

DESARROLLO DE SOFTWARE

UNIDAD N° 4





Contenidos

- Fundamentos del Desarrollo Web
- Herramientas de Desarrollo del Navegador (DevTools)
- HTML (HyperText Markup Language)
- CSS (Cascading Style Sheets)
- Diseño Web Adaptativo (Responsive Design)
- JavaScript
- JavaScript: Manipulación del DOM (Document Object Model)
- JavaScript: Manejo de Controles y Arrays
- JavaScript: Manejo de Errores y Eventos
- Tablas Dinámicas y Paginación
- Formularios y Validación
- Fetch API
- Almacenamiento del Lado del Cliente



Recursos de lectura

- <https://developer.mozilla.org/es/>
- <https://es.react.dev/>
- <https://www.reactjs.wiki/>
- <https://vite.dev/guide/>
- <https://react-hook-form.com/get-started>
- <https://tailwindcss.com/docs/installation/using-vite>



Material de Estudio

Primera Parte

Obligatorio:

- [¿Cómo funciona internet?](#)
- [¿Qué es un nombre de dominio?](#)
- [¿Qué es una URL?](#)
- [¿Qué es un servidor WEB?](#)
- [HTML](#)
- [CSS](#)
- [CSS - Cascada y Herencia](#)
- [Media Queries](#)

Complementario:

- [DevTools](#)
- [Diseño responsive](#)
- [Mobile first](#)



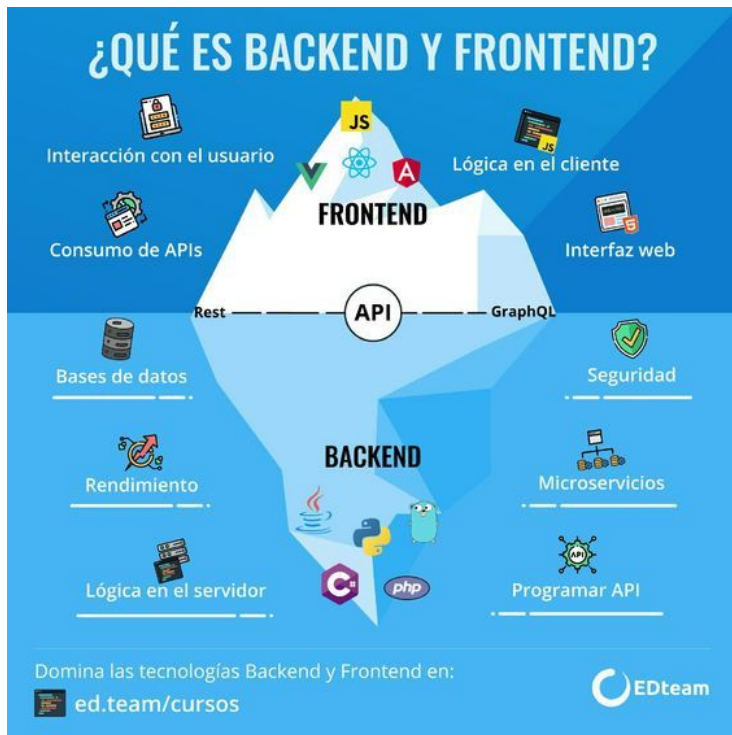
Material de Estudio

Segunda Parte

Obligatorio:

- [Javascript](#)
- [Motor JS](#)
- [Motor de Renderizado](#)
- [DOM](#)
- [Usando el DOM](#)
- [Document API](#)
- [Eventos](#)
- [Objetos](#)
- [Array](#)
- [Promise](#)
- [Fetch](#)
- [sessionStorage y localStorage](#)
- [Cookie](#)
- [IndexedDB](#)

Fundamentos del Desarrollo Web





Fundamentos del Desarrollo Web

Frontend (Cliente)	Backend (Servidor)	Conocimientos generales	Herramientas y flujos de trabajo modernos
<p>Lenguajes esenciales:</p> <ul style="list-style-type: none">• HTML (Lenguaje de etiquetas de hipertexto)• CSS (Hojas de estilo en cascada)• JavaScript <p>Tecnologías y herramientas comunes:</p> <ul style="list-style-type: none">• Frameworks y bibliotecas: React, Vue.js, Angular.• Preprocesadores CSS: SASS, LESS.• Gestores de paquetes: npm, yarn.• Bundlers y transpiladores: Webpack, Babel, Vite.	<p>Lenguajes y entornos comunes:</p> <ul style="list-style-type: none">• JavaScript (Node.js), Python (Django/Flask), PHP, Java (Spring), C# (.NET), Ruby (Rails). <p>Componentes clave:</p> <ul style="list-style-type: none">• Servidor web: Nginx, Apache.• Frameworks de backend• Base de datos <p>Conceptos importantes:</p> <ul style="list-style-type: none">• Rutas (Routing): define cómo responder a solicitudes HTTP.• APIs (REST, GraphQL): permiten que frontend y backend se comuniquen.• Autenticación/autorización: control de acceso con JWT, OAuth, etc.	<p>Protocolos y arquitectura:</p> <ul style="list-style-type: none">• HTTP/HTTPS• DNS, IP, URL, dominios.• Modelo cliente-servidor.• Ciclo de vida de una petición web. <p>Buenas prácticas:</p> <ul style="list-style-type: none">• Seguridad• Performance• Accesibilidad (a11y)• Responsive design• Control de versiones <p>Testing y mantenimiento:</p> <ul style="list-style-type: none">• Testing: unitario, integración, e2e.• CI/CD: integración y despliegue continuo (ej: GitHub Actions, Jenkins).• Documentación del código y APIs.	<p>Editores de código: VS Code, WebStorm, Visual Studio.</p> <p>DevTools del navegador.</p> <p>Contenedores y entornos virtuales: Docker, Vagrant.</p> <p>Despliegue en la nube: Vercel, Netlify, Heroku, AWS, etc.</p>



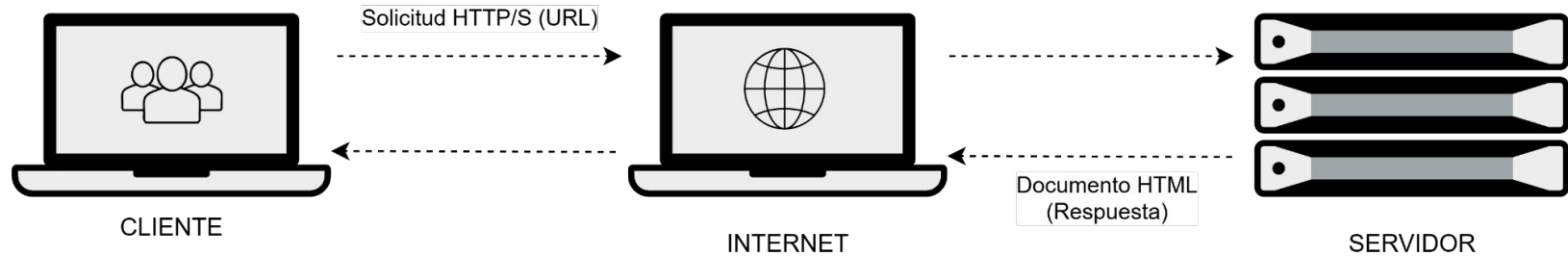
Fundamentos del Desarrollo Web

- **IP:** Dirección numérica única que identifica a cada dispositivo a una red.
- **Dominio:** Nombre fácil de recordar que se usa para acceder a una dirección IP.
- **DNS:** Sistema que traduce un Dominio a su dirección IP real.
- **URL:** Una dirección completa que se escribe en el navegador

Ejemplo:

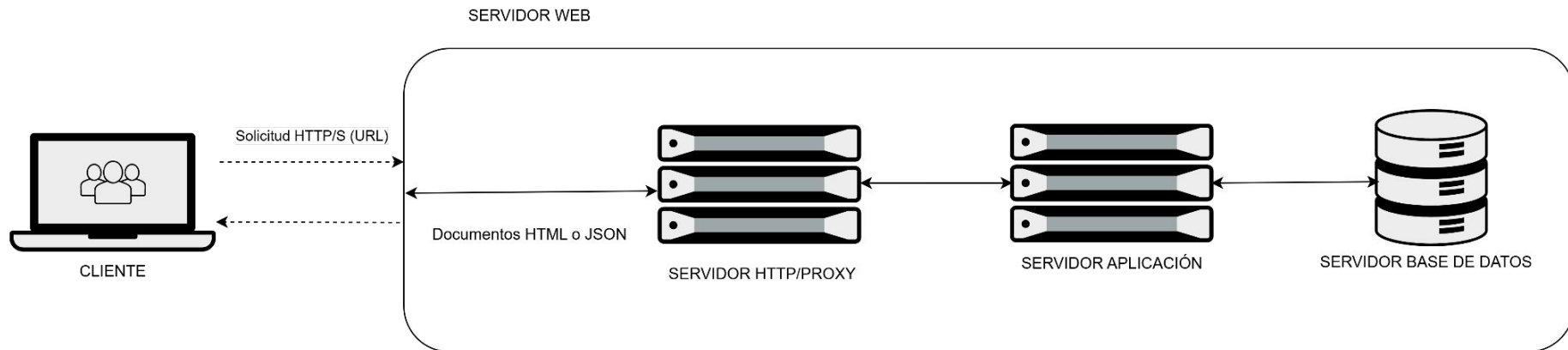
- URL: <https://www.google.com/search?q=chatgpt>
 - Protocolo: [https://](#)
 - Dominio: [www.google.com](#)
 - Ruta del recurso: [/search](#)
 - Parámetros: [?q=chatgpt](#)
- DNS convierte [www.google.com](#) a la IP [142.250.217.46](#).
- Tu navegador usa esa IP para conectarse al servidor de Google.

