- ## Introducción a HTML
    - Vimos cómo estructurar un documento con etiquetas básicas.
    - La **web semántica** nos ayuda a dar sentido al contenido con etiquetas correctas (`<header>`, `<main>`, `<section>`, etc.).
- **Estructura de árbol del documento HTML (DOM)**
    - Todo documento HTML se organiza jerárquicamente como un árbol.
    - Esto facilita entender la relación entre elementos (padres, hijos, hermanos).
- **Herramientas para desarrollo**
    - **Emmet** en VSCode: permite escribir HTML y CSS más rápido con atajos.
    - **Live Server**: extensión que recarga el navegador automáticamente al guardar cambios.
- **Selectores de CSS**
    - **Por etiqueta/tag** → `p { }`
    - **Por clase/class** → `.mi-clase { }`
    - **Por id** → `#mi-id { }`
    - También usamos el atributo **style** en línea como alternativa rápida (aunque no es la mejor práctica a largo plazo).
- **Atributos clave en HTML**
    - `id` → identifica un único elemento.
    - `class` → se usa para agrupar y aplicar estilos comunes a varios elementos.
    - `style` → permite escribir CSS directamente en la etiqueta (ej: `<p style="color:red;">`).
- **Tarea sugerida / investigación personal**
    - Revisar más sobre **CSS**.
    - Estudiar la diferencia entre **display: block** y **display: inline**, ya que es un concepto esencial para el diseño web.

Elementos en línea

Para explicarlo de manera sencilla, y simplificando, ciertas etiquetas HTML no afectan en absoluto al flujo de los demás elementos, limitándose a marcar ciertos fragmentos con una determinada semántica y dejando que el texto y otros elementos contiguos sigan fluyendo en la misma línea, colocándose a ambos lados de dicho elemento.

Ejemplos de estas etiquetas son: <a>, ,<del>... y quizá la más útil de todas: la etiqueta , que se usa para envolver elementos en-línea para darles estilo sin cambiar su comportamiento. A estos elementos HTML se les llama elementos de línea o, incluso más a menudo, por su denominación en inglés: elementos "inline".

Elementos de bloque

Por el contrario, ciertas etiquetas se renderizan en el navegador en líneas independientes, no mezcladas con el resto del texto. Ejemplos de estas etiquetas son los encabezados (<h1> hasta <h6>), las citas en bloque (<blockquote>), por supuesto los párrafos (<p>), y

quizá la más conocida de todas que es la etiqueta usada normalmente para envolver a otros elementos. A estos elementos se les denomina elementos de bloque.

- **Modelo Cliente-Servidor**
    - El cliente (ej: navegador, app) hace peticiones de data o servicios y el servidor responde.
    - Todo lo que desarrollamos en web gira alrededor de este intercambio.

Cliente: Dispositivo que envia solicitudes, es el que maneja el **frontend** (HTML, CSS, JavaScript, React, Angular, Vue.js).

Server: Máquina potente que es hospedador de sitios web, bases de datos o servicios. Maneja **backend** (Node.js, Python, Django/Flask, Django rest framework, PHP, Java (Spring), Ruby on Rails).

- ☐ Databases: MySQL, PostgreSQL, MongoDB.
- ☐ Protocolos de comunicación: HTTP, HTTPS, WebSockets, REST APIs.

Red: Internet o red local que conecta los clientes y servers.

Ciclo de petición/respuesta: Proceso de comunicación entre cliente y servidor.

1. El usuario introduce una URL en el navegador.
2. El navegador envía una solicitud HTTP al servidor.
3. El servidor procesa la solicitud y recupera los datos.
4. El navegador muestra la página web al usuario.

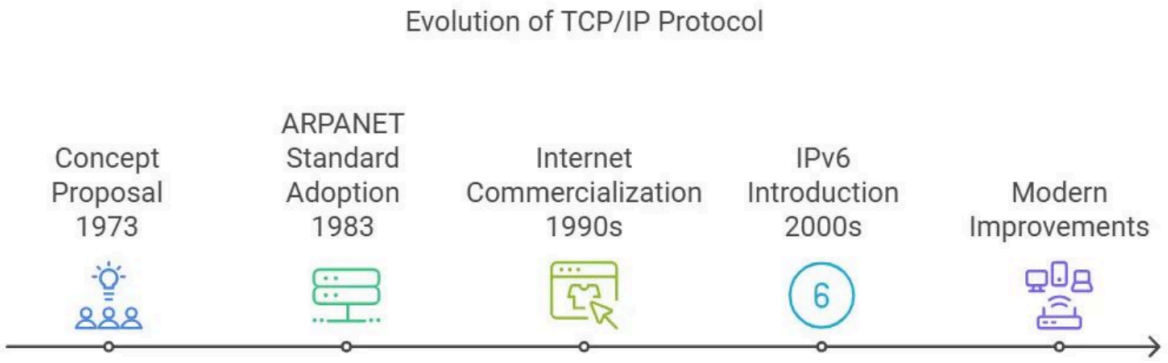
PRIMORDIAL EN WEB:

