Universidad CENFOTEC
SOMOS LO QUE SABEMOS

FRO-03 Desarrollo de Aplicaciones Web con React

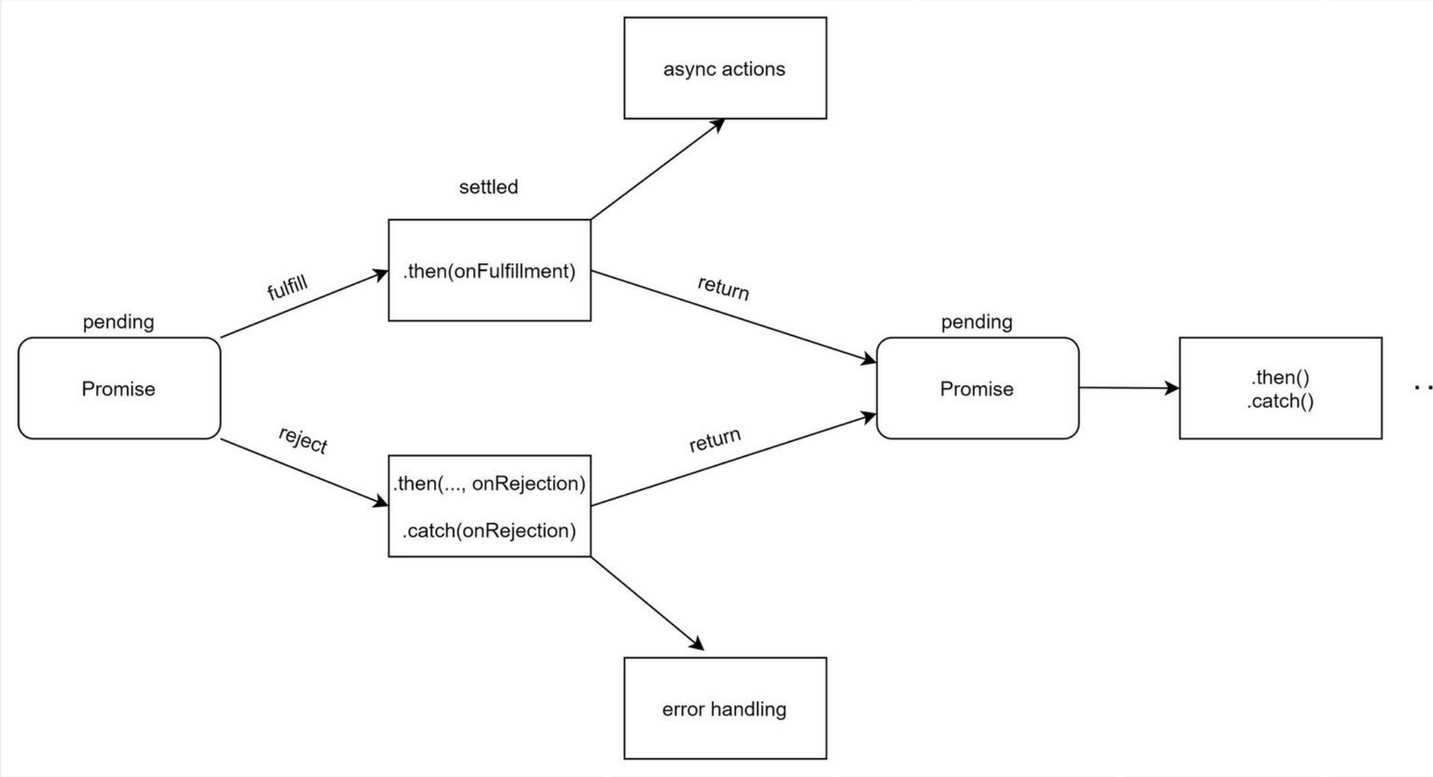
Prof. David Luna
dluna@ucenfotec.ac.cr

Promesas

El objeto Promise en JavaScript representa una operación asíncrona (y su valor resultante) que eventualmente se completará (o fallará).

Una Promise puede estar en uno de estos estados:

pending (pendiente): estado inicial, ni cumplida ni rechazada.
fulfilled (cumplida): significa que la operación se completó con éxito.
rejected (rechazada): significa que la operación falló.