Fundamentos del Desarrollo Web (Visión simplificada)

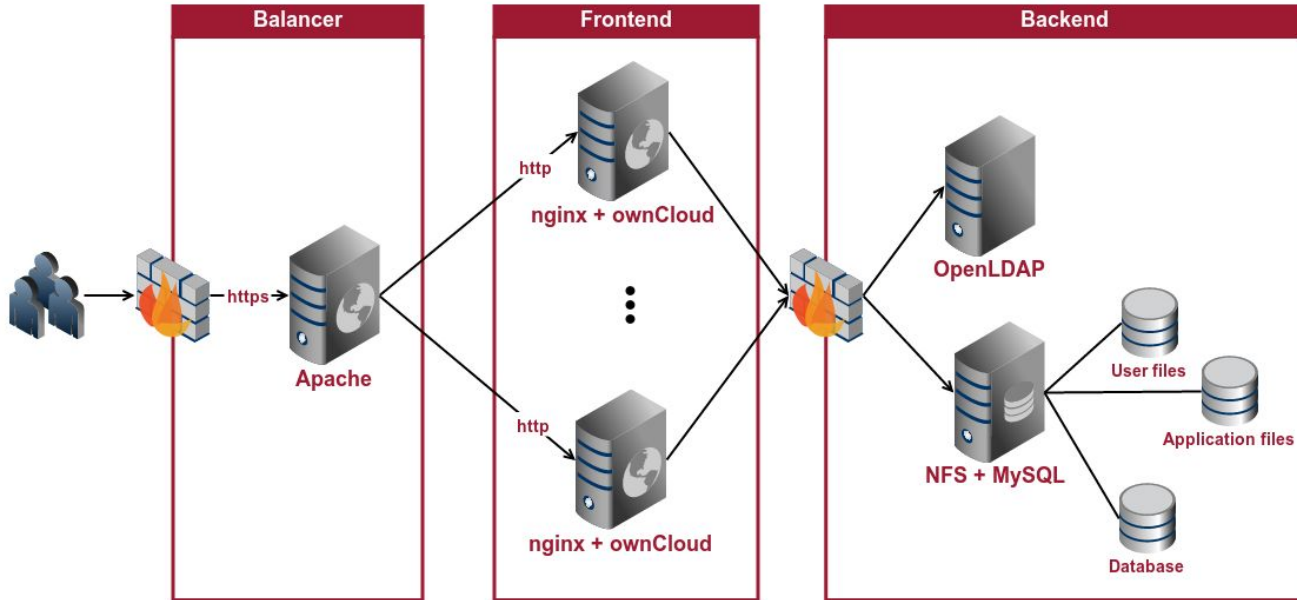




Fundamentos del Desarrollo Web (En nuestra local)



Fundamentos del Desarrollo Web (Caso de uso real)





Frontend

Parte visible de un sitio o aplicación web, es decir, todo lo que el usuario ve e interactúa directamente desde el navegador.

Contiene:

- HTML > Estructura
- CSS > Estilos visuales
- Javascript > Lógica



Frontend - Qué contiene?

- Menús, formularios, botones, imágenes, etc.
- Animaciones, validaciones, navegación entre páginas.
- Responsive.
- Accesibilidad para personas con discapacidad.



Frontend - Tipos basado en renderizado

1 - Renderizado en el servidor (Server-Side Rendering - SSR):

El servidor genera un HTML completo en cada solicitud y lo envía al navegador. Contenido cambia con frecuencia y necesito buen SEO.

Ejemplo: Blogs, ecommerce, portales de noticias.

Pros:

- Buena performance ideal.
- SEO optimizado.

Contras:

- Carga más lenta entre páginas.
- Requiere más recursos del servidor.

Frameworks: [Next.js](#) (modo SSR), Laravel, RoR, [ASP.NET](#) MVC



Frontend - Tipos basado en renderizado

2 - Renderizado en el client (Client-Side Rendering - CSR):

Se entrega al navegador un HTML vacío, el cual es completado con Javascript. El navegador construye la interfaz dinámicamente.

Ejemplo: Apps tipo dashboard, panel de usuario, SPA.

Pros:

- Navegación fluida.
- Separación clara entre frontend y backend.

Contras:

- Peor SEO (inicialmente el contenido no está en el HTML).
- Primera carga más lenta.

Frameworks: React (con Vite o CRA), Vue, Angular



Frontend - Tipos basado en renderizado

3 - Aplicación de una sola página (Single Page Application - SPA):

Es un tipo particular de CSR. Toda la app corre en una sola página index.html, y la navegación no recarga la página.

Ejemplo: Gmail, Trello, Google Drive.

Pros:

- UX fluida.
- Ideal para apps con muchas interacciones.

Contras:

- Manejo más complejo del estado y las rutas.
- SEO limitado (uso de técnicas de prerendering).



Frontend - Tipos basado en renderizado

4 - Aplicación web progresiva (Progressive Web App - PWA):

No es un método de renderizado en sí, es un enfoque que permite:

- funcionar offline
- se vea como una app móvil
- se instala en dispositivo

Ejemplo: Spotify web app.

Pros:

- Carga rápida.
- Experiencia tipo app.
- Funciona offline

Contras:

- Compleja de configurar.

Requiere: Service workers.



Frontend - Tipos basado en renderizado

5 - MPA (Multi-Page Application):

Ejemplo: Sitios tradicionales en php, WordPress, ASP.

6 - Static Site Generation (SSG):

Ejemplo: Blogs, landing pages, portfolios.

Herramientas: Astro, entre otros



Herramientas a utilizar



Visual Studio Code

<https://code.visualstudio.com/>

```
jim -- bash -- 103x20
Last login: Tue Sep 25 13:00:56 on ttys006
Jim-Hoskins-iMac:~ jim$ ls -la
total 8
drwxr-xr-x  13 jim  staff   442 Sep 25 13:00 .
drwxr-xr-x   6 root   admin  204 Sep 25 12:50 ..
-rw-r--r--   1 jim  staff    3 Sep 25 12:50 .CPUTextEncoding
drwx-----  2 jim  staff   68 Sep 25 12:52 .Trash
-rw-r--r--   1 jim  staff   57 Sep 25 13:00 .bash_history
drwx----- 10 jim  staff  340 Sep 25 12:57 Desktop
drwx-----  4 jim  staff  136 Sep 25 12:50 Documents
drwx-----  4 jim  staff  136 Sep 25 12:50 Downloads
drwx-----@ 33 jim  staff 1122 Sep 25 12:50 Library
drwx-----  3 jim  staff  102 Sep 25 12:50 Movies
drwx-----  3 jim  staff  102 Sep 25 12:50 Music
drwx-----  4 jim  staff  136 Sep 25 12:50 Pictures
drwxr-xr-x   5 jim  staff  170 Sep 25 12:50 Public
Jim-Hoskins-iMac:~ jim$ Where am i?
```

Línea de comandos



git

Git

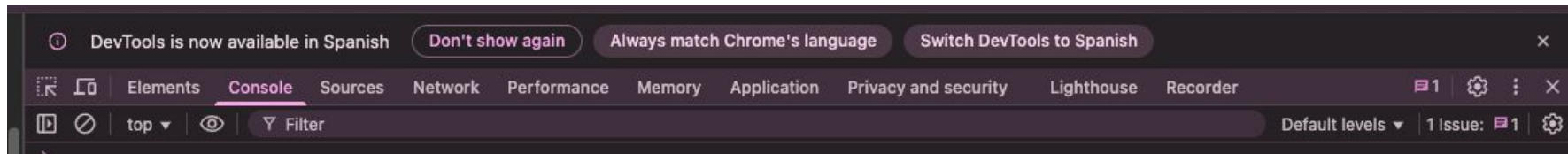


DevTools de navegadores

Herramientas integradas en todos los navegadores.

Para acceder, dentro del navegador:

- F12
- Click derecho -> Inspeccionar
- Configuración -> Más Herramientas -> Herramientas del Desarrollador (Chrome)





DevTools de navegadores

Pestaña	Función principal	¿Qué permite hacer?
Elements	Inspección de HTML y CSS	Ver, editar estructura y estilos en vivo
Console	Visualización de mensajes y errores de JavaScript	Ejecutar comandos JS, ver logs y advertencias
Sources	Depuración de código JavaScript	Colocar breakpoints, revisar archivos, debugging paso a paso
Network	Análisis de solicitudes y recursos de red	Ver peticiones HTTP, tiempos de carga, cabeceras



DevTools de navegadores

Performance	Análisis del rendimiento de la página	Medir FPS, repaints, scripting, tiempos de renderizado
Application	Gestión de almacenamiento local y recursos	Revisar cookies, localStorage, sessionStorage, cachés
Security	Revisión de certificados y políticas de seguridad	Verificar HTTPS, problemas de contenido mixto
Lighthouse	Auditoría de calidad (solo en Chrome)	Evaluar performance, accesibilidad, SEO, PWA
Memory	Análisis del uso de memoria	Detectar fugas de memoria, visualizar heap



DevTools de navegadores (Extensiones)

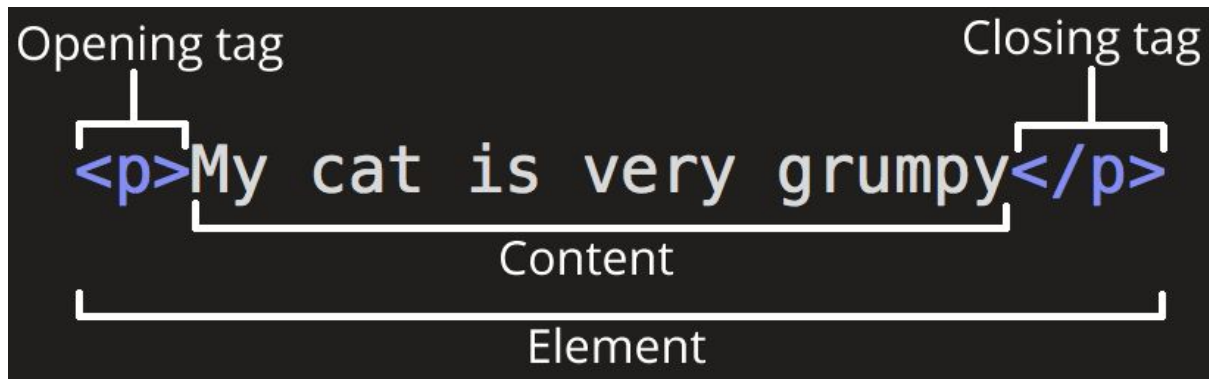
- React Developer Tools
- React Context Devtool



HTML (HyperText Markup Language)

- Lenguaje para estructurar páginas web. Donde se define una estructura de un documento usando **etiquetas** (<h1>, <p>, , etc) y es interpretado por navegadores para mostrar contenido visual al usuario.
- Es el **esqueleto de la página**: HTML con CSS (estilos) y JS (comportamiento) forman la base del desarrollo web.

