Posiciona el elemento (static, relative, absolute, fixed, sticky).
* top, right, bottom, left → Controla la posición si position no es static.
* z-index → Controla la superposición de elementos.
* overflow → Define qué hacer con el contenido desbordado (visible, hidden, scroll, auto).

### **🎯 Flexbox (Diseño Responsive)**

* display: flex; → Activa el modelo Flexbox.
* justify-content → Alinea elementos en el eje principal (flex-start, center, space-between).
* align-items → Alinea en el eje secundario (flex-start, center, stretch).
* flex-direction → Define la dirección (row, column).
* flex-wrap → Permite que los elementos se ajusten (nowrap, wrap).

### **🔲 Grid (Diseño en Cuadrícula)**

* display: grid; → Activa el modelo Grid.
* grid-template-columns → Define las columnas.
* grid-template-rows → Define las filas.
* gap → Define el espacio entre filas y columnas.
* grid-column → Define en qué columnas debe estar un elemento.

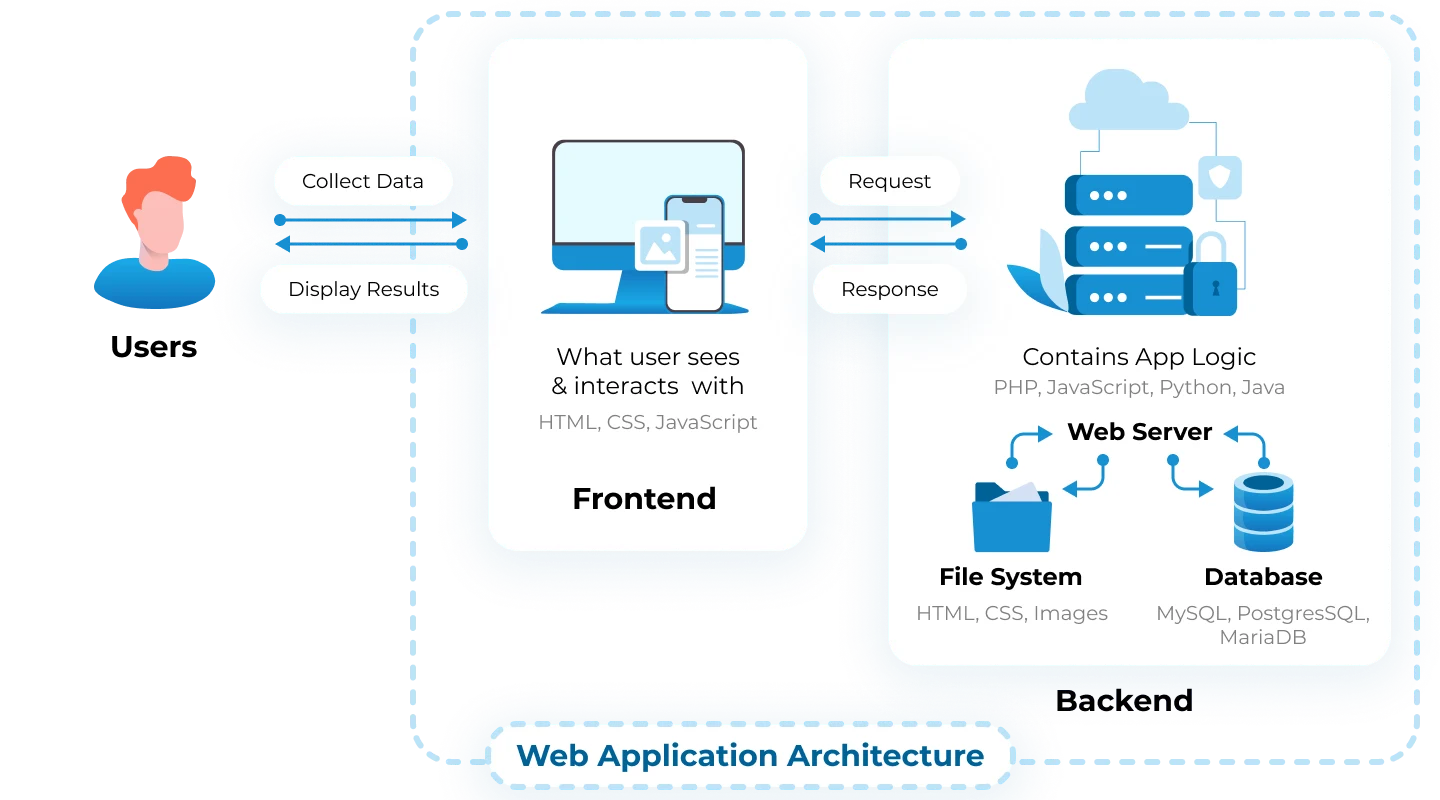
### **🎬 Animaciones y Transiciones**

* transition → Agrega animaciones en cambios de propiedades.
* animation → Crea animaciones avanzadas.
* transform → Permite rotar, escalar y mover elementos (rotate(), scale(), translate()).

### **🎭 Filtros y Efectos Visuales**

* filter → Agrega efectos (blur(), brightness(), contrast()).
* backdrop-filter → Aplica efectos a elementos detrás.

Arquitectura de una aplicación WEB



La arquitectura de aplicaciones web es el marco estructural que guía el diseño y el desarrollo de una aplicación web . Implica un conjunto de principios y prácticas recomendadas que determinan cómo funcionan juntas las distintas partes de la aplicación para ofrecer las funciones previstas.