Promesas

```
<script>
const miPromesa = new Promise(function(resolve, reject) {
    setTimeout(() => {
      resolve('Si Funciona');
    }, 3000);
  });
  console.log(miPromesa)


  miPromesa.then((data)=>console.log(data))
  miPromesa.error((error)=>console.log("error"))
</script>
```

Async / Await

Async/Await es una forma de escribir promesas que nos permite escribir código asincrónico, pero de manera sincrónica.

La palabra clave async:

- ➔ Se coloca antes de la declaración de una función.
- ➔ Indica que la función devolverá implícitamente una Promise.
- ➔ Si la función devuelve un valor que no es una Promise, JavaScript lo envolverá automáticamente en una Promise resuelta con ese valor.

```
async function miFuncionAsincrona() {
    return 'Resultado';
}
 miFuncionAsincrona().then(valor => console.log(valor));
```

La palabra clave await:

- ➔ Solo se puede usar dentro de una función declarada como async.
- ➔ Se coloca antes de una expresión que devuelve una Promise.
- ➔ Pausa la ejecución de la función async hasta que la Promise que se está esperando se resuelva (fulfilled) o se rechace (rejected).
- ➔ Si la Promise se resuelve, la expresión await devuelve el valor resuelto de la Promise.
- ➔ Si la Promise se rechaza, la expresión await lanza un error, que puede ser capturado con un bloque try...catch.

Async / Await

```
function promesaConRetraso(ms, exito = true) {
 return new Promise((resolve, reject) => {
     setTimeout(() => {
         if (exito) { resolve(`Resuelto después de ${ms} ms`); }
         else { reject(`Rechazado después de ${ms} ms`);
     } }, ms); });
}

async function usarAwait() {
     console.log('Inicio de usarAwait');
     const resultado1 = await promesaConRetraso(1000);
     console.log('Resultado 1:', resultado1);
      const resultado2 = await promesaConRetraso(500);
     console.log('Resultado 2:', resultado2);
     console.log('Fin de usarAwait');
}
 usarAwait();
console.log('Después de llamar a usarAwait');
```

Manejo de Errores

```
const cargarDatos = async () => {
  try{
      const url =
"https://jsonplaceholder.typicode.com/todos/1";
      if(res.ok){
        const datos = await res.json();
        console.log(datos);
      } else {
        console.log(res.status); // 404
      }
  } catch(err) {
    console.log(err)
  }
};

cargarDatos();
```

Manejo de Errores

```
const cargarDatos = async () => {
      try{
        const url = "https://jsonplaceholder.typicode.com/todos/1";
        const res = await fetch(url);
        const datos = await res.json();
        return datos
      } catch(err) {
        console.log(err)
      }
    };

const datos = cargarDatos().then(datos => console.log(datos));
```

Promise.all

```
const cargarDatos = async () => {
  try{
    const url1 = "https://jsonplaceholder.typicode.com/todos/1";
    const url2 = "https://jsonplaceholder.typicode.com/todos/2";
    const url3 = "https://jsonplaceholder.typicode.com/todos/3";
    const resultados = await Promise.all([
      fetch(url1),
      fetch(url2),
      fetch(url3)
    ]);
    const promesasDeDatos = await resultados.map(result => result.json());
    const datosFinales = Promise.all(promesasDeDatos);
    return datosFinales
  } catch(err) {
    console.log(err)
  }
};
```

```
[{
  completed: false,
  id: 1,
  title: "delectus aut autem",
  userId: 1
}, {
  completed: false,
  id: 2,
  title: "quis ut nam facilis et officia qui",
  userId: 1
}, {
  completed: false,
  id: 3,
  title: "fugiat veniam minus",
  userId: 1
}]
```

Buenas practicas

- ➔ Usa Promise.all para operaciones paralelas: Cuando varias tareas no dependen unas de otras, ejecútalas simultáneamente para mejorar el rendimiento.
- ➔ Utiliza try/catch para capturar errores: Centraliza el manejo de errores para evitar fallos inesperados y mejorar la depuración.
- ➔ Evita bloqueos innecesarios: No uses await en operaciones que no necesitan ser pausadas. Esto puede ralentizar el flujo del programa.
- ➔ Manejo eficiente de errores: En flujos complejos, combina try/catch con funciones reutilizables para reducir la redundancia.

Llamando APIs desde el UseEffect