Anatomía de un elemento



- **Etiqueta de apertura:** nombre del elemento (ej. `p` para el párrafo), envuelto en corchetes. Esta etiqueta de apertura indica donde inicia el elemento.
- **Contenido:** es el contenido del elemento. En el ej, es el texto del párrafo.
- **Etiqueta de cierre:** es la misma etiqueta de apertura, pero incluye barra diagonal del nombre de la etiqueta. Esto marca donde termina. No incluir puede generar una mala visualización, aunque algunos navegadores logran interpretar donde puede o no terminar un elemento, completando la estructura automáticamente (**NO SE DEBE CONFIAR EN ESTO NUNCA, Y ASEGURARSE DE CERRAR EL ELEMENTO**).



Anidamiento

```
HTML
```

```
<p>Mi gato es <strong>muy</strong> gruñón.</p>
```

Mi gato es **muy** gruñón.

```
HTML
```

```
<p>Mi gato es <strong>muy gruñón.</p></strong>
```

Mi gato es **muy gruñón.**

- Casos especiales

```
HTML
```

```

```



Atributos

Attribute

```
<p class="editor-note">My cat is very grumpy</p>
```

Un atributo debe tener:

- Espacio entre el atributo y el elemento.
- Nombre de atributo seguido de un signo igual.
- Un valor de atributo, envuelto con comillas de apertura y cierre. Hay casos particulares.



Anatomía de un documento HTML

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Título de la página</title>
  </head>
  <body>
    <h1>Encabezado principal</h1>
    <p>Párrafo de ejemplo en una página HTML.</p>
  </body>
</html>
```

- **<!DOCTYPE html>**: indica que es HTML5.
- **<html>**: elemento raíz del documento.
- **<head>**: metadatos, título, enlaces a estilos o scripts.
- **<body>**: contenido visible (texto, imágenes, enlaces, etc.)



Etiquetas comunes

- **Encabezados:** `<h1>--<h6>`
- **Párrafos:** `<p>`
- **Enlaces:** `Texto`
- **Imágenes:** ``
- **Listas:**
 - a. Ordenadas `...`
 - b. Desordenadas `...`
- **Divisiones genéricas:** `<div>`, `` (sin semántica) `()`.



Etiquetas semánticas

Son etiquetas que transmiten el **significado del contenido**, mejorando accesibilidad y SEO .

- **<header>**: cabecera del sitio o sección
- **<nav>**: bloque de navegación
- **<main>**: contenido principal de la página
- **<section>**: sección genérica de contenido
- **<article>**: contenido independiente (artículos, entradas)
- **<aside>**: contenido secundario (sidebar)
- **<footer>**: pie de página o sección
- **<figure>** y **<figcaption>**: imágenes con título/descripción ()
- Otras: **<mark>**, **<time>**, **<progress>**, **<details>**, **<summary>**

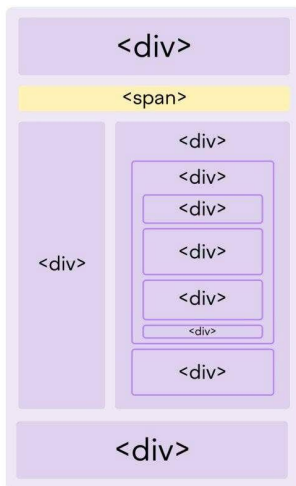


Etiquetas semánticas

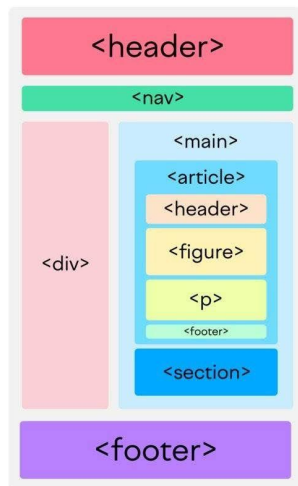


What Is Semantic HTML?

Non-Semantic HTML



Semantic HTML





Ejercicio #16

- Crear una carpeta con el nombre “Ejercicio_16”
- Abrir la Visual Studio y cargar la carpeta “Ejercicio_16”
- Crear un archivo “**index.html**”
- Agregar los siguientes elementos, en el orden dado:
 - Elemento “**header**”, contenido “**Soy una página**”
 - Elemento “**nav**”, dentro incluir lo siguiente:
 - Elemento “**a**”, contenido “**Inicio**”
 - Elemento “**a**”, contenido “**Productos**”
 - Elemento “**main**”, dentro incluir lo siguiente:
 - Elemento “**div**”, contenido “**Hola a todos!**”
 - Elemento “**footer**”, contenido “**Información de mi empresa**”
- Abrir la carpeta “Ejercicio_16” y hacer doble click en “**index.html**”

Resultado: Debe abrirse un navegador y se debe mostrar el contenido de “**index.html**”



Ejercicio #16.2

Tenemos que relacionar dos páginas.

- Crear un segundo archivo en Visual Studio, llamado **“products.html”**.
- Copiar el contenido desde **“index.html”**
- Cambiar el contenido del elemento **“div”** y poner **“Soy un listado de Productos”**
- En los archivos **“index.html”** y **“products.html”** deben modificar las etiquetas **“a”**, agregando el atributo **“href”**
- Para las etiqueta que es **“Inicio”**, el atributo **“href”** debe apuntar al archivo **“index.html”**
- Para las etiqueta que es **“Productos”**, el atributo **“href”** debe apuntar al archivo **“products.html”**

Resultado: Desde el navegador, si haces click en los enlaces, tienes que navegar desde una página a otra.



Forms

Un formulario HTML permite al usuario ingresar y enviar datos a un servidor o procesarlo en el cliente.

```
<form action="/procesar" method="POST">  
  <!-- elementos del formulario -->  
</form>
```



Elementos de un formulario

<code><input type="text"></code>	Campo de texto de una sola línea	Nombre: <code><input type="text"></code>
<code><input type="password"></code>	Campo para contraseñas (oculta el texto)	Contraseña: <code><input type="password"></code>
<code><input type="email"></code>	Valida que el contenido tenga formato de email	<code><input type="email"></code>
<code><input type="number"></code>	Permite solo números	<code><input type="number"></code>
<code><input type="checkbox"></code>	Casilla de verificación	<code><input type="checkbox"></code> Acepto términos
<code><input type="radio"></code>	Opción única dentro de un grupo	<code><input type="radio" name="color"></code> Rojo



Elementos de un formulario

<code><input type="submit"></code>	Botón para enviar el formulario	<code><input type="submit" value="Enviar"></code>
<code><input type="reset"></code>	Botón para limpiar los campos	<code><input type="reset" value="Borrar"></code>
<code><input type="date"></code>	Selector de fecha	<code><input type="date"></code>
<code><input type="file"></code>	Selector de archivos para subir	<code><input type="file"></code>
<code><textarea></code>	Área de texto multilínea	<code><textarea rows="4" cols="40"></code> <code></textarea></code>
<code><select></code> + <code><option></code>	Menú desplegable	<code><select><option>Argentina</option></code> <code></select></code>
<code><label></code>	Etiqueta para asociar texto con un control	<code><label for="email">Correo</label></code>



Ejemplo

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

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

  <label for="pais">País:</label>
  <select id="pais" name="pais">
    <option value="ar">Argentina</option>
    <option value="br">Brasil</option>
    <option value="uy">Uruguay</option>
  </select><br><br>

  <label><input type="checkbox" name="terminos" required> Acepto los términos</label><br><br>

  <input type="submit" value="Enviar">
</form>
```



Ejercicio #17

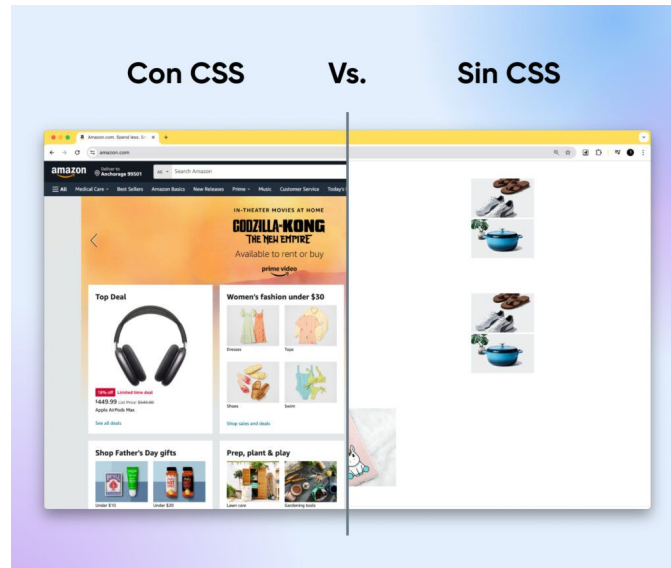
- Crear una carpeta con el nombre “**Ejercicio_17**”, abrir Visual Studio y cargar la carpeta creada
- Crear un archivo “**index.html**”
- Agregar formulario para realizar registro de usuario:
 - Elemento “**form**”, dentro incluir lo siguiente:
 - Elemento “**label**”, contenido “**Usuario:**”
 - Elemento “**input**” tipo texto, requerido.
 - Elemento “**label**”, contenido “**Email:**”
 - Elemento “**input**” tipo email, requerido.
 - Elemento “**label**”, contenido “**Contraseña:**”
 - Elemento “**input**”, tipo password, requerido.
 - Elemento “**input**” tipo submit

Resultado: Debe visualizar el formulario de registro con las validaciones en cada campo.