En términos más simples, es como el plano que organiza el frontend (lo que ven los usuarios), el backend (servidor y lógica de negocios ) y la base de datos, garantizando que interactúen sin problemas.

El objetivo principal es crear un sistema unificado y escalable que satisfaga las necesidades funcionales de la aplicación, considerando también factores como el rendimiento, la seguridad y la capacidad de mantenimiento.

**Objetivos principales de la arquitectura de aplicaciones web**

Los objetivos principales de la arquitectura de aplicaciones web giran en torno a garantizar la eficiencia, confiabilidad y escalabilidad de un sistema digital.

En primer lugar, su objetivo es proporcionar una experiencia de usuario fluida e intuitiva organizando los componentes del frontend, como interfaces e interacciones, de una manera fácil de usar.

En segundo lugar, la arquitectura aborda la funcionalidad del backend, centrándose en la creación de un servidor robusto y eficiente que maneja la lógica de la aplicación , procesa datos y se comunica de manera efectiva con el frontend.

La escalabilidad es otro objetivo crucial, cuyo objetivo es diseñar un sistema que pueda manejar el crecimiento de usuarios o del volumen de datos sin comprometer el rendimiento.

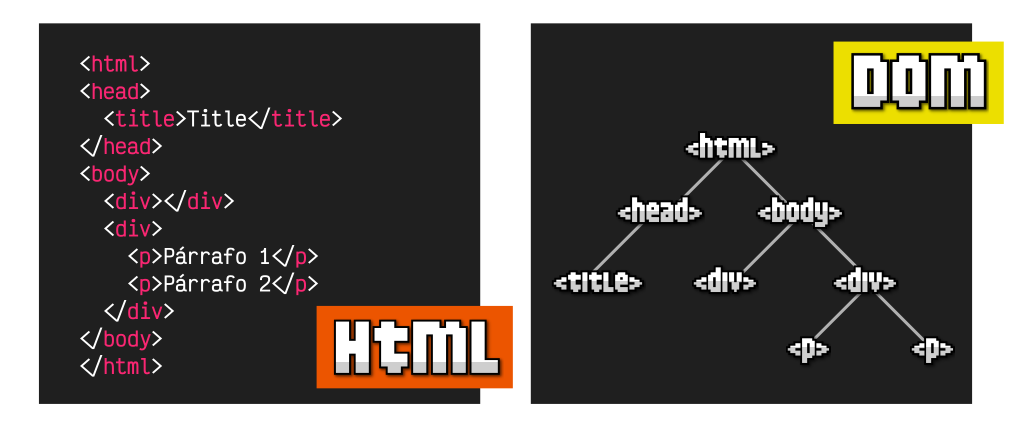
Las consideraciones de seguridad están integradas en la arquitectura de la aplicación web para salvaguardar la información confidencial y proteger contra posibles amenazas.