1. **Reactividad (lógica de negocio).**
2. **Persistencia de los datos.**
3. **Responsividad (adaptarse a varios dispositivos).**

TCP/IP CAPAS:

1. **Primordial**: por cable o aire para conectar la red. Capa física de hardware como routers.
2. **Internet**: protocolos y esquemas IP. **Protocolos:**
    - ☐ IP (Internet Protocol): Direcciones únicas.
    - ☐ ICMP (Internet Control Message Protocol): Para errores.
    - ☐ ARP (Address Resolution Protocol): Traducir IP a dirección MAC (Media Access Protocol).
3. **Transporte**: control de flujo, manejo de errores, protocolos de integridad de los datos y seguridad de la info.
    - ☐ **TCP**: Transmission Control Protocol: Asegura que todo se transmita en orden y completo.
    - ☐ **UDP**: User Datagram Protocol: Rápido pero no asegura que todo se envie.
4. **Aplicación**: la app.
    - ☐ **HTTP/HTTPS**
    - ☐ **SMTP, IMAP, POP3 (para email).**

- ☐ **FTP/SFTP (para archivos).**
- ☐ **DNS (traducir dominios en direcciones IP).**



- **Siglas importantes**
- **HTML HyperText Markup Language**
  - **HTTP** – *HyperText Transfer Protocol*: protocolo que define cómo se comunican navegadores y servidores web.
  - **HTTPS** – *HyperText Transfer Protocol Secure*: igual que HTTP pero cifrado con TLS/SSL.
  - **TCP** – *Transmission Control Protocol*: protocolo de transporte confiable, asegura que los datos lleguen completos y en orden.
  - **UDP** – *User Datagram Protocol*: protocolo de transporte rápido, no garantiza entrega ni orden (se usa en streaming, videojuegos, VoIP).
  - **IP** – *Internet Protocol*: asigna direcciones únicas a dispositivos y permite el enrutamiento de paquetes.
  - **ICMP** – *Internet Control Message Protocol*: usado para diagnósticos y control de errores (ejemplo: comando *ping*).
  - **ARP** – *Address Resolution Protocol*: traduce direcciones IP a direcciones físicas (MAC).
  - **DNS** – *Domain Name System*: convierte nombres de dominio en direcciones IP.
  - **FTP/SFTP** – *File Transfer Protocol / Secure File Transfer Protocol*: protocolos para transferencia de archivos.
  - **SMTP / IMAP / POP3** – protocolos usados en comunicación de correo electrónico.
- **Métodos HTTP (importante memorizarlos)**
  - GET: obtener datos (lectura).
  - POST: enviar datos (crear recursos).
  - PUT: actualizar completamente un recurso.
  - PATCH: actualizar parcialmente un recurso.
  - DELETE: eliminar un recurso.

    extras:

  - HEAD: obtener headers del recurso.
  - CONNECT: abrir dos conexiones de camino.
  - OPTIONS: descubrir las opciones de comunicación.

    - TRACE: propósito de diagnóstico de desarrollo.
- **Códigos de Respuesta HTTP (imprescindibles)**
    - `200` → OK (todo bien).
    - `201` → Created (se creó un recurso).
    - `204` → No Content (acción exitosa sin contenido).
    - `400` → Bad Request (error en la petición).
    - `401` → Unauthorized (falta autenticación).
    - `403` → Forbidden (no tienes permiso).
    - `404` → Not Found (no existe el recurso).
    - `500` → Internal Server Error (error en el servidor).

Other Common HTTP Request Headers	
User-Agent	Contains browser specific information.
Accept	Specifies what type of content the client can accept.
Accept-Encoding	Specifies the type of encoding the client can accept.
Accept-Language	Specifies the language of the client.
Content-Length	This is the length of the request body in octets. This actual value doesn't matter much but some HTTP methods require this header such as PUT but it can simply be set to 0 if its required.
Referer	Contains the URL where the request originated.
Cookie	Submits cookies to the server for session management.
Connection	Used to tell the server whether to close the connection or leave the connection open for subsequent requests.
Authorization	A header that contains platform authentication related information.

HTTP Response Status Codes	
1xx	Informational
2xx	OK, the request was successful
3xx	The client is to be redirected.
4xx	The request cannot be processed for some reason such as its unauthorized or the request has an error.
5xx	The server encountered an error.

Hay una serie de etiquetas que son las más usadas para crear cualquier documento HTML:

- • <body> para el contenido
- • <head> para información sobre el documento
- • división dentro del contenido
- • <a> para enlaces
- • para poner el texto en negrita (enfatizar), si solo es negrita
- •
 para saltos de línea
- • <H1>…<H6> para títulos dentro del contenido
- • <img> para añadir imágenes al documento
- • para listas ordenadas, <ul> para listas desordenadas, <li> para elementos dentro de la lista
- • <p> para parágrafos
- • para estilos de una parte del texto

Principales unidades en CSS (Cascade Style Sheet, hojas de estilo de cascada) usado en estilos de HTML, se usa Tailwind.