CSS (Cascading Style Sheets)

Hojas de Estilo en Cascada o **CSS** es un lenguaje de estilos que describe cómo debe renderizar un elemento en la pantalla.

```
1 h1 {  
2   color: red;  
3   font-size: 5em;  
4 }
```





Aplicar CSS

Forma	Cómo se usa	Ventajas	Desventajas
Inline	En el mismo elemento, usando el atributo “style”: <code><h1 style="color: red;">Hola</h1></code>	Rápido para pruebas	No reutilizable, difícil de mantener
Interno (style)	Dentro de una etiqueta “style” en el “head”: <code><head><style> h1 { color: red; } </style></head></code>	Útil para archivos simples o pruebas	No separa contenido de estilo
Externo	En un archivo “.css” referenciado: <code><link rel="stylesheet" href="estilos.css"></code>	Reutilizable, limpio y organizado	Requiere carga adicional



Aplicar CSS

Tipo de Selector	Ejemplo	Qué selecciona
Universal	<code>* { }</code>	Todos los elementos
Elemento	<code>p { }</code>	Todos los <code><p></code>
Clase	<code>.rojo { }</code>	Elementos con <code>class="rojo"</code>



Aplicar CSS

Tipo de Selector	Ejemplo	Qué selecciona
ID	<code>#principal { }</code>	Elemento con id="principal"
Agrupado	<code>h1, h2, p { }</code>	Todos los h1, h2, y p
Descendiente	<code>article p { }</code>	Todos los p dentro de article



Aplicar CSS

Tipo de Selector	Ejemplo	Qué selecciona
Hijo directo	<code>ul > li { }</code>	li hijos directos de ul
Hermano adyacente	<code>h1 + p { }</code>	Primer p que sigue a un h1
Atributo	<code>input[type="text"]</code>	Inputs de tipo texto



Aplicar CSS

Tipo de Selector

Ejemplo

Qué selecciona

Pseudo-clase

`a:hover`

Estado al pasar el mouse

Pseudo-elemento

`p::first-line`

La primera línea de cada párrafo



Ejercicio #18

- Crear una carpeta con el nombre “**Ejercicio_18**” y abrir la carpeta en Visual Studio
- Copiar los archivos “**index.html**” y “**products.html**” del ejercicio 16.
- Elemento “**header**”:
 - style=”border: 1px solid gray;”
- Elemento “**nav**”:
 - style=”border: 1px solid red; height: 30px;”
- Elemento “**main**”:
 - style=”border: 1px solid blue; height: 100px;”
- Elemento “**footer**”:
 - style=”border: 1px solid green; height: 30px;”
- Aplicar los mismos en archivo externo

Resultado: Al abrir el navegador y cargar “**index.html**” debe mostrarse los bordes en cada sección.



Cascada, especificidad y herencia

Cascada: significa que el orden de las reglas **SI** importan en CSS.

Especificidad: es el modo en que el navegador decide qué reglas aplicar si existen reglas con diferentes selectores y se aplican a un elemento específico.

Herencia: algunos valores de las propiedades CSS que se asignan a un padre, **PUEDE** o **NO** ser heredada por los hijos



Jerarquía de especificidad

Nivel	Tipo de selector	Ejemplo
1 (Mayor prioridad)	!important (sobrescribe todo, excepto estilos en línea con !important)	color: red !important;
2	Estilos en línea (atributo style en el HTML)	<p style="color: blue;">
3	Selectores de ID	#miElemento {}
4	Selectores de clase, atributo o pseudo-clase	.rojo {}, [type="text"], :hover
5	Selectores de elemento o pseudo-elemento	p {}, h1 {}, ::after
6 (Menor prioridad)	Selector universal y combinadores	* {}, div p, >



Diseño web

Encabezado

Visible en la parte superior de cada página de un sitio. Contiene información relevante para todas las páginas (como el nombre del sitio o el logo) y un sistema de navegación fácil de usar.

Contenido principal

Es el área más grande, contiene contenido único para la página actual.

Contenido secundario

1) Información complementaria del contenido principal; 2) información compartida entre un subconjunto de páginas; 3) sistema de navegación alternativo. De hecho, todo lo que no es absolutamente requerido por el contenido de la página principal.

Pie de página

Visible en la parte inferior de cada página de un sitio. Análogamente al encabezado contiene información global, en este caso menos llamativa como avisos legales o información de contacto.

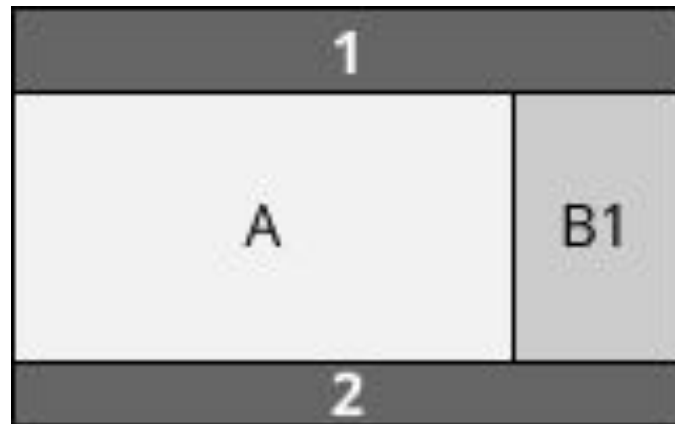
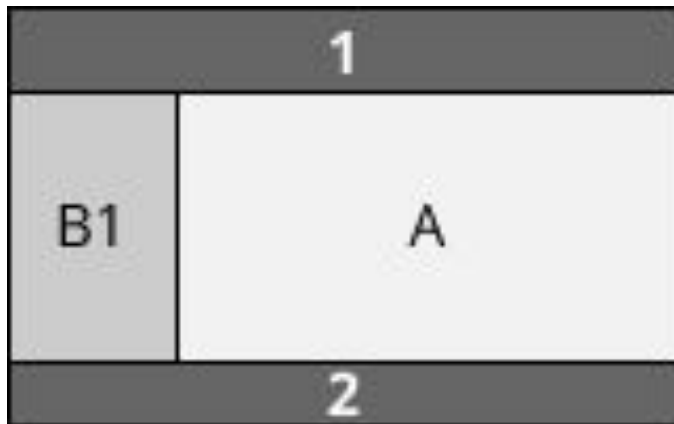


Diseño de 1 columna

1
A
B1
B2
2

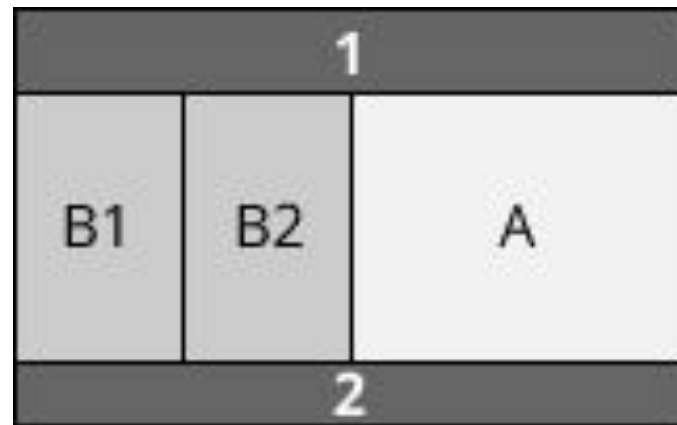
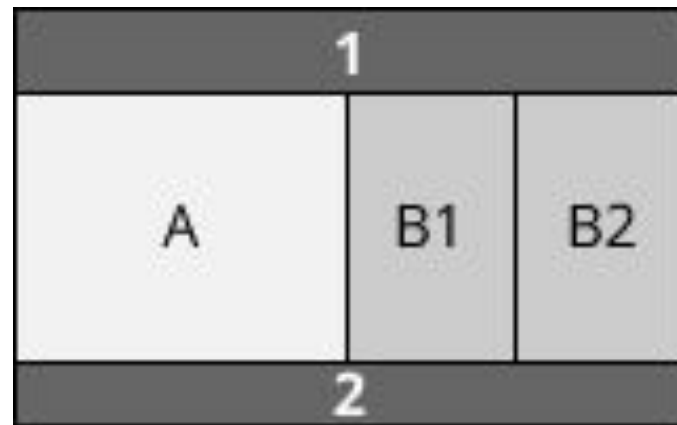
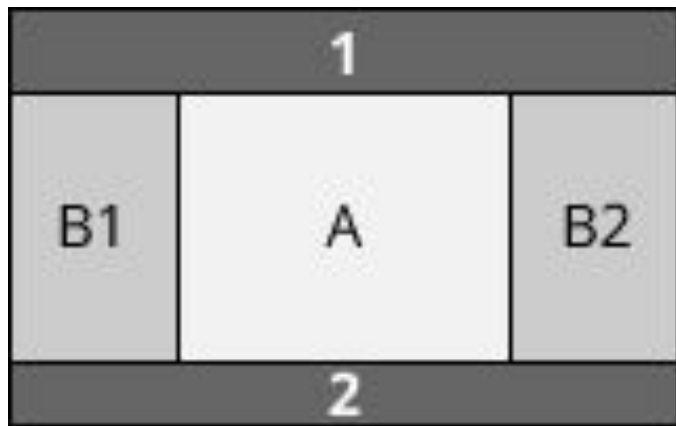


