GUÍA PRÁCTICA 03: DE REACT CON TYPESCRIPT

Objetivo: Consolidar el conocimiento de componentes, formularios, eventos y navegación con Routing. Además, se introduce **props**, **composición de componentes**, **validación de formularios** y **estructura de carpetas profesional**.

1. Estructura del Proyecto

Una vez creado el proyecto con Vite + React + TypeScript:

npm create vite@latest react-avanzado -- --template react-ts

cd react-avanzado

npm install

npm run dev

Organiza tu proyecto así:

css

src/

│

├── components/

│ ├── Header.tsx

│ ├── Footer.tsx

│ └── EstudianteCard.tsx

│

├── pages/

│ ├── Home.tsx

│ ├── Registro.tsx

│ └── Estudiantes.tsx

│

├── App.tsx

├── main.tsx

└── types/

└── estudiante.d.ts

2. Instalación del Router

npm install react-router-dom

Configura el enrutamiento en main.tsx y App.tsx:

```
// main.tsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import { BrowserRouter } from 'react-router-dom'
import App from './App'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')!).render(
 <React.StrictMode>
  <BrowserRouter>
   <App />
  </BrowserRouter>
 </React.StrictMode>
)
// App.tsx
import { Routes, Route, Link } from 'react-router-dom'
import Home from './pages/Home'
import Registro from './pages/Registro'
import Estudiantes from './pages/Estudiantes'

function App() {
 return (
  <div>
   <nav>
    <Link to="/">Inicio</Link> |
    <Link to="/registro">Registro</Link> |
    <Link to="/estudiantes">Estudiantes</Link>
   </nav>
   <hr />
   <Routes>
    <Route path="/" element={<Home />} />
```

```
      <Route path="/registro" element={<Registro />} />
      <Route path="/estudiantes" element={<Estudiantes />} />
    </Routes>
   </div>
  )
}

export default App
```

---

3. Props y Componentes Reutilizables

```
// components/EstudianteCard.tsx
type Estudiante = {
  nombre: string;
  carrera: string;
  correo: string;
};

export default function EstudianteCard({ nombre, carrera, correo }: Estudiante) {
  return (
    <div style={{ border: '1px solid #ccc', padding: '10px', margin: '5px' }}>
      <h4>{nombre}</h4>
      <p>{carrera}</p>
      <small>{correo}</small>
    </div>
  );
}
// pages/Estudiantes.tsx
import EstudianteCard from '../components/EstudianteCard'

export default function Estudiantes() {
  const lista = [
```

```
  { nombre: 'Ana Torres', carrera: 'Informática', correo: 'ana@mail.com' },
  { nombre: 'Luis Ramos', carrera: 'Redes', correo: 'luis@mail.com' }
 ];

 return (
  <div>
   <h2>Listado de Estudiantes</h2>
   {lista.map((e, i) => (
    <EstudianteCard key={i} {...e} />
   ))}
  </div>
 );
}
```

4. Formulario con Validación Básica

```
// pages/Registro.tsx
import { useState, FormEvent } from 'react'
import { useNavigate } from 'react-router-dom'

export default function Registro() {
 const [nombre, setNombre] = useState('')
 const [correo, setCorreo] = useState('')
 const [carrera, setCarrera] = useState('')
 const [error, setError] = useState('')
 const navigate = useNavigate()

 const handleSubmit = (e: FormEvent) => {
  e.preventDefault()
  if (!nombre || !correo || !carrera) {
   setError('Todos los campos son obligatorios')
   return
  }
```

```
  console.log({ nombre, correo, carrera })
  navigate('/estudiantes')
 }

 return (
  <form onSubmit={handleSubmit}>
   <h2>Registro de Estudiante</h2>
   {error && <p style={{ color: 'red' }}>{error}</p>}

   <input
    placeholder="Nombre"
    value={nombre}
    onChange={(e) => setNombre(e.target.value)}
   /><br />

   <input
    placeholder="Correo"
    value={correo}
    onChange={(e) => setCorreo(e.target.value)}
    type="email"
   /><br />

   <input
    placeholder="Carrera"
    value={carrera}
    onChange={(e) => setCarrera(e.target.value)}
   /><br />

   <button type="submit">Registrar</button>
  </form>
 );
```

}

5. Buenas prácticas que ya estás aplicando

Práctica	¿Por qué es importante?
Separar componentes	Permite reutilizar y mantener el código
Usar props	Facilita la personalización de componentes
Formularios controlados	Da control total al programador
Routing	Crea una SPA con múltiples vistas sin recargar
Tipado con TypeScript	Evita errores y documenta el código

ACTIVIDAD FINAL

Desafío: Crea una app de gestión de cursos

Requisitos:

- Un componente CursoCard que reciba nombre del curso, docente y duración como props.
- Una página /cursos que muestre una lista de cursos.
- Un formulario en /nuevo-curso para agregar un curso.
- Al enviar el formulario, redirige a /cursos y muestra el nuevo curso.
- Validar que los campos del curso no estén vacíos.
- Crear un campo select con las carreras disponibles.
- Estilizar usando clases CSS o Tailwind.