Mantenida por el WorldWide Web Consortium (W3C)

HTML es el esqueleto, CSS es el cuerpo.

PILARES CSS:

Herencia: Hijos heredan estilos de padres.
Especificidad: Se aplica el estilo de mayor especificidad.
Cascada: Todo estilo nuevo sobreescribe al antiguo.

Nivel	Año	Descripción
CSS1	1996	Propiedades de fuente, colores, alineación, etc.
CSS2	1998	Propiedades de posicionamiento, tipos de medios, etc.
CSS2.1	2005	Corrige errores de CSS2 y modifica ciertas propiedades
CSS3	2011	Inicio de módulos separados con funcionalidades nuevas

CSS Externo
En la cabecera del HTML, el bloque , incluimos una relación al archivo CSS en cuestión:

```
<link rel="stylesheet" type="text/css" href="index.css" />
```

CSS Interno o etiqueta Style
Otra de las formas que existen para incluir estilos en un documento HTML es la de añadirlos directamente en la cabecera HTML del documento:

```
<!DOCTYPE html>
<html>
<head>
    <title>Título de la página</title>
    <style type="text/css">
        div {
            background:#FFFFFF;
        }
    </style>
</head>
...
```

CSS Embebido o Inline
Por último, la tercera forma de aplicar estilos en un documento HTML es hacerlo directamente en las propias etiquetas, a través del atributo style:

```
<p>¡Hola <span style="color:#FF0000">amigo lector</span>!</p>
```

Al igual que en el método anterior, si hay algún cambio tengo que hacerlo en todas las etiquetas o tags de todos los ficheros en donde lo haya puesto

```
<!Doctype html>
<html>
<body>
<p>¡Hola <span style="color:#FF0000">amigo lector</span>!</p>
<!-- la parte de CSS es la de Style -->
<!--y la etiqueta span es para resaltar una parte del texto-->
</body>
</html>
```

¡Hola amigo lector!

Ventajas de CSS
Ø Mayor control de la presentación del sitio web

Ø Si necesitamos hacer modificaciones de presentación lo hacemos en un sólo lugar y no tenemos que editar todos los documentos HTML por separado.
Ø Mayor legibilidad más fácil de interpretar y entender.
Ø Se reduce la duplicación de estilos en diferentes lugares, por lo que la información a transmitir es considerablemente menor (las páginas se descargan más rápido).
Ø Es más fácil crear versiones diferentes de presentación para otros tipos de dispositivos: tabletas, smartphones o dispositivos móviles, etc...

Desventajas de CSS
Ø A veces, dependiendo del navegador, la página que ha sido maquetada con CSS puede verse distinta
Ø El uso de las tablas nos permitía crear diseños complejos de forma mucho más sencilla que utilizando CSS

- `px` (píxeles): Unidad absoluta, representa un número fijo de píxeles en la pantalla.
- `rem` (Root EM): Unidad relativa basada en el tamaño de la fuente raíz del documento. Por defecto, `1rem = 16px`.
- `em`: Similar a `rem`, pero relativa al tamaño de fuente del elemento padre.
- `%` (porcentajes): Se calcula con base en el tamaño del contenedor padre.
- `vw` (viewport width): 1% del ancho total de la ventana del navegador.
- `vh` (viewport height): 1% de la altura total de la ventana del navegador.

📝 Ejemplo con `rem` y `px`

```
body {
    font-size: 16px; /* 1rem equivale a 16px */
}

h1 {
    font-size: 2rem; /* 32px */
}

p {
    font-size: 1rem; /* 16px */
}
```

📌 **En Tailwind, `p-4` equivale a `1rem`, que es `16px`.**

🎨 Clases de Tailwind CSS que utilizarás

🔹 Espaciado: Margen (`m-`) [DOC] y Padding (`p-`) [DOC]

- `m-4` → Aplica un margen de `1rem` en los cuatro lados del elemento.
- `mt-6` → Margen arriba de `1.5rem`.
- `ml-2` → Margen izquierdo de `0.5rem`.

- `p-6` → Aplica un relleno interno de `1.5rem` en todos los lados.
- `px-4` → Aplica un relleno solo en los lados izquierdo y derecho.

```
<div class="p-6 m-4 bg-gray-200">
    Contenido con padding y margen
</div>
```

🔹 Bordes (`border-`) y Radio de Bordes (`rounded-`) [DOC]

- `border` → Agrega un borde fino alrededor del elemento.
- `border-2` → Aumenta el grosor del borde.
- `border-gray-300` → Cambia el color del borde.
- `rounded-lg` → Aplica bordes redondeados de tamaño grande.
- `rounded-full` → Hace el borde completamente circular.

```
<div class="border border-gray-400 rounded-lg p-4">
    Caja con borde y esquinas redondeadas
</div>
```

🔹 Flexbox (`flex`, `justify-`, `items-`) [DOC]

- `flex` → Activa el modelo de caja flexible.
- `justify-start` → Alinea los elementos a la izquierda.
- `justify-center` → Centra los elementos horizontalmente.
- `justify-end` → Alinea los elementos a la derecha.
- `justify-between` → Distribuye el espacio uniformemente entre los elementos.
- `items-center` → Centra los elementos verticalmente.