Diseño de 2 columnas



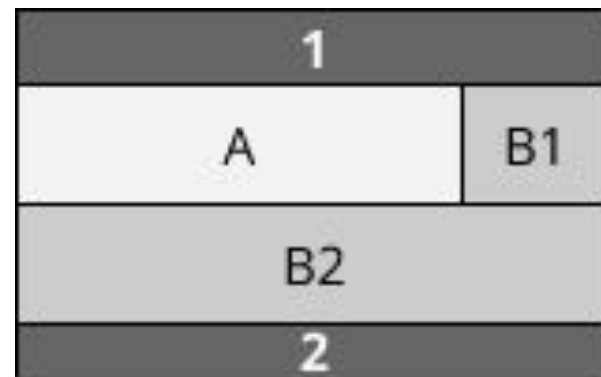
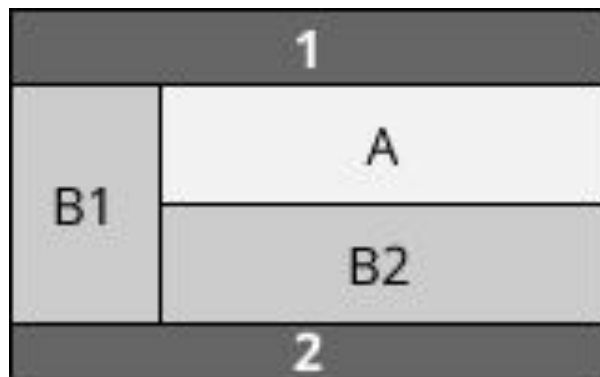
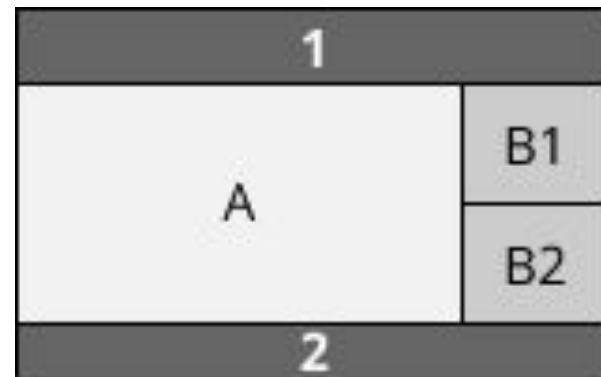
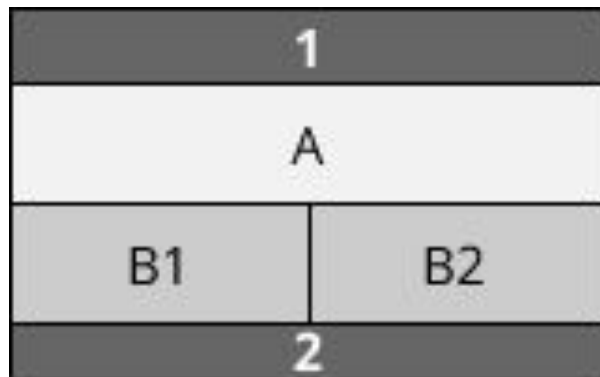


Diseño de 3 columnas

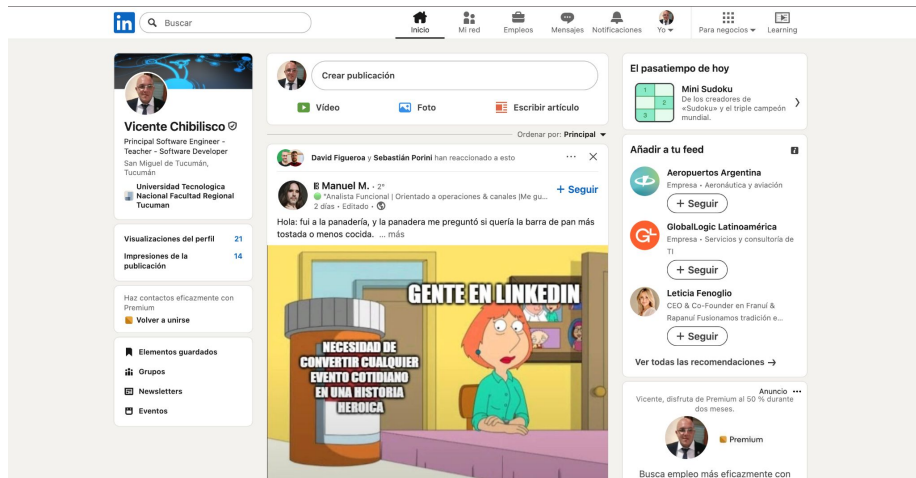




Combinaciones



Ejemplos



LA GACETA

Oferta ~~\$8.199~~ \$999/mes

SUSCRIBITE



Día del Niño, sin impacto en el consumo: las ventas cayeron pese a las promociones

La comercialización bajó un 0,3% respecto de 2024, cuando se había derrumbado un 14,4%. El resultado global no logró revertir la tendencia de estancamiento.



Ejemplos

Buscar productos, marcas y más...

Activa ahora con **50%OFF** **TOTAL**

Enviar a Vicente Barrio Judicial Manz...

Categorías ▾ Ofertas Cupones Supermercado Moda Mercado Play Vender Ayuda

GRATIS

vicente ... ▾ Mis compras Favoritos ▾

☒ **Productos de VECAT BRAND-AGENT** >
Tienda oficial Slime

☒ **Adaptador Matchmaker Freno Shimano A Shift...**

- 1 +

\$ 56.272
[Eliminar](#)
+5 disponibles

Envío
Gratis
Envío gratis
Aprovechá tu envío gratis agregando más productos de **VECAT BRAND-AGENT**.
[Ver más productos de este vendedor](#)

Resumen de compra
Producto **\$ 56.272**
Envío **Gratis**
[Ingresar código de cupón](#)
Total \$ 56.272

Continuar compra

Estoy buscando...

Enviar a Vicente Barrio Judicial Manzana B Casa 6 SN >

☒ **Productos de DINAMIC STORE** >
Tienda oficial

☒ **Portabiciqueta De Techo Aluminio Ce...**
[Eliminar](#)
1 u. ▾ **-47% \$199.999 \$ 105.379**

Envío **Gratis**

Aprovechá tu envío gratis agregando más productos de **DINAMIC STORE**.
[Ver más productos de este vendedor](#)

Producto **\$199.999 \$ 105.379**
Envío **Gratis**
[Ingresar código de cupón](#)
Total \$ 105.379

Continuar compra



Flexbox - Qué es?

- Un sistema de diseño **unidimensional**: organiza elementos en **fila (row)** o en **columna (column)**.
- Fue creado para **alinear y distribuir espacio** entre elementos de un contenedor.



Flexbox - Característica

- Dirección: controla si los elementos se alinean en **horizontal** o **vertical**.
- Alineación: centra fácilmente elementos en el eje principal y cruzado (justify-content, align-items).
- Espaciado: distribuye huecos automáticamente (space-between, space-around, gap).
- Flexibilidad: los elementos pueden **estirarse** o **encogerse** según el espacio (flex-grow, flex-shrink, flex-basis).



Flexbox - Ejemplo

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="stylesFlexbox.css">
</head>
<body>
  <div class="page">
    <header>Encabezado</header>
    <div class="container">
      <nav>Menú lateral</nav>
      <main>Contenido principal</main>
    </div>
    <footer>Pie de página</footer>
  </div>
</body>
</html>
```

```
body {
  margin: 0;
  height: 100vh;
}

.page {
  height: 100vh;
  display: flex;
  flex-direction: column;
}

.page header {
  background: #e3e3e3;
}

.page .container {
  flex: 1;
  display: flex;
}

.page .container nav {
  width: 200px;
  background: #cfcfcf;
}

.page .container main {
  flex: 1;
  background: #f4f4f4;
}

.page footer {
  background: #ddd;
}
```



Grid - Qué es?

- Un sistema de diseño **bidimensional**: organiza en **filas y columnas**.
- Ideal para construir **layouts completos** de una página.



Grid - Característica

- Define una cuadrícula con `grid-template-columns` y `grid-template-rows`.
- Permite posicionar elementos en **celdas específicas** con `grid-column` y `grid-row`.
- Soporta **alineación avanzada** y áreas con nombre (`grid-template-areas`).
- Se adapta muy bien a diseños **responsivos**.



Grid - Ejemplo

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="page">
    <header class="header">Encabezado</header>
    <div class="sidebar">Menú lateral</div>
    <main class="main">Contenido principal</main>
    <footer class="footer">Pie de página</footer>
  </div>
</body>
</html>
```

```
body {
  margin: 0;
  height: 100vh;
}

.page {
  display: grid;
  grid-template-columns: 200px 1fr;
  grid-template-rows: auto 1fr auto;
  grid-template-areas:
    "header header"
    "sidebar main"
    "footer footer";
  height: 100vh;
}

.header {
  grid-area: header;
  background: #e3e3e3;
}

.sidebar {
  grid-area: sidebar;
  background: #cfcfcf;
}

.main {
  grid-area: main;
  background: #f4f4f4;
}

.footer {
  grid-area: footer;
  background: #ddd;
}
```



Flexbox vs Grid - Diferencias

Característica	Flexbox (1D)	Grid (2D)
Dimensión	Una (fila o columna)	Dos (filas y columnas)
Uso ideal	Distribuir y alinear elementos en línea	Crear estructuras completas de página
Ejes	Principal + cruzado	Filas + columnas
Facilidad	Rápido para centrar y alinear	Mejor para layouts complejos
Ejemplo típico	Barra de navegación, botones, cards	Página con header, sidebar, contenido, footer



Flexbox vs Grid - Cuando usar cada uno

- **Flexbox** → cuando trabajás con un **grupo de elementos en fila o columna** (ej: menús, tarjetas, botones).
- **Grid** → cuando necesitás un **layout completo y estructurado** (ej: plantilla de página con cabecera, contenido y pie).



Diseño responsive - Fundamentos

1. Diseño fluido (fluid layouts)

- Usar unidades relativas como %, em, rem, vh, vw en lugar de valores fijos en px.
- Ejemplo: en lugar de width: 600px, usar width: 80%.

2. Imágenes y contenido flexibles

- Las imágenes se hacen **escalables** (max-width: 100%) para que no “rompan” el diseño.
- Videos y otros elementos multimedia deben adaptarse a su contenedor.

3. Media Queries

- Permiten **aplicar estilos diferentes** según el tamaño de la pantalla o características del dispositivo.
- Ejemplo: un menú que en escritorio se ve horizontal, pero en móvil se transforma en lista desplegable.



Metaetiqueta viewport

```
<meta name="viewport" content="width=device-width,initial-scale=1" />
```

Informa a los navegadores de los dispositivos móviles que deben establecer el ancho de la ventana gráfica al ancho del dispositivo y escalar el documento al 100%.



