**Carpetas duplicadas en vite por renombre de archivos por ser case sensitive:**

Al renombrar archivos con solo la mayuscula vite guarda en memoria el nombre del archivo en minúscula por lo que al intentar importar el nuevo archivo con el nuevo nombre indica que hay dos archivos con el mismo nombre, esto por que Windows no es case sensitive y crea un conflicto.

Para resolverlo:

\*\*Intenta solo abrir y cerrar VSCode\*\*

\*\*Perfecto, recién me paso esto con otros archivos y lo que hice fue cambiar el nombre momentáneamente y volverlo a cambiar al nombre que quiero, cerre vscode y lo volvi a abrir y ya acepta los nombres

Remove-Item -Path "node\_modules\.vite" -Recurse -Force

npm cache clean –force

npm run dev

Y luego Cierra y abre VSCode

**Carpeta Utils:**

En ella se guardan funciones que se usan mas de una vez, esto para conservar el proyecto D.R.Y

**Nomenclatura BEM en CSS:**

**B**lock: primero escribimos el nombre del bloque : .persona-de-palitos

**E**lement: Brazos, cabeza.. ets .persona-de-palitos\_ \_cabeza

**M**odificador: rojo, verde, grande .. ets persona-de-palitos - - verde

.persona-de-palitos\_ \_cabeza- - grande

Repasa como poner un video en un fondo y como hacer el efecto de blurr

.formContainer\_\_Backgroundvideo{

  position: absolute;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

  object-fit: cover;

  z-index: -1;

   border-radius: 0px 0px 38px 38px;

}

.formContainer\_\_title, .formContainer\_\_title--secondary{

   color: var(--light-background);

   font-size: var(--font-size-primary-mobile);

   background: rgba(88, 88, 88, 0.2);

   backdrop-filter: blur(1.5px);

   padding: 0.2rem .4rem;

   border-radius: 10px;

}

Lo de el formulario, evitar que se genere el comportamiento default:

<https://chatgpt.com/s/t_68f00a672ed8819190bc7273f6a3f4a5>

Detectar cuando un elemento en el DOM es clickeado o no es clickeado:

import { useEffect, useRef, useState } from "react";

export default function Example() {

  const inputRef = useRef(null);

  const [isActive, setIsActive] = useState(false);

  useEffect(() => {

    function handleClickOutside(event) {

      if (inputRef.current && !inputRef.current.contains(event.target)) {

        setIsActive(false);

      }

    }

    document.addEventListener("mousedown", handleClickOutside);

    return () => {

      document.removeEventListener("mousedown", handleClickOutside);

    };

  }, []);

  return (

    <div>

      <input

        ref={inputRef}

        onFocus={() => setIsActive(true)}

        placeholder="Click fuera para cerrar"

      />

      {isActive && <p>Activo</p>}

    </div>

  );

}

Cuando manejas un evento de ratón, event.target es **el elemento concreto del DOM donde ocurrió el clic**

| **Concepto** | **Significado** |
| --- | --- |
| **useRef(null)** | **Crea una referencia a un elemento del DOM** |
| **event.target** | **Elemento del DOM donde se hizo click** |
| **.contains()** | **Devuelve true si un nodo contiene a otro** |
| **!inputRef.current.contains(event.target)** | **El click fue fuera del input** |
| **document.addEventListener("mousedown", handleClickOutside)** | **Escucha clicks globales** |

Que hace un use ref?

Lo que hace es guardar un valor entre renderizados sin recargar la pagina cuando se actualiza, al crear el useRef este crea un objeto que es el que guarda dicho valor entre renderizados, el cual tendrá una propiedad que se llama “current” en la cual se guarda dicho valor.

const refTest = useRef("hola")

    console.log(refTest)

consola: {current: “hola”}

Sus dos usos casi únicos son:

1. Guardar una referencia al DOM

**⚙️ 2. ¿Qué es useEffect?**

useEffect se usa para **ejecutar efectos secundarios** en un componente funcional, como:

* Llamadas a APIs
* Manipular el DOM
* Configurar o limpiar listeners
* Actualizar valores fuera del flujo del renderizado