En Tailwind CSS, los valores de **margen (`m-`) y padding (`p-`)** utilizan unidades relativas como `rem`.

JAVASCRIPT

Agrega interactividad a las webs, es el cerebro, asi como HTML esqueleto y CSS cuerpo, nace en 1995.

Formas de incluir JavaScript:
● Interno: Dentro de un <script> en HTML
● Externo: En un archivo .js vinculado
● En línea: Directamente en atributos HTML (no recomendado)

JavaScript es un lenguaje de tipado dinámico
● No es necesario definir el tipo de dato al declarar una variable.
● El tipo de una variable puede cambiar en tiempo de ejecución.

Declaración de variables:
● var (evitar usarlo)
● let (para variables que cambian)
● const (para valores fijos)

Tipos de datos primitivos:
● String → "Hola, mundo!"
● Number → 25, 19.99
● Boolean → true, false
● Undefined → Variable sin valor asignado
● Null → Representa ausencia de valor
● BigInt → Números muy grandes
1234567890123n

Tipos de datos estructurados:
● Object → { nombre: "Ana", edad: 30 }
● Array → ["rojo", "verde", "azul"]
● Function → function saludar() { return "¡Hola!"; }

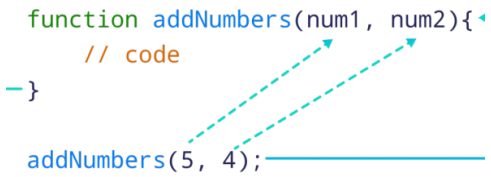
Condicionales: Tomar decisiones en el código
● Se utilizan para ejecutar diferentes bloques de código según una condición. ● Operadores de comparación: ==, ===, !=, !==, >, <, >=, <=
Tipos de bucles: for, while, do...while, for...of, for...in

```
let colores = ["rojo", "verde", "azul"];
for (let color of colores) {
  console.log(color);
}

```

```
let persona = { nombre: "Ana", edad: 30, ciudad: "Madrid" };
for (let clave in persona) {
  console.log(clave + ": " + persona[clave]);
}

```



Funciones flecha (Arrow functions):

```
// an arrow function to add two numbers
const addNumbers = (a, b) => a + b;

// call the function with two numbers
const result = addNumbers(5, 3);
console.log(result);

// Output: 8
```

```
const sayHello = () => "Hello, World!";

// call the arrow function and print its return value
console.log(sayHello());

// Output: Hello, World!
```

```
const square = x => x * x;

// use the arrow function to square a number
console.log(square(5));

// Output: 25
```

DOM

Document Object Model

Estructura, disponible en navegador, forma de árbol de cómo se maneja la página web.

FRAMEWORKS (NEXT) de JavaScript: Next, Vue, Angular.

REACT
React es una librería de JavaScript que permite construir (UI) interfaces de usuario de manera declarativa. Se basa en componentes, que son piezas de interfaz reutilizables.
Usa un concepto llamado **estado** para manejar cómo cambia la interfaz cuando cambian los datos. Ejemplo: un botón que incrementa un número. Cada vez que cambias el valor del número, React actualiza la pantalla automáticamente (esto es reactividad)

NEXT
Next.js es un framework construido sobre React que agrega:
● Rutas automáticas (cada carpeta o archivo en app/ es una página).
● Renderizado en el servidor para mejorar rendimiento y SEO (Search Engine Optimization).
● Herramientas listas para producción (optimización de imágenes, CSS, etc.).
● Permite dividir páginas en server components y client components, optimizando el rendimiento.

NPM (Node Package Manager)
Es necesario ya que el navegador solo entiende HTML, CSS y Javascript.
Empaqueta todo en un package.json para que el navegador lo comprenda, otros gestores de paquetes son yarn, pnpm.
JSON JavaScript Object Notation.
Node no depende del navegador, ya que se ejecuta en el equipo.

Aprender a manejar APIS (Application Programming Interface).

Proceso de instalado:
node -v (version)
npm -v (version)
En la carpeta donde se desea realizar:
npx create-next-app@latest (ingresar datos, los primeros 3 son no).
¿Quieres usar TypeScript? (Yes/No)
TypeScript es como JavaScript, pero con tipos. Esto ayuda a detectar errores antes de ejecutar el código. Recomendado (por ahora): No.
¿Quieres un linter?
Linter ayuda a detectar errores de estilo y formato. Para este curso, selecciona: None.
¿Quieres usar Tailwind CSS?
Tailwind te permite dar estilos con clases rápidas y fáciles. Respuesta: Yes.
¿Quieres usar src/ directory?
Para mantenerlo ordenado: Yes.
¿Quieres usar App Router?
Sí, selecciona Yes.
→ Esto significa que tu aplicación se estructurará en la carpeta app/.
¿Quieres usar Turbopack?
Yes. Turbopack es un empaquetador moderno y rápido que reemplaza a Webpack en proyectos Next.js.