Media queries

Permite cambiar nuestras reglas de CSS cuando cumplen determinadas condiciones

```
@media media-type and (media-feature-rule) {  
    /* Las reglas CSS van aquí */  
}
```

Consta de:

- Un tipo de medio, que le dice al navegador para qué tipo de medio es este código (por ejemplo, impresión o pantalla).
- Una expresión de medio, que es una regla, o prueba que debe aprobarse para que se aplique el CSS contenido.
- Un conjunto de reglas CSS que se aplicarán si la prueba pasa y el tipo de medio es correcto.

https://developer.mozilla.org/es/docs/Learn_web_development/Core/CSS_layout/Media_queries



Media queries - media type

Tipo de medio	Descripción	Ejemplo de uso
all	Es el valor por defecto . Se aplica a todos los dispositivos y tipos de medios (pantallas, impresoras, lectores de voz, etc.). Normalmente no es necesario escribirlo porque ya está implícito.	css @media all and (max-width: 768px) { ... }
print	Se aplica cuando el documento se envía a impresoras o se visualiza en modo print preview (vista previa de impresión). Sirve para adaptar el diseño a formato papel.	css @media print { body { font-size: 12pt; } }
screen	Se aplica cuando el contenido se muestra en pantallas (computadoras, móviles, tablets, smart TVs, etc.). Es el más usado para diseño responsive.	css @media screen and (max-width: 600px) { ... }



Media queries - media type

```
/* Aplica a todas las pantallas pequeñas */
@media screen and (max-width: 768px) {
  body {
    background: lightblue;
  }
}

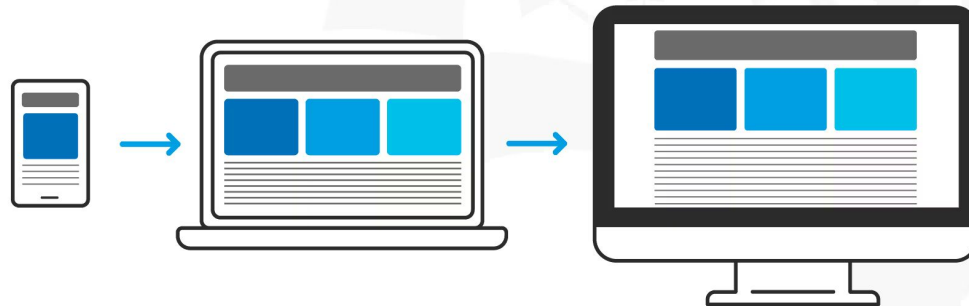
/* Estilos especiales al imprimir */
@media print {
  body {
    background: white;
    font-size: 12pt;
  }
}
```



Mobile first

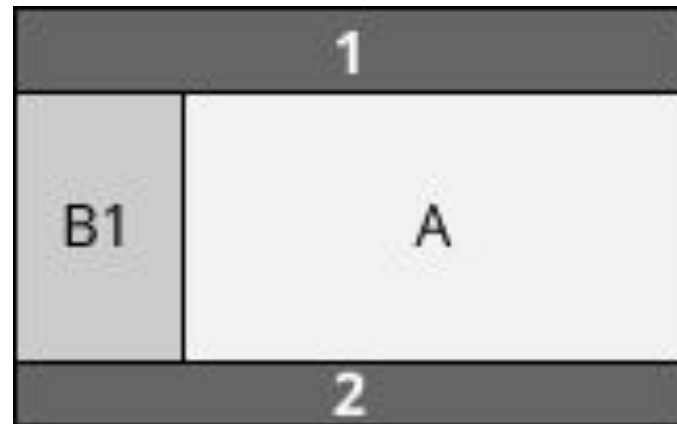
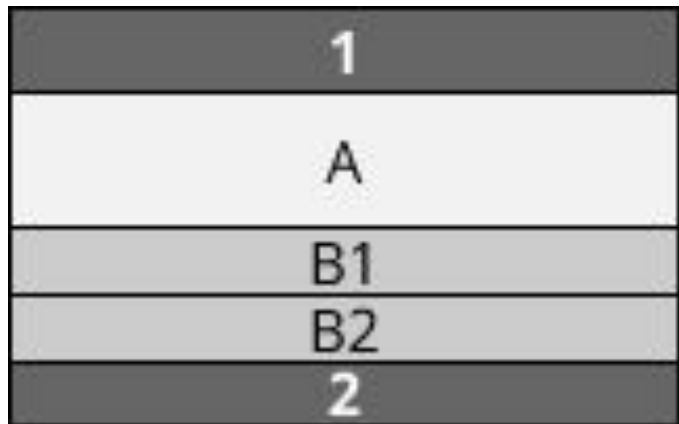
Enfoque que se centra en la creación de una experiencia destinada a dispositivos móviles.

Mobile-First Design





Ejercicio #19





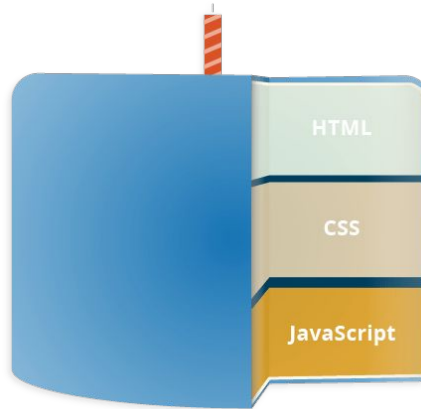
Ejercicio #19 - Usar Grid

- Crear dos páginas, login.html y dashboard.html
- login.html debe tener un form, con user y password requerido, y un botón que redirecciona a dashboard.html
- En login.html, el form, version mobile, debe ocupar toda la pantalla, haciendo fácil la lectura. Versión escritorio debe ocupar el centro de la pantalla.
- En dashboard.html, version mobile, solo debe tener header, main y footer. El header debe tener el botón para expandir el panel de navegación. En escritorio, debe ser un layout con header, nav, main y footer, donde nav y main estan ordenados horizontalmente, nav con 200px de width. Usar grid.



Javascript

Lenguaje de programación interpretado, multiparadigma (POO, funcional y procedural/imperativa) y dinámico, que permite implementar funciones complejas en páginas web, o del lado del servidor.





Motor de JS

Analizan y ejecutan código JS. Los motores modernos usan utilizan compilación justo a tiempo (JIT), donde convierte código JS a código máquina.

Puedes encontrar motores en los navegadores web, y para servidores como NodeJS.

V8

Motor de Google escrito en C++. Presente en Chrome y NodeJs.

SpiderMonkey

Motor de Mozilla utilizado en Firefox. Escrito en C++ y Rust.

JavaScriptCore

Permite ejecutar JavaScript desde aplicaciones Swift, Objective-C y C.



Motor de Renderizado

Es la parte del navegador que transforma el HTML, CSS y otros recursos de una página web en una representación visual dentro de la pantalla.

Blink	Gecko	WebKit	Trident (MSHTML)
Desarrollado por Google como parte de Chromium	Motor de Mozilla, utilizado en Firefox y FirefoxOS	Registrado por Apple, y utilizado en Safari	Era el motor de Internet Explorer. Actualmente reemplazado por EdgeHTML.



ECMAScript

Es el estándar técnico de Javascript. Define características y la sintaxis del lenguaje para garantizar que los diferentes motores.



Mini ejemplo

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Intro JS</title>
</head>
<body>
  <h1>Hola JavaScript</h1>
  <script>
    console.log("Hola desde la consola");
    alert("Hola desde un alert!");
  </script>
</body>
</html>
```



Variables

Característica	var	let	const
Scope	Función / Global	Bloque {}	Bloque {}
Redeclaración	✓ Sí	✗ No	✗ No
Reasignación	✓ Sí	✓ Sí	✗ No
Hoisting	Sí (inicializa undefined)	Sí (pero TDZ)	Sí (pero TDZ)
Uso recomendado	Evitar	Variables mutables	Constantes / refs

Hoisting: **mueve las declaraciones** (de variables y funciones) **al inicio de su scope** antes de ejecutar el código
TDZ: Temporal dead zone



Tipos de datos primitivos

Tipo	Ejemplo	Nota clave
string	"Hola", 'Mundo', `texto`	Texto
number	42, 3.14, NaN, Infinity	Números
bigint	9007199254740991n	Enteros grandes
boolean	true, false	Lógico
undefined	let x;	Sin valor asignado
null	let y = null;	Ausencia intencional
symbol	Symbol("id")	Valor único

Nota: A partir de ES2022 también existe **`typeof null === "object"`**, lo cual es un *bug histórico* en JavaScript.



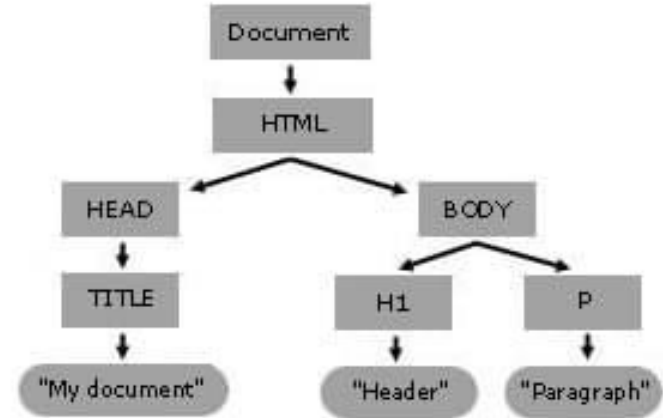
Modelo de Objetos del Documento (DOM)

Es la representación de los objetos que confirman la estructura y el contenido de un documento en la web.

Representa la página para que los programas puedan modificar la estructura, el estilo y contenido del documento. El DOM representa el documento como nodos y objetos.

Usando de Objetos del Documento (DOM)

```
<html lang="en">
  <head>
    <title>My Document</title>
  </head>
  <body>
    <h1>Header</h1>
    <p>Paragraph</p>
  </body>
</html>
```



https://developer.mozilla.org/es/docs/Web/API/Document_Object_Model/Using_the_Document_Object_Model
<https://developer.mozilla.org/es/docs/Web/API/Document>



Ejercicio #20 - Crear una tabla

- Crear una página index.html
- Introducir un botón, en el evento onclick debe ejecutar una función de JS.
- Usando la API Document, se creará los elementos siguiendo esta jerarquía:
 - table
 - thead
 - tr
 - th, contenido "Número de orden"
 - th, contenido "Nombre de Cliente"
 - tbody
 - tr
 - td, contenido "#1"
 - td, contenido "Julian"
 - tr
 - td, contenido "#2"
 - td, contenido "Maria"



Eventos

Acción o suceso que ocurre en la página web, y JS puede detectar y manejar.

El navegador crea y otorga un objeto Event con información de lo que pasó:

- Tipo.
- Elemento que lo generó.
- Tecla presionada.
- etc



Eventos de mouse

- click - click del mouse
- dblclick - doble click.
- mouseover / mouseout - entrar / salir de un elemento.
- mousemove - mover el mouse.
- mousedown / mouseup / presionar / soltar botón del mouse.