**Las principales tareas en la arquitectura de software incluyen:**

* Diseño del sistema: desarrollar una estructura y organización de alto nivel para todo el sistema de software, describiendo los componentes clave de la aplicación web y sus interacciones.
* Análisis de requisitos: colaborar con las partes interesadas para comprender y analizar los requisitos del software, garantizando la alineación con los objetivos comerciales y las necesidades de los usuarios.
* Planificación de escalabilidad: planifique el crecimiento futuro diseñando un sistema que pueda manejar de manera eficiente el aumento de carga, usuarios y datos.
* Optimización del rendimiento: identifique posibles cuellos de botella y optimice la arquitectura del software para mejorar la velocidad y la capacidad de respuesta.
* Selección de tecnología: elegir tecnologías, marcos y herramientas adecuados que se alineen con los objetivos y requisitos del proyecto.
* Planificación de seguridad: implementar medidas de seguridad a nivel arquitectónico para protegerse contra posibles amenazas y vulnerabilidades.

<https://litslink.com/blog/web-application-architecture>

DOM



El **DOM (Document Object Model)** de HTML es una representación estructurada de una página web en forma de árbol. Es una interfaz de programación que permite a los scripts (como JavaScript) acceder y manipular el contenido, la estructura y el estilo de un documento HTML o XML.

### **Características clave del DOM:**

1. **Estructura en árbol**: Cada elemento HTML es un nodo en el árbol del documento.
2. **Modificable en tiempo real**: Se pueden cambiar elementos, atributos, texto y estilos de la página sin necesidad de recargarla.
3. **Accesible con JavaScript**: A través de document y métodos como getElementById() o querySelector(), se pueden modificar los elementos.

### **Ejemplo de estructura del DOM:**

Si tienes este HTML:

html

<!DOCTYPE html>

<html>

<head>

<title>Mi Página</title>

</head>

<body>

<h1 id="titulo">Hola, DOM</h1>

<p class="texto">Este es un párrafo.</p>

</body>

</html>

El DOM lo representaría como un árbol donde html es el nodo raíz, head y body son sus hijos, y así sucesivamente.

Con JavaScript, podrías modificarlo así:

javascript

document.getElementById("titulo").textContent = "Nuevo título";

Esto cambiaría el contenido del <h1> sin recargar la página.

JavaScript

**JavaScript (JS)** es un lenguaje de programación que se usa para hacer que las páginas web sean interactivas y dinámicas. Permite manipular el HTML y CSS, responder a eventos del usuario, realizar solicitudes a servidores y más.

### **Características principales de JavaScript:**

1. **Lenguaje de scripting**: Se ejecuta en el navegador sin necesidad de compilación.
2. **Interactividad y dinamismo**: Puede cambiar el contenido de la página en respuesta a eventos.
3. **Manipulación del DOM**: Permite acceder y modificar elementos HTML y estilos CSS.
4. **Orientado a eventos**: Responde a acciones del usuario como clics, movimientos del mouse y entradas de teclado.
5. **Compatible con múltiples plataformas**: Funciona en navegadores, servidores (con Node.js) y más.

### **Ejemplo básico de JavaScript:**

#### **Mostrar un mensaje en la consola:**

javascript

console.log("¡Hola, JavaScript!");

#### **Cambiar el contenido de una página:**

Si tienes este HTML:

html

<h1 id="titulo">Hola</h1>

<button onclick="cambiarTexto()">Cambiar Texto</button>

<script>

function cambiarTexto() {

document.getElementById("titulo").textContent = "¡Texto cambiado!";

}

</script>

Cuando el usuario haga clic en el botón, el texto del <h1> cambiará.

## **1. Introducción a JavaScript (Teoría + Práctica)**

📌 Explica qué es JavaScript y por qué es importante.  
 ✔ ¿Qué hace JavaScript? (interactividad en páginas web).  
 ✔ Cómo se ejecuta en el navegador (consola del navegador).