¿Quieres cambiar el import alias?
Déjalo por defecto: @/.
Paquetes instalados automáticamente
Cuando se crea el proyecto, Next.js instala varios paquetes importantes:
● react → la librería base para construir interfaces.
● react-dom → permite que React funcione en el navegador.
● next → el framework que añade rutas, optimización, SSR (server-side rendering), etc.
● tailwindcss → herramienta para estilos rápidos y consistentes.

Para ejecutarlo se necesita que este dentro de la carpeta del proyecto y darle
npm run dev
Copiar la dirección web y moverse con / dependiendo del proyecto creado.

CONCEPTOS:

- SPA (Single Page Application): Aplicación web que carga una sola página HTML y actualiza el contenido dinámicamente sin recargar toda la página. Ejemplo: React, Vue, Angular.
- SSR (Server Side Rendering): El servidor genera el HTML completo antes de enviarlo al navegador. Mejora el SEO (Search Engine Optimization) y la velocidad inicial, pero depende más del servidor.
- Arquitectura de software: Es la forma en que se organizan los componentes de un sistema y cómo se comunican entre sí. Define la estructura general y las reglas del juego del proyecto.
- Arquitectura basada en componentes: Divide la interfaz en pequeñas piezas reutilizables (componentes). Cada componente tiene su propia lógica y estilo.
- Por qué es importante: Facilita la reutilización, el mantenimiento y el trabajo en equipo.
- Diferencia con la forma tradicional: En lugar de tener archivos HTML y JS separados por páginas completas, trabajamos con piezas pequeñas que se combinan para formar la interfaz.
- Componente: Unidad mínima de la interfaz en React. Puede ser un botón, una tarjeta, un formulario, etc.
- Props: Son los “argumentos” que recibe un componente. Permiten pasar datos de un componente padre a uno hijo.
- Estado (State): Es la información interna de un componente que puede cambiar con el tiempo.
- Hook useState: Es una función especial de React que permite crear y actualizar el estado dentro de un componente.
- Watch (observar un estado): Significa reaccionar cuando el valor del estado cambia, por ejemplo, actualizando la vista.

https://react.dev/learn
Componente
Las aplicaciones React están compuestas por componentes. Un componente es una parte de la interfaz de usuario (UI) que tiene su propia lógica y apariencia. Un componente puede ser tan pequeño como un botón o tan grande como una página completa.

○ Es como una “pieza” de la interfaz de usuario.
○ Cada componente puede representar un botón, un tablero o una página completa.
○ En React (y por tanto en Next.js) un componente es una función de JavaScript que devuelve código JSX (mezcla de HTML + JS).

Estado (state)
○ Es la “memoria” de un componente.
○ Sirve para guardar valores que cambian con el tiempo (por ejemplo: el turno actual, el tablero del juego).
Cuando el estado cambia, React redibuja la interfaz automáticamente.

Hook
○ Es una función especial de React que te permite usar características avanzadas (como estado o efectos) dentro de un componente.
○ Siempre empiezan con use (ej: useState, useEffect).

useState
○ Es un hook que te permite crear un estado dentro de un componente.
○ Devuelve dos cosas:
■ El valor actual del estado.
■ Una función para cambiar ese valor.

Ejemplo:
const [count, setCount] = useState(0);
■ count → valor actual (inicia en 0).
■ setCount → función que actualiza count.

Funciones propias de React que vas a usar
○ onClick: evento que se ejecuta cuando el usuario hace clic en un botón o casilla.
○ return: dentro de un componente, indica qué debe “dibujar” React en pantalla.
○ props: son los valores que un componente hijo recibe desde su padre (ejemplo: cada casilla recibe su valor y qué hacer cuando la presionan).

Renderizado condicional:
```
return (
  <div>
    <h1>Bienvenido</h1>
    {isAdmin ? (
      <button>Panel de control</button>
    ) : (
      <p>No tienes acceso al panel</p>
    )}
  </div>
);
```
o

```
return (
  <div>
    <h1>Bienvenido</h1>
    {isAdmin && <button>Panel de control</button>}
  </div>
);
```
&& no tiene else.