```
button.addEventListener("click", () => {  
    alert("Hiciste clic en el botón!");  
});
```



Eventos de teclado

- `keydown` - se presiona una tecla.
- `keyup` - suelta una tecla.
- `keypress` - (obsoleto, usar `keydown/keyup`).

```
document.addEventListener("keydown", (event) => {  
    console.log("Tecla presionada:", event.key);  
});
```



Eventos de formulario

- submit - cuando se envía un formulario.
- change - cuando cambia el valor de un input/select.
- input - cuando se escribe un campo.
- focus / blur - cuando un campo recibe o pierde foco.

```
form.addEventListener("submit", (e) => {  
  e.preventDefault(); // evita que se recargue la página  
  console.log("Formulario enviado!");  
});
```



Eventos de ventana/documento

- load - cuando la página termino de cargar.
- resize - cuando cambia de tamaño la ventana.
- scroll - cuando se hace scroll.
- DOMContentLoaded - cuando el DOM está listo (antes de que se carguen las imagenes).

```
window.addEventListener("resize", () => {  
  console.log("La ventana cambió de tamaño");  
});
```



Eventos - otros

- Portapapeles
- Drag & Drop
- Multimedia
- Red
- Dispositivos
- Animación y transición
- DOM



Ejercicio #21 - Navbar

- Clonar repositorio <https://github.com/vchibilisco/Dsw2025Ei21>
- Utiliza JavaScript para detectar el clic en el botón de menú.
- Al hacer clic, muestra u oculta el menú de navegación.
- Asegúrate de que la funcionalidad solo se active en resoluciones móviles.



Objetos

Es una colección de propiedades, donde cada propiedad es una combinación de un nombre (clave) y un valor. Es fundamental saber trabajar con objetos, ya que son usados para modelar y manipular

Nombre del
objeto

```
const miAuto = {  
  color: "rojo",  
  marca: "Toyota",  
  modelo: "Prius",  
}
```

llave valor

Propiedades



Objetos - Creación

Literal

```
let persona = {  
  nombre: "Juan",  
  edad: 30,  
  profesion: "Desarrollador"  
};
```

Constructor

```
let persona = new Object();  
persona.nombre = "Juan";  
persona.edad = 30;  
persona.profesion = "Desarrollador";
```

Clases

```
class Persona {  
  constructor(nombre, edad, profesion) {  
    this.nombre = nombre;  
    this.edad = edad;  
    this.profesion = profesion;  
  }  
}  
  
let juan = new Persona("Juan", 30, "Desarrollador");
```



Objetos - Acceso a propiedades

Notación de Punto

```
console.log(persona.nombre); // "Juan"
```

Notación de Corchetes

```
console.log(persona["edad"]); // 30
```



Objetos - Métodos

```
let persona = {  
  nombre: "Juan",  
  saludar: function() {  
    console.log("Hola, mi nombre es " + this.nombre);  
  }  
};  
  
persona.saludar(); // "Hola, mi nombre es Juan"
```



Objetos - Funciones más usadas

```
console.log(Object.keys(persona)); // ["nombre", "edad", "profesion"]
```

```
console.log(Object.values(persona)); // ["Juan", 30, "Desarrollador"]
```

```
console.log(Object.entries(persona));  
// [["nombre", "Juan"], ["edad", 30], ["profesion", "Desarrollador"]]
```



Objetos - Funciones más usadas

```
let objetoDestino = {a: 1};  
let objetoOrigen = {b: 2, c: 3};  
Object.assign(objetoDestino, objetoOrigen);  
console.log(objetoDestino); // {a: 1, b: 2, c: 3}
```

```
let persona = { nombre: "Juan" };  
Object.freeze(persona);  
persona.nombre = "Carlos"; // No cambia el valor  
console.log(persona.nombre); // "Juan"
```

```
let persona = { nombre: "Juan" };  
Object.seal(persona);  
persona.edad = 30; // No se puede añadir  
persona.nombre = "Carlos"; // Se puede modificar  
console.log(persona); // { nombre: "Carlos" }
```



Function vs Arrow

Característica

Function tradicional

Arrow function

Sintaxis

Verbosa, requiere function

Más compacta, usa =>

this

Dinámico → depende de cómo se invoque la función (objeto, call, apply, bind)

Léxico → hereda el this del scope donde fue creada

arguments

Disponible automáticamente como objeto iterable

✗ No existe → usar ...args (rest parameters)

Constructores (new)

✓ Se puede usar como constructor (ej: new Persona())

✗ No puede usarse como constructor (lanza error)

Métodos en objetos

✓ Ideal → this apunta al objeto

✗ Problema → this no apunta al objeto, sino al contexto exterior

Clases

Usado en métodos por defecto

Útil en propiedades de clase para *auto-bind* en React

Hoisting

✓ Se eleva completa (se puede usar antes de declararse)

✗ Se comporta como variable (const / let) → no existe antes de la declaración



Function vs Arrow

Característica

Function tradicional

Arrow function

Objeto prototipo

Crea un prototipo asociado (usado en POO clásica)

✗ No tiene prototipo

Uso recomendado

Constructores, métodos de objeto, cuando se necesite this dinámico o arguments

Callbacks, funciones cortas, promesas, funciones dentro de funciones

Legibilidad

Más verbosa, más clara en definiciones largas

Más concisa, ideal en callbacks y expresiones inline

Return implícito

No → se debe escribir return explícitamente

✓ Puede omitir return si el cuerpo es una sola expresión

Manejo de super (en clases)

Depende del contexto donde se invoque

Usa el super del contexto padre (igual que this)

Performance

Leve diferencia: funciones tradicionales optimizan mejor en engines antiguos

Similar rendimiento en engines modernos, pero no crean contexto propio

Function vs Arrow

```
// Diferencia en `this`  
const persona = {  
  nombre: "Ana",  
  saludarTrad: function () { console.log("Hola soy " + this.nombre); },  
  saludarArrow: () => console.log("Hola soy " + this.nombre)  
};
```

```
persona.saludarTrad(); // ✓ Hola soy Ana  
persona.saludarArrow(); // ✗ Hola soy undefined
```

```
// Diferencia en arguments  
function sumarTrad() {  
  console.log(arguments); // ✓ [2, 3, 4]  
}  
sumarTrad(2, 3, 4);
```

```
const sumarArrow = (...args) => {  
  console.log(args); // ✓ [2, 3, 4] con rest parameter  
};  
sumarArrow(2, 3, 4);
```

```
// Diferencia en constructores  
function Animal(nombre) {  
  this.nombre = nombre;  
}  
const perro = new Animal("Firulais"); // ✓ funciona  
  
const AnimalArrow = (nombre) => { this.nombre = nombre; };  
// const gato = new AnimalArrow("Michi"); // ✗ Error
```




Array

Estructura de datos que permite almacenar múltiples valores en una sola variable. Los arrays pueden contener cualquier tipo de dato, incluido otros arrays, objetos, números, cadenas y valores booleanos.



Array - Características

Los arrays son indexados, es decir, cada elemento tiene un índice asociado, y comienza en 0.

```
let numeros = [10, 20, 30];  
console.log(numeros[0]); // 10  
console.log(numeros[2]); // 30
```



Array - Creación

Literal

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

Constructor Array

```
let numeros = new Array(10, 20, 30);
```



Array - Modificar elemento usando índice

```
frutas[1] = "Mango";  
console.log(frutas); // ["Manzana", "Mango", "Naranja"]
```



Array - Funciones más usadas

push

```
let numeros = [1, 2, 3];  
numeros.push(4);  
console.log(numeros); // [1, 2, 3, 4]
```

pop

```
let numeros = [1, 2, 3];  
let ultimo = numeros.pop();  
console.log(ultimo); // 3  
console.log(numeros); // [1, 2]
```

shift

```
let numeros = [1, 2, 3];  
let primero = numeros.shift();  
console.log(primero); // 1  
console.log(numeros); // [2, 3]
```

unshift

```
let numeros = [2, 3];  
numeros.unshift(1);  
console.log(numeros); // [1, 2, 3]
```



Array - Funciones más usadas

length

```
let frutas = ["Manzana", "Banana"];  
console.log(frutas.length); // 2
```

forEach

```
let frutas = ["Manzana", "Banana", "Naranja"];  
frutas.forEach(function(fruta) {  
    console.log(fruta);  
});  
// "Manzana"  
// "Banana"  
// "Naranja"
```

map

```
let numeros = [1, 2, 3];  
let dobles = numeros.map(function(num) {  
    return num * 2;  
});  
console.log(dobles); // [2, 4, 6]
```



Array - Funciones más usadas

filter

```
let numeros = [1, 2, 3, 4, 5];  
let pares = numeros.filter(function(num) {  
    return num % 2 === 0;  
});  
console.log(pares); // [2, 4]
```

find

```
let numeros = [1, 2, 3, 4, 5];  
let encontrado = numeros.find(function(num) {  
    return num > 3;  
});  
console.log(encontrado); // 4
```

reduce

```
let numeros = [1, 2, 3, 4];  
let suma = numeros.reduce(function(accumulator, valorActual) {  
    return accumulator + valorActual;  
}, 0);  
console.log(suma); // 10
```





Ejercicio #22 - Tabla

- Tomar como base ejercicio #21
- Descargar json de este repositorio <https://github.com/vchibilisco/Dsw2025Ej22>
- Generar una tabla basado en el json. Debe mostrar todos los productos, las columnas serán “Código”, “Nombre”, “Stock”, “Vencimiento”.
- Arriba de la tabla se debe agregar un buscador por nombre, con un botón. El usuario debe hacer click en el botón, y se debe filtrar la tabla basado en el criterio de búsqueda.





Promise

Una **Promise** (promesa) es un objeto en JavaScript que representa un **proceso asíncrono** que puede:

-  **resolverse** → cuando termina bien.
-  **rechazarse** → cuando ocurre un error.

Se usa para manejar operaciones que tardan un tiempo (consultas a API, lectura de archivos, temporizadores, etc).

