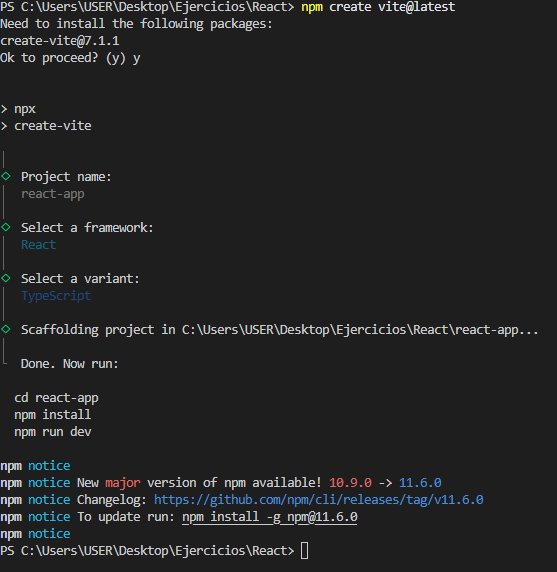
1. Necesitamos primero descarcar **npm create vite@latest**



1. Instalamos en orden lo que pide
2. Para correr se coloca **npm run dev**

**Hooks más usados en React**

**1. useState**

* **Función:** Permite agregar **estado local** a un componente funcional.
* **Sintaxis:**

const [valor, setValor] = useState(valorInicial);

* **Para qué sirve:**
  + Guardar datos que cambian dentro del componente (inputs, contadores, selección de items, etc.).
* **Ejemplo:**

const [count, setCount] = useState(0);

<button onClick={() => setCount(count + 1)}>Incrementar</button>

**2. useEffect**

* **Función:** Ejecuta código en **momentos específicos del ciclo de vida** del componente.
* **Sintaxis:**

useEffect(() => {

// código a ejecutar

}, [dependencias]);

* **Para qué sirve:**
  + Cargar datos al montar el componente (fetch, APIs).
  + Detectar cambios en variables de estado o props.
  + Limpiar recursos al desmontar (timers, listeners).
* **Ejemplo:**

useEffect(() => {

console.log("Se ejecuta al montar y cuando 'count' cambia", count);

}, [count]);

**3. useRef**

* **Función:** Permite crear una **referencia a un elemento DOM** o a un valor que no provoca re-render al cambiar.
* **Sintaxis:**

const ref = useRef<HTMLElement>(null);

* **Para qué sirve:**
  + Acceder a elementos del DOM (focus, scroll, canvas).
  + Guardar valores mutables entre renders sin re-renderizar.
* **Ejemplo:**

const inputRef = useRef<HTMLInputElement>(null);

<input ref={inputRef} />

<button onClick={() => inputRef.current?.focus()}>Focus</button>

**4. useMemo**

* **Función:** Memoriza un **valor calculado** para que no se recalcule en cada render.
* **Sintaxis:**

const memoizado = useMemo(() => calcularAlgo(value), [value]);

* **Para qué sirve:**
  + Optimizar cálculos pesados que dependen de ciertas variables.
* **Ejemplo:**

const numerosGrandes = useMemo(() => calcularSumatoria(lista), [lista]);

**5. useCallback**

* **Función:** Memoriza una **función** para que no se redefina en cada render.
* **Sintaxis:**

const callback = useCallback(() => hacerAlgo(valor), [valor]);

* **Para qué sirve:**
  + Evitar re-render innecesario en componentes hijos que reciben la función como prop.
* **Ejemplo:**

const handleClick = useCallback(() => setCount(count + 1), [count]);

**6. useContext**

* **Función:** Permite acceder al **contexto global** creado con React.createContext.
* **Sintaxis:**

const valor = useContext(MiContexto);

* **Para qué sirve:**
  + Compartir datos entre componentes sin pasar props manualmente (temas, usuario, configuración).
* **Ejemplo:**

const theme = useContext(ThemeContext);

<div style={{ color: theme.color }}>Hola</div>

**7. useReducer**

* **Función:** Maneja estado complejo mediante un **reducer** (como Redux dentro del componente).
* **Sintaxis:**

const [state, dispatch] = useReducer(reducer, estadoInicial);

* **Para qué sirve:**
  + Cuando useState se queda corto por lógica compleja o muchos valores relacionados.
* **Ejemplo:**

function reducer(state, action){

switch(action.type){

case "increment": return { count: state.count + 1 };

default: return state;

}

}

const [state, dispatch] = useReducer(reducer, { count: 0 });

<button onClick={() => dispatch({ type: "increment" })}>+</button>

**Resumen rápido de los Hooks**

| **Hook** | **Para qué sirve** |
| --- | --- |
| useState | Estado local simple |
| useEffect | Efectos secundarios / ciclo de vida |
| useRef | Referencias a DOM o valores mutables |
| useMemo | Memorizar valores calculados |
| useCallback | Memorizar funciones |
| useContext | Consumir contexto global |
| useReducer | Manejar estado complejo con lógica de reducer |

### 🚀 ¿Qué hace cada caso?

#### 1. Sin dependencias []

useEffect(() => {

console.log("Me ejecuto una sola vez al montar el componente");

}, []);

👉 Se ejecuta **solo una vez**, cuando el componente aparece en pantalla.  
Ejemplo: pedir datos a un API al inicio.