Renderizado de lista:
```
function ListaNombres() {
  return (
    <ul>
      {nombres.map((nombre, index) => (
        <li key={index}>{nombre}</li>
      ))}
    </ul>
  );
}
```
key es una propiedad obligatoria para ayudar a React a identificar cada elemento de la lista y actualizarla eficientemente

… operador spread para copiar algo que evita modificar los atributos originales directamente.
_ usado para cubrir un hueco en una función que no existe (funciones flecha).
filter(n => n>2) ciclo que compara el atributo ingresado con cada uno de los atributos de un arreglo, si no cumple la condición, lo borra.
!== operador para definir si algo está o no está.

sort((a,b) => {}) función para ordenar dos elementos a y b.
a.text.localeCompare(b.text) Compara dos strings (a.text y b.text)
Devuelve:
un número negativo → a va antes que b
cero → son iguales
un número positivo → a va después que b

Input:
```
<input
        type="number"
        min="1"
        max="1182"
        className="border text-xl bg-white mt-2 mx-10 rounded-lg"
        value={index} //convierte al <input> en un componente controlado
        onChange={(e) => {
          const val = parseInt(e.target.value, 10);
          if(isNaN(val)) { //Si está vacío (no es número) lo posiciona a 1
            setIndex(1);
          }else{
            //Si no está vacío entonces lo limita entre 1 y 1182
```

```
                //que es la cantidad maxima de personajes.
                const limite = Math.min(1182, Math.max(1, val));
                //Math.max garantiza que el valor nunca sea menor que 1, si val es -10, el
mayor entre -10 y 1 es 1.
                //Math.min garantiza que el valor no exceda de 1182, si val es 2000, math.max
lo deja en 2000 ya que es mayor a 1
                //Más en el Math.min se seleccionará 1182 ya que es el minimo entre 2000 y
1182.
                setIndex(limite);
            }
          }}
        />
```

Mostrar componente (character) en el page.js de API:

```
<hr />
        <Character url={`${API_BASE_URL}/characters/${index}`}></Character>
        </div>
```

Estructura de un componente API:

```
import { useEffect, useState } from "react";
export default function Character({ url }) {
    const [loading, setLoading] = useState(true);
    const [data, setData] = useState({});

    useEffect(() => {
        console.log(url, data);
        fetch(url)
        .then((response) => response.json())
        .then((responseJson) => {
            setData(responseJson);
        });
        setLoading(false);
    }, [url]);

    return (<div>{loading ?
        <div className="text-center text-2xl">⏳</div>
        : <div className="bg-indigo-300 flex flex-auto flex-col rounded-full mx-10 my-2 p-5">
            <div className="text-2xl text-center font-bold text-white">
                {data.name}
            </div>
            <div className="flex justify-center">
                <img className="h-40 w-40 bg-indigo-100 rounded-full"
src={"https://cdn.thesimpsonsapi.com/500" + data.portrait_path} alt="" />
(alt es texto alternativo y src construye la URL dinámica)
            </div>
        </div>}
        </div>
    )
}
```

useEffect(() => { ... }, [url]);
useEffect ejecuta el código cuando el componente se renderiza por primera vez
y cada vez que url cambie.
() => { ... } → lo que debe ejecutarse.
[url] → dependencia: si url cambia, el efecto se vuelve a ejecutar.
fetch(url) → hace una solicitud HTTP a esa URL.
.then((response) => response.json()) → convierte la respuesta en JSON.
.then((responseJson) => { setData(responseJson); }) → guarda la respuesta en data usando setData

Responsividad: Es la capacidad de una aplicación (como las hechas con React, Vue, o Svelte) para actualizar la interfaz automáticamente cuando cambia el estado interno.

API:

- **REST:** JSON -> Formato de serialización.
- **SOAP:** XML -> Formato de Serialización.

Serialización:
Un serializador (o serializer, en inglés) es un componente o proceso que convierte datos de un formato a otro, normalmente para que puedan almacenarse o transmitirse más fácilmente.

Preguntas previo:

- ¿Qué son los headers de la respuesta?

Los headers de la respuesta (o encabezados de respuesta) son un conjunto de metadatos que el servidor envía junto con la respuesta HTTP al cliente (por ejemplo, un navegador o una aplicación). Estos encabezados no son el contenido que el usuario ve (como una página HTML o un JSON), sino información adicional sobre la respuesta, como su tipo, duración, permisos, o estado de la conexión.

Django

Framework de desarrollo (backend) de alto nivel para python.
Patrón MTV (Modelo, Template, View)
view: tiene toda la lógica de negocio.
template: solo vistas, aplicación aparte en react, vista HTML renderizada.
model: bd.

Django viene con muchas funcionalidades listas para usar:

- ☐ ➤ ORM (Object-Relational Mapping)
- ☐ ➤ Panel de administración automático
- ☐ ➤ Sistema de templates
- ☐ ➤ Formularios y validación
- ☐ ➤ Autenticación y permisos
- ☐ ➤ Protección contra ataques web

Maneja el esquema **DRY** – Don't Repeat Yourself
Favorece la reutilización y evita escribir código redundante.