```
const promesa = new Promise((resolve, reject) => {  
  let exito = true;  
  
  setTimeout(() => {  
    if (exito) {  
      resolve("La operación fue exitosa     } else {  
      reject("Ocurrió un error     }  
  }, 2000); // simula una tarea que demora 2 segundos  
});  
  
promesa  
  .then(resultado => console.log(resultado)) // si se cumple  
  .catch(error => console.log(error));      // si falla
```



async/await

Es una **forma más clara y legible** de trabajar con Promises.

- `async` se coloca antes de la función → significa que la función devolverá una Promise.
- `await` se usa dentro de una función `async` → hace que el código “espere” el resultado de una Promise **sin bloquear** el programa.

```
const obtenerDatos = async () => {
  try {
    console.log("Obteniendo datos...");
    const datos = await new Promise((resolve) => {
      setTimeout(() => resolve(["Producto 1", "Producto 2"]), 2000);
    });
    console.log("Datos recibidos:", datos);
  } catch (error) {
    console.log("Error:", error);
  }
};

obtenerDatos();
```



fetch

- Es una **función nativa de JavaScript** (incluida en los navegadores modernos y Node.js con node-fetch o undici) que permite hacer **peticiones HTTP**.
- Devuelve una **Promise**, por lo que se trabaja con `.then/catch` o `async/await`.
- Se usa para **traer datos de un servidor, API o archivo** (ejemplo: JSON).

```
fetch(url, opciones)
  .then(respuesta => respuesta.json()) // convertir a JSON
  .then(data => console.log(data))     // usar los datos
  .catch(error => console.error(error)); // manejar errores
```



fetch - GET

```
fetch("https://jsonplaceholder.typicode.com/posts/1")  
  .then(res => res.json())  
  .then(data => console.log("Post:", data))  
  .catch(err => console.error("Error:", err));
```

```
async function obtenerPosts() {  
  try {  
    const respuesta = await fetch("https://jsonplaceholder.typicode.com/posts");  
    const data = await respuesta.json();  
    console.log("Lista de posts (GET):");  
    console.log(data.slice(0, 5)); // muestro los primeros 5  
  } catch (error) {  
    console.error("Error en GET:", error);  
  }  
}  
  
obtenerPosts();
```



fetch - POST

```
fetch("https://jsonplaceholder.typicode.com/posts", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({
    titulo: "Nuevo producto",
    stock: 30
  })
})
.then(res => res.json())
.then(data => console.log("Creado:", data))
.catch(err => console.error("Error:", err));
```

```
async function crearPost() {
  try {
    const respuesta = await fetch("https://jsonplaceholder.typicode.com/posts", {
      method: "POST",
      headers: {
        "Content-Type": "application/json"
      },
      body: JSON.stringify({
        titulo: "Nuevo producto",
        stock: 30
      })
    });

    const data = await respuesta.json();
    console.log("Resultado del POST:");
    console.log(data);
  } catch (error) {
    console.error("Error en POST:", error);
  }
}

crearPost();
```



Forms y validaciones HTML

```
<form id="miFormulario">
  <label>Email:</label>
  <input type="email" id="email" required>

  <label>Contraseña:</label>
  <input type="password" id="password" required>

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



Forms y validaciones con Javascript

```
<form id="miFormulario">
  <label>Email:</label>
  <input type="text" id="email">

  <label>Edad:</label>
  <input type="number" id="edad">

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

<p id="errores" style="color:red;"></p>

<script>
  const form = document.getElementById("miFormulario");
  const errores = document.getElementById("errores");

  form.addEventListener("submit", (e) => {
    e.preventDefault(); // evita que el form se envíe automáticamente

    const email = document.getElementById("email").value;
    const edad = parseInt(document.getElementById("edad").value);
    let mensajes = [];

    // validación de email
    if (!email.includes("@")) {
      mensajes.push("El email no es válido");
    }

    // validación de edad
    if (isNaN(edad) || edad < 18) {
      mensajes.push("Debes tener al menos 18 años");
    }

    // mostrar errores o enviar
    if (mensajes.length > 0) {
      errores.innerText = mensajes.join(", ");
    } else {
      errores.innerText = "";
      alert("Formulario enviado con éxito ✅");
      form.reset();
    }
  });
</script>
```



Ejercicio #23 - Formulario

- Tomar como base ejercicio #22
- Agregar un archivo HTML nuevo llamado product.html. Será una copia de dashboard.html, solamente lo básico para que el dashboard funcione
- Agregar campos crear un producto:
 - Label e input para SKU (requerido)
 - Label e input para Código Único de Identificación (requerido)
 - Label e input para Nombre (requerido)
 - Label e input para Description (opcional)
 - Label e input para Precio unitario (no puede ser menor a 0)
 - Label e input para Stock (no puede ser menor a 0)
- Con Javascript tomar los valores de cada input, crear un objeto, y mostrarlo por consola.
- Las validaciones pueden ser con HTML o Javascript. Si eligen Javascript, deben mostrar mensaje de error.



Almacenamiento en el browser

Método	Capacidad aprox.	Persistencia	Tipo de datos	Acceso	Uso típico
Cookies	~4 KB por cookie	Hasta que expire (fecha o sesión)	Texto (string, clave=valor)	Sincrónico (se envían al servidor en cada request)	Sesiones de usuario, recordar login, datos que necesita el servidor.
localStorage	~5–10 MB (depende del navegador)	Persistente (permanece hasta que el usuario lo borre)	Texto (strings, aunque se puede usar JSON.stringify para objetos)	Sincrónico	Guardar configuraciones, preferencias de usuario, temas, etc.
sessionStorage	~5 MB	Solo dura mientras la pestaña esté abierta	Texto (strings, igual que localStorage)	Sincrónico	Datos temporales por pestaña, como un carrito de compras mientras no se cierre la ventana.
IndexedDB	Cientos de MB o más (muy grande)	Persistente	Objetos complejos (soporta índices, transacciones, binarios, blobs)	Asíncrono	Aplicaciones web grandes: apps offline, almacenamiento masivo (ej: correos, notas, catálogos).



Almacenamiento en el browser

Método	Capacidad aprox.	Persistencia	Tipo de datos	Acceso	Uso típico
Cookies	~4 KB por cookie	Hasta que expire (fecha o sesión)	Texto (string, clave=valor)	Sincrónico (se envían al servidor en cada request)	Sesiones de usuario, recordar login, datos que necesita el servidor.
localStorage	~5–10 MB (depende del navegador)	Persistente (permanece hasta que el usuario lo borre)	Texto (strings, aunque se puede usar JSON.stringify para objetos)	Sincrónico	Guardar configuraciones, preferencias de usuario, temas, etc.
sessionStorage	~5 MB	Solo dura mientras la pestaña esté abierta	Texto (strings, igual que localStorage)	Sincrónico	Datos temporales por pestaña, como un carrito de compras mientras no se cierre la ventana.
IndexedDB	Cientos de MB o más (muy grande)	Persistente	Objetos complejos (soporta índices, transacciones, binarios, blobs)	Asíncrono	Aplicaciones web grandes: apps offline, almacenamiento masivo (ej: correos, notas, catálogos).



Almacenamiento en el browser

```
// Crear cookie
document.cookie = "usuario=Vicente; expires=Fri, 31 Dec 2025 23:59:59 GMT";

// Leer cookies
console.log(document.cookie);
```

```
// Guardar
localStorage.setItem("usuario", "Vicente");

// Leer
console.log(localStorage.getItem("usuario")); // Vicente
```

```
sessionStorage.setItem("tema", "oscuro");
console.log(sessionStorage.getItem("tema")); // oscuro
```



Almacenamiento en el browser

```
const request = indexedDB.open("MiBaseDeDatos", 1);

// Se ejecuta la primera vez o cuando cambia la versión
request.onupgradeneeded = function (event) {
  const db = event.target.result;
  // Creamos un almacén (como una tabla)
  const store = db.createObjectStore("productos", { keyPath: "id" });
  // Creamos un índice para búsquedas por nombre
  store.createIndex("nombreIndex", "nombre", { unique: false });
};

request.onsuccess = function (event) {
  console.log("Base de datos abierta con éxito ✅");
};

request.onerror = function (event) {
  console.error("Error al abrir la base de datos ❌", event);
};
```

```
function agregarProducto(producto) {
  const request = indexedDB.open("MiBaseDeDatos", 1);

  request.onsuccess = function (event) {
    const db = event.target.result;
    const transaction = db.transaction("productos", "readwrite");
    const store = transaction.objectStore("productos");

    store.add(producto);

    transaction.oncomplete = () => console.log("Producto agregado ✅");
    transaction.onerror = (err) => console.error("Error:", err);
  };
}

// Ejemplo de uso
agregarProducto({ id: 1, nombre: "Leche", stock: 25 });
agregarProducto({ id: 2, nombre: "Arroz", stock: 100 });
```

```
function obtenerTodos() {
  const request = indexedDB.open("MiBaseDeDatos", 1);

  request.onsuccess = function (event) {
    const db = event.target.result;
    const transaction = db.transaction("productos", "readonly");
    const store = transaction.objectStore("productos");

    const query = store.getAll();
    query.onsuccess = () => {
      console.log("Todos los productos:", query.result);
    };
  };
}

obtenerTodos();
```



Ejercicio #24 - Almacenamiento

- Tomar como base ejercicio #23
- Se debe tener un archivo JavaScript donde realizará todas las operaciones con localStorage.
- Se debe inicializar un array “products” vacío.
- La tabla productos se debe completar con los datos del array “products”.
- Desde el formulario, se debe agregar un objeto al array “products”.
- No guardar productos con SKU duplicados.
- El filtrado de la tabla debe funcionar obteniendo datos desde localStorage

Plus: Implementar paginación en la tabla