🎯 **Ejercicio práctico**:  
 Abre la consola en el navegador (F12 → "Console") y ejecuta:

javascript

console.log("¡Hola, JavaScript!");

## **🔹 2. Sintaxis básica y variables**

✔ Declaración de variables (let, const, var).  
 ✔ Tipos de datos: **números, strings, booleanos, arrays, objetos**.

🎯 **Ejercicio práctico**:

javascript

let nombre = "Martín";

let edad = 25;

console.log("Hola, mi nombre es " + nombre + " y tengo " + edad + " años.");

## **🔹 3. Operadores y estructuras de control**

✔ Operadores matemáticos (+, -, \*, /, %).  
 ✔ Condiciones (if, else, switch).  
 ✔ Bucles (for, while).

🎯 **Ejercicio práctico**:

javascript

let numero = 10;

if (numero > 5) {

console.log("El número es mayor que 5");

} else {

console.log("El número es menor o igual a 5");

}

El **switch** en JavaScript es una estructura de control que permite ejecutar diferentes bloques de código según el valor de una variable o expresión. Es útil cuando tienes múltiples casos a evaluar en lugar de usar varios if...else.

let opcion = prompt("Elige una opción: 1. Saludar 2. Despedirse 3. Decir gracias");

switch (opcion) {

case "1":

console.log("¡Hola!");

break;

case "2":

console.log("¡Adiós!");

break;

case "3":

console.log("¡Gracias!");

break;

default:

console.log("Opción no válida");

}

## **🔹 4. Funciones en JavaScript**

✔ Declaración de funciones (function, arrow functions).  
 ✔ Parámetros y valores de retorno.

🎯 **Ejercicio práctico**:

javascript

function sumar(a, b) {

return a + b;

}

console.log(sumar(3, 7)); // 10

## **🔹 5. Manipulación del DOM**

✔ document.getElementById(), querySelector().  
 ✔ Modificación de elementos (innerHTML, textContent).  
 ✔ Eventos (addEventListener).

🎯 **Ejercicio práctico**:

html

<h1 id="titulo">¡Hola!</h1>

<button onclick="cambiarTexto()">Cambiar Texto</button>

<script>

function cambiarTexto() {

document.getElementById("titulo").textContent = "¡Texto cambiado!";

}

</script>

## **🔹 6. Arrays y objetos**

✔ Métodos de Arrays (push, map, filter).  
 ✔ Objetos en JavaScript.

🎯 **Ejercicio práctico**:

javascript

let frutas = ["Manzana", "Banana", "Naranja"];

frutas.push("Uva");

console.log(frutas);

let persona = { nombre: "Martín", edad: 25 };

console.log(persona.nombre);

## **🔹 7. Introducción a APIs y Fetch**

✔ Explicar qué es una API.  
 ✔ Uso de fetch() para obtener datos.

🎯 **Ejercicio práctico**:

javascript

fetch("https://jsonplaceholder.typicode.com/posts/1")

.then(response => response.json())

.then(data => console.log(data));

## **🔹 8. Proyectos finales para aplicar conocimientos**

🎯 **Ideas de proyectos**: ✔ Un contador de clics.  
 ✔ Un pequeño juego de adivinanza.  
 ✔ Un "To-Do List" con almacenamiento en localStorage.  
 ✔ Consumir una API pública y mostrar datos en la web.

9.WHile

let i = 0;

while (i < 3) { // muestra 0, luego 1, luego 2

alert( i );

i++;

}

For

for (let i = 0; i < 3; i++) {

alert(i);

}

Protocolo HTTP/HTTPS

El **protocolo HTTP (HyperText Transfer Protocol)** y su versión segura **HTTPS (HyperText Transfer Protocol Secure)** son los protocolos utilizados para la comunicación entre navegadores y servidores web.

## **🔹 HTTP (HyperText Transfer Protocol)**

Es un protocolo de comunicación que permite la transferencia de información en la web. Funciona mediante **peticiones y respuestas** entre un cliente (navegador) y un servidor.