- ☐ ➤ Configuración centralizada
- ☐ ➤ Uso de herencia en templates y
- ☐ formularios
- ☐ ➤ Reutilización de vistas genéricas y
- ☐ mixins

- **Modelos:** definición de entidades, tipos de campos y relaciones (`ForeignKey`, `ManyToMany`, `OneToOne`).
- **ORM (Object Relational Mapper):** cómo Django traduce las operaciones de Python a SQL.
- **QuerySets y Managers:** consultas básicas y personalizadas (`filter`, `exclude`, `values`, `objects.all()`).
- **Vistas:** flujo `request → view → response`, vistas basadas en clases.
- **Templates:** uso del motor de plantillas. Contextos.

🔹 ¿Qué es un Proyecto Django?

- ➤ Es el contenedor principal de toda la configuración global del sitio web.
- ➤ Contiene la configuración general del sitio (settings.py)
- ➤ Define rutas principales (urls.py)
- ➤ Se crea con: django-admin startproject mysite

➤ **Ejemplo**: un sistema completo de blog, e-commerce, o red social.

🔹 ¿Qué es una Aplicación (App)?

- ➤ Es un módulo funcional reutilizable dentro del proyecto.
- ➤ Se encarga de una sola responsabilidad (blog, usuarios, pagos…)
- ➤ Tiene sus propios modelos, vistas, templates, etc.

- ➤ Se crea con: python manage.py startapp blog

🧠 Metáfora

- 🏢 Proyecto = Edificio
- 🚪 Aplicaciones = Departamentos dentro del edificio

🧠 ¿Qué es un Modelo?

Es una clase que representa una tabla en la base de datos.

Cada instancia de la clase = un registro Cada atributo de la clase = una columna

```python
from django.db import models

class Post(models.Model):
    titulo = models.CharField(max_length=100)
    contenido = models.TextField()
    publicado = models.DateTimeField(auto_now_add=True)
```

🧠 ¿Por qué usamos makemigrations y migrate?

Cuando definimos o modificamos modelos en Django, esos cambios deben reflejarse en la base de datos. makemigrations detecta los cambios en los modelos y crea archivos que representan esas modificaciones (migraciones). migrate ejecuta esas migraciones en la base de datos, creando o alterando tablas según lo definido en los modelos.

🧠 Piensa en makemigrations como “guardar los planos” y migrate como “construir según esos planos”.

CAMPOS

Tipo	Clase de campo	Uso común
Texto corto	`CharField(max_length=...)`	Títulos, nombres, etiquetas
Texto largo	`TextField()`	Contenidos grandes como posts o descripciones
Entero	`IntegerField()`	Edad, cantidad, puntuaciones
Booleano	`BooleanField()`	Activo/inactivo, sí/no
Fecha y hora	`DateTimeField(...)`	Fechas de creación, actualización, etc.
Decimal	`DecimalField(...)`	Precios, montos con decimales
Archivo/imagen	`FileField()` / `ImageField()`	Uploads de archivos o fotos
Clave foránea	`ForeignKey(...)`	Relaciones entre modelos

¿Cómo se maneja el ORM de Django?

Definiendo modelos como clases de Python, los cuales se traducen automáticamente a tablas de base de datos y se interactúa con ellos mediante "QuerySets" (conjuntos de consultas) en lugar de escribir SQL directamente. Este sistema permite realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) usando métodos de Python, como .all(), .filter(), .get(), .save() y .delete(), que son convertidos a SQL por el ORM.

EL ORM DE DJANGO

- ➤ ORM = Object-Relational Mapper
- ➤ Permite interactuar con la base de datos usando objetos de Python, en lugar de escribir SQL directamente.
- ➤ Traduce sentencias como Book.objects.all() en consultas SQL.

¿Qué beneficios tiene?

- ➤ Evita SQL repetitivo y propenso a errores.
- ➤ Aumenta la legibilidad y mantenibilidad del código.
- ➤ Funciona con múltiples motores (PostgreSQL, MySQL, SQLite…).

```python
class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.CharField(max_length=100)
```

```python
# Obtener todos los libros
Book.objects.all()

# Filtrar por autor exacto
Book.objects.filter(author="Isabel Allende")

# Filtrar por año mayor a 2000
Book.objects.filter(year__gt=2000)

# Obtener un solo libro (espera exactamente uno)
Book.objects.get(id=1)

# Crear un nuevo registro
Book.objects.create(title="Cien años de soledad",
                    author="Gabriel García Márquez", year=1967)

# Actualizar un registro
book = Book.objects.get(id=1)
book.title = "Nuevo título"
book.save()

# Eliminar un registro
book.delete()
```

El atributo objects

- ➤ Es una instancia del manager por defecto de Django.
- ➤ Permite hacer consultas sobre la tabla del modelo

```
Book.objects.all()          # SELECT * FROM book;
Book.objects.get(id=1)      # SELECT ... WHERE id = 1;
Book.objects.create(...)    # INSERT INTO ...
```

¿Qué es un Manager?