#### 2. Con dependencias [algo]

useEffect(() => {

console.log("Me ejecuto cuando cambia 'algo'");

}, [algo]);

👉 Se ejecuta cada vez que la variable algo cambia.  
Ejemplo: actualizar un filtro cuando cambia la lista.

**Nivel 1: Validaciones y formularios simples**

**Ejercicio 1: Formulario básico**

* Crea un formulario con **nombre** y **apellido**.
* Valida que los campos no estén vacíos.
* Muestra un mensaje de error debajo del input.
* Usa **Bootstrap** para estilos (form-control, btn btn-primary).

**Objetivo:** Familiarizarte con inputs, estados y validaciones básicas.

**Ejercicio 2: Validación avanzada**

* Crea un formulario con **nombre, apellido y edad**.
* Valida:
  + Nombre y apellido mínimo 3 letras.
  + Edad debe ser un número entre 1 y 120.
* Deshabilita el botón si hay errores.
* Usa useState y onChange.

**Objetivo:** Aprender validaciones condicionales y deshabilitar botones.

**Ejercicio 3: Validación de lista**

* Crea un array de objetos: { Nombre: string, Apellido: string }.
* Muestra la lista en un componente.
* Valida que ningún campo de la lista esté vacío antes de mostrar.
* Usa Bootstrap list-group.

**Objetivo:** Practicar validación de arrays y mapeo de componentes.

**Nivel 2: Interacciones y callbacks**

**Ejercicio 4: Botón con callback**

* Crea un botón que reciba una función onClick.
* Al hacer click, valida un objeto { Nombre, Apellido } y muestra un alert si todo está bien.
* Cambia el estilo del botón mientras “carga” (isLoading).

**Objetivo:** Aprender a pasar callbacks entre componentes y manejar estados de carga.

**Ejercicio 5: Selección de elementos**

* Crea una lista de elementos con map.
* Al hacer click en un elemento, márcalo como activo y muestra un mensaje con el índice.
* Usa useState para guardar el índice activo.
* Usa Bootstrap list-group-item active.

**Objetivo:** Practicar manejo de listas, eventos y estados dinámicos.

**Nivel 3: Consumo de APIs**

**Ejercicio 6: Llamada simple a API**

* Usa useEffect para hacer un fetch a https://jsonplaceholder.typicode.com/users.
* Muestra los usuarios en una lista.
* Maneja estado de carga y errores.
* Usa Bootstrap para estilos.

**Objetivo:** Aprender a usar useEffect y manejar datos externos.

**Ejercicio 7: API con filtros**

* A partir del ejercicio anterior, agrega un input para filtrar usuarios por nombre.
* Filtra la lista mientras escribes.
* Muestra mensaje si no hay coincidencias.

**Objetivo:** Filtrado dinámico y renderizado condicional.

**Nivel 4: Routing y navegación**

**Ejercicio 8: Rutas básicas**

* Instala react-router-dom.
* Crea 2 rutas: / y /about.
* La página / muestra la lista de usuarios, /about muestra un mensaje estático.
* Agrega un navbar con links.
* Usa Bootstrap navbar.

**Objetivo:** Aprender rutas y navegación en React.

**Ejercicio 9: Ruta dinámica**

* Al hacer click en un usuario de la lista, navega a /user/:id.
* Muestra los detalles del usuario usando el id de la ruta.
* Maneja estado de carga mientras se busca el usuario.

**Objetivo:** Practicar rutas dinámicas y parámetros de URL.

/src

├── /assets

│ └── logo.png

├── /components

│ └── Navbar.jsx

│ └── Button.jsx

├── /pages

│ └── Home.jsx

│ └── Login.jsx

├── /layouts

│ └── MainLayout.jsx

├── /hooks

│ └── useAuth.js

├── /context

│ └── AuthContext.jsx

├── /services

│ └── api.js

├── /utils

│ └── formatDate.js

├── /routes

│ └── AppRouter.jsx

├── App.jsx

├── main.jsx