### **📌 Características de HTTP:**

* **Basado en texto**: Usa solicitudes como GET, POST, PUT, DELETE.
* **Sin estado (stateless)**: No recuerda información entre peticiones.
* **Sin cifrado**: La información viaja en texto plano, lo que la hace vulnerable a ataques.

### **📌 Ejemplo de una petición HTTP GET:**

Un navegador solicita una página web:

vbnet

GET /index.html HTTP/1.1

Host: www.ejemplo.com

El servidor responde con el contenido de la página.

## **🔹 HTTPS (HyperText Transfer Protocol Secure)**

Es la versión segura de HTTP, que añade **cifrado mediante SSL/TLS** para proteger la información durante la transmisión.

### **📌 Beneficios de HTTPS:**

✔ **Cifrado**: Protege los datos del usuario mediante **SSL/TLS**.  
 ✔ **Autenticación**: Verifica la identidad del servidor con un **certificado digital**.  
 ✔ **Integridad**: Asegura que los datos no han sido modificados por terceros.

### **📌 Diferencias clave entre HTTP y HTTPS:**

| **Característica** | **HTTP** | **HTTPS** |
| --- | --- | --- |
| Seguridad | No cifrado | Cifrado con SSL/TLS |
| Uso en sitios sensibles | No recomendado | Recomendado (bancos, login, pagos) |
| Certificado necesario | No | Sí (Certificado SSL) |
| SEO | No favorecido | Google lo favorece en los rankings |

## **🔹 Ejemplo de URL con HTTP vs HTTPS**

* **HTTP**: http://www.ejemplo.com
* **HTTPS**: https://www.ejemplo.com (🔒 más seguro)

Hoy en día, **HTTPS es el estándar** en la web para proteger la privacidad y seguridad de los usuarios.

APIs

Una **API (Application Programming Interface)** es un conjunto de reglas y definiciones que permite a diferentes aplicaciones comunicarse entre sí. Básicamente, es un intermediario que facilita la interacción entre software sin que los desarrolladores necesiten conocer los detalles internos de cada aplicación.

## **🔹 ¿Cómo funciona una API?**

Las APIs funcionan mediante **peticiones y respuestas**. Un cliente (como una aplicación o un navegador) hace una solicitud a la API, y esta devuelve una respuesta con los datos solicitados.

Ejemplo de una petición HTTP a una API:

bash

GET <https://api.weather.com/v1/clima?ciudad=BuenosAires>

https://v2.jokeapi.dev/joke/Any?lang=es

📌 **Respuesta de la API (JSON)**:

json

{

"ciudad": "Buenos Aires",

"temperatura": "22°C",

"condición": "Soleado"