- ➤ Es una clase que gestiona las operaciones de base de datos de un modelo.
- ➤ Puedes crear managers personalizados con métodos específicos.

```
class BookManager(models.Manager):
    def colombianos(self):
        return self.filter(author__icontains="colombia")

class Book(models.Model):
    ...
    objects = BookManager()

# Book.objects.colombianos()
```

INSTALACIÓN: https://docs.djangoproject.com/en/5.2/howto/windows

python -m venv venv (inciar entorno virtual)

cd venv, cd Scripts => activate.bat (activarlo) o tambien con: venv/scripts/activate
pip install django (instala django)

Crear una carpeta aparte dentro del venv para ejecutar lo siguiente:

django-admin startproject mysite
py manage.py startapp polls

Dentro de settings.py INSTALLED APPS agregar:

"polls.apps.PollsConfig", (dependiendo del nombre de la app que haya creado y el Config en apps.py)

Crear migraciones (hacer cada vez que se cambien los modelos y restricciones validaciones):

py manage.py makemigrations polls
py manage.py sqlmigrate polls 0001
py manage.py migrate

Crear superuser:

py manage.py createsuperuser (rellenar los espacios de usuario y contraseña, no olvidar)

Correr servidor:
py manage.py runserver (entramos luego a la dirección que nos envía http://127.0.0.1:8000/admin/ para así entrar a la bd).

Si desea abrir la consola de python: py manage.py shell

https://docs.djangoproject.com/en/5.2/intro/tutorial01

https://docs.djangoproject.com/en/5.2/intro/tutorial02

⚙️ Django REST Framework (DRF)

Es un poderoso toolkit para construir APIs web en Django. Permite serializar datos y definir endpoints RESTful. Facilita la autenticación, permisos, paginación y más. Ideal para contruir APIs limpias en Django.

Serializadores: transformación de modelos a JSON y validación de datos.

- ☐ Convierten objetos complejos (como modelos Django) en datos simples (JSON, XML).
- ☐ Validan datos de entrada y los convierten en objetos Python.
- ☐ Se usan para definir la estructura de los datos que expondrá tu API.

```python
from rest_framework import serializers
from .models import Book

class BookSerializer(serializers.ModelSerializer):
    class Meta:
        model = Book
        fields = ['id', 'title', 'author', 'year']
```

Puntos Clave:

ModelSerializer genera automáticamente los campos a partir del modelo. Puedes usar Serializer si quieres más control manual. La validación ocurre automáticamente con is_valid(). Puedes sobrescribir métodos como create() y update() si necesitas lógica personalizada.

```python
# Serializar un objeto
book = Book.objects.first()
serializer = BookSerializer(book)
print(serializer.data)  # {'id': 1, 'title': '...', ...}
```

- **ViewSets y Routers:** exposición de endpoints y manejo automático de rutas.

☐ **ViewSet** es una clase que agrupa operaciones CRUD de un recurso (modelo).
☐ DRF proporciona métodos listos como: list, create, retrieve, update, destroy.
☐ Se integra con routers para generar las URLs automáticamente.

```python
from rest_framework import viewsets
from .models import Book
from .serializers import BookSerializer

class BookViewSet(viewsets.ModelViewSet):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
```

ModelViewSet incluye todos los métodos CRUD por defecto. Puedes sobrescribir métodos (get_queryset, perform_create, etc.) para personalizar la lógica. Puedes restringir acciones usando mixins si no necesitas el CRUD completo. Se combina con un router para evitar escribir manualmente las rutas.

```python
from rest_framework import viewsets
from .models import Book
from .serializers import BookSerializer

# Un ViewSet agrupa las operaciones CRUD para el modelo Book
class BookViewSet(viewsets.ModelViewSet):

    # Define el conjunto de datos sobre el que actuará este ViewSet
    queryset = Book.objects.all()
    # Aquí se hace la consulta a la base de datos usando el ORM de Django

    # Asocia un serializador que define cómo convertir los objetos Book
    # en JSON y viceversa (y que incluye validaciones)
    serializer_class = BookSerializer

    # Opcional: puedes agregar lógica adicional sobrescribiendo métodos como:
    # def perform_create(self, serializer):
    #     serializer.save(owner=self.request.user)
```

Router:

- Es una clase que automatiza la creación de URLs para los ViewSets.
- Asocia las rutas HTTP (GET, POST, PUT, DELETE) a los métodos del ViewSet.
- Evita que tengas que escribir manualmente las rutas en urls.py.

DefaultRouter incluye además una interfaz de navegación para el API.

La línea router.register('books', BookViewSet) genera rutas como:

- ☐ ➤ GET /api/books/ → list()
- ☐ ➤ POST /api/books/ → create()
- ☐ ➤ GET /api/books/1/ → retrieve()
- ☐ ➤ PUT /api/books/1/ → update()
- ☐ ➤ DELETE /api/books/1/ → destroy()

```python
# urls.py
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import BookViewSet

# Creamos el router y registramos el ViewSet
router = DefaultRouter()
router.register(r'books', BookViewSet)

# Incluimos las rutas generadas automáticamente
urlpatterns = [
    path('api/', include(router.urls)),
]
```

- **Filtrado:** uso de `django-filter` para consultas parametrizadas en endpoints.