}

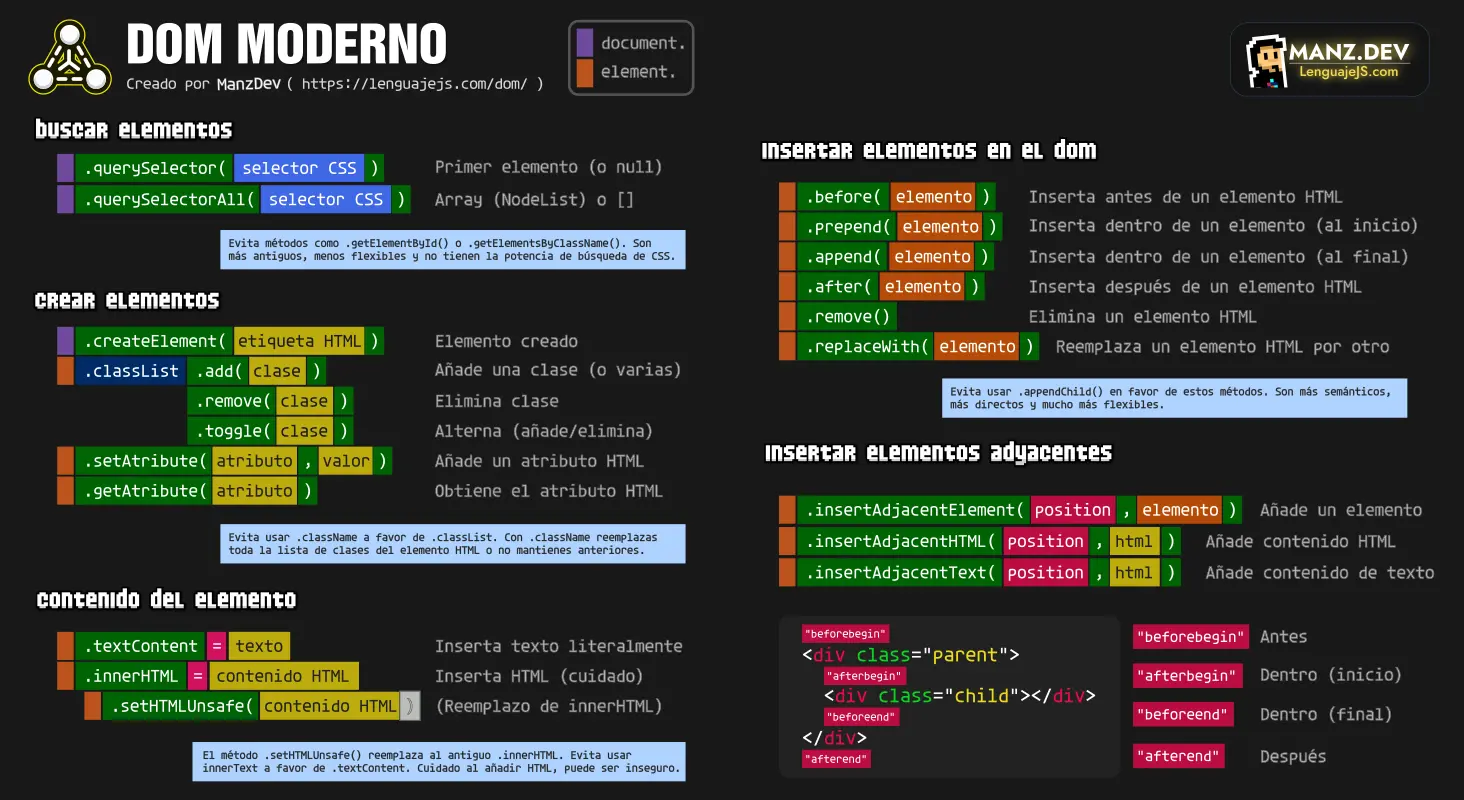
Aquí, la API del clima devuelve información sobre el clima en Buenos Aires.

## **🔹 Tipos de APIs**

1. **APIs REST (Representational State Transfer)** 🌍  
   * Basadas en HTTP.
   * Usan formatos como **JSON** o **XML**.
   * Métodos comunes: GET, POST, PUT, DELETE.
2. **APIs SOAP (Simple Object Access Protocol)** 💾  
   * Más estrictas y seguras que REST.
   * Utilizan XML para la comunicación.
3. **APIs GraphQL** 📊  
   * Permiten obtener solo los datos específicos que se necesitan.
   * Más eficientes que REST para ciertas aplicaciones.
4. **APIs de terceros** 🔌  
   * Proporcionadas por empresas como Google, Twitter o OpenAI para acceder a sus servicios.

Complementario:

A continuación tienes una breve cheatsheet para manejar el DOM de Javascript a modo de resumen. Las explicaciones para entender a utilizar todos estos conceptos lo veremos a lo largo de este curso y sus capítulos, con algunos ejemplos:



Ejercicio menu de navegacion:

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Menú de Navegación</title>

<style>

/\* Estilos para el menú \*/

nav {

background-color: #333;

padding: 10px;

}

ul {

list-style: none;

margin: 0;

padding: 0;

display: flex;

}

li {

margin-right: 20px;

}

a {

text-decoration: none;

color: white;

font-size: 18px;

padding: 8px 12px;

transition: 0.3s;

}

a:hover {

background-color: #555;

border-radius: 5px;

}

</style>

</head>

<body>

<nav>

<ul>

<li><a href="#">Inicio</a></li>

<li><a href="#">Acerca de</a></li>

<li><a href="#">Servicios</a></li>

<li><a href="#">Contacto</a></li>

</ul>

</nav>

<br>

<form action="#" method="POST">

<label for="nombre">Nombre:</label>

<input type="text" id="nombre" name="nombre">

<label for="email">Correo:</label>

<input type="email" id="email" name="email">

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

</form>

</body>

</html>

Formularios

Los formularios en HTML sirven para recopilar datos de los usuarios y enviarlos a un servidor o procesarlos de diferentes maneras. Son esenciales en sitios web interactivos y permiten a los usuarios:

1. **Enviar información**: Pueden ser utilizados para que los usuarios envíen datos, como nombre, dirección de correo electrónico, comentarios, etc.
2. **Interacción con el sistema**: A través de formularios, los usuarios pueden realizar acciones como registrarse, iniciar sesión, comprar productos, solicitar información, entre otros.
3. **Recopilación de datos**: Los formularios permiten a las empresas o servicios recopilar datos de los usuarios, como encuestas, suscripciones a boletines o feedback.
4. **Validación de datos**: Pueden incluir validaciones, como verificar si el formato del correo electrónico es correcto o si los campos obligatorios están completos.
5. **Interactividad en la web**: Facilitan una comunicación bidireccional entre el usuario y el sistema, como hacer una consulta o enviar una solicitud.

### **Ejemplos comunes de formularios:**

* **Registro y autenticación de usuarios** (nombre, correo, contraseña).
* **Encuestas y formularios de contacto**.
* **Carritos de compra y pagos en línea**.
* **Suscripciones a boletines o newsletters**.

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Formulario de Correo</title>

</head>

<body>

<h2>Enviar un Correo</h2>

<form action="mailto:correo@ejemplo.com" method="post" enctype="text/plain">

<label for="para">Para:</label>

<input type="email" name="para" id="para" required><br><br>

<label for="asunto">Asunto:</label>

<input type="text" name="asunto" id="asunto"><br><br>

<label for="mensaje">Mensaje:</label><br>

<textarea name="mensaje" id="mensaje" rows="5"></textarea><br><br>

<input type="submit" value="Enviar">

</form>

</body>

</html>

El **pensamiento computacional** es un enfoque para resolver problemas, diseñar sistemas y entender procesos mediante los principios y conceptos de la informática. Implica descomponer problemas complejos en partes más manejables, identificar patrones, abstraer detalles y crear algoritmos para encontrar soluciones.

### **Componentes clave del pensamiento computacional:**

1. **Descomposición**: Desglosar un problema grande y complejo en partes más pequeñas y más fáciles de manejar. Esto permite abordar cada componente por separado y encontrar soluciones más simples.
2. **Reconocimiento de patrones**: Identificar similitudes o patrones en los problemas, lo que permite aplicar soluciones previas a nuevos problemas o simplificar el proceso de resolución.
3. **Abstracción**: Ignorar los detalles irrelevantes para centrarse en los aspectos más importantes o relevantes de un problema. Esto permite crear modelos generales y más eficientes.
4. **Algoritmos**: Crear un conjunto de pasos o instrucciones que guíen la resolución de un problema de manera lógica y ordenada. Los algoritmos son esenciales para que las computadoras ejecuten tareas automáticamente.

### **Aplicaciones del pensamiento computacional:**

* **Educación**: Enseñar a los estudiantes a pensar de manera estructurada y lógica, ayudándolos a desarrollar habilidades para resolver problemas no solo en programación, sino también en matemáticas, ciencias y otras áreas.
* **Desarrollo de software**: En la creación de aplicaciones, programas y sistemas informáticos, el pensamiento computacional ayuda a estructurar y optimizar soluciones.
* **Solución de problemas cotidianos**: No solo se aplica en la programación; es una forma de abordar problemas en diversas áreas de la vida diaria de manera lógica y organizada.

### **En resumen:**

El pensamiento computacional no es solo para programadores, sino que es una habilidad valiosa que puede ayudar a las personas a abordar y resolver problemas de manera más eficiente en muchas disciplinas, utilizando un enfoque lógico y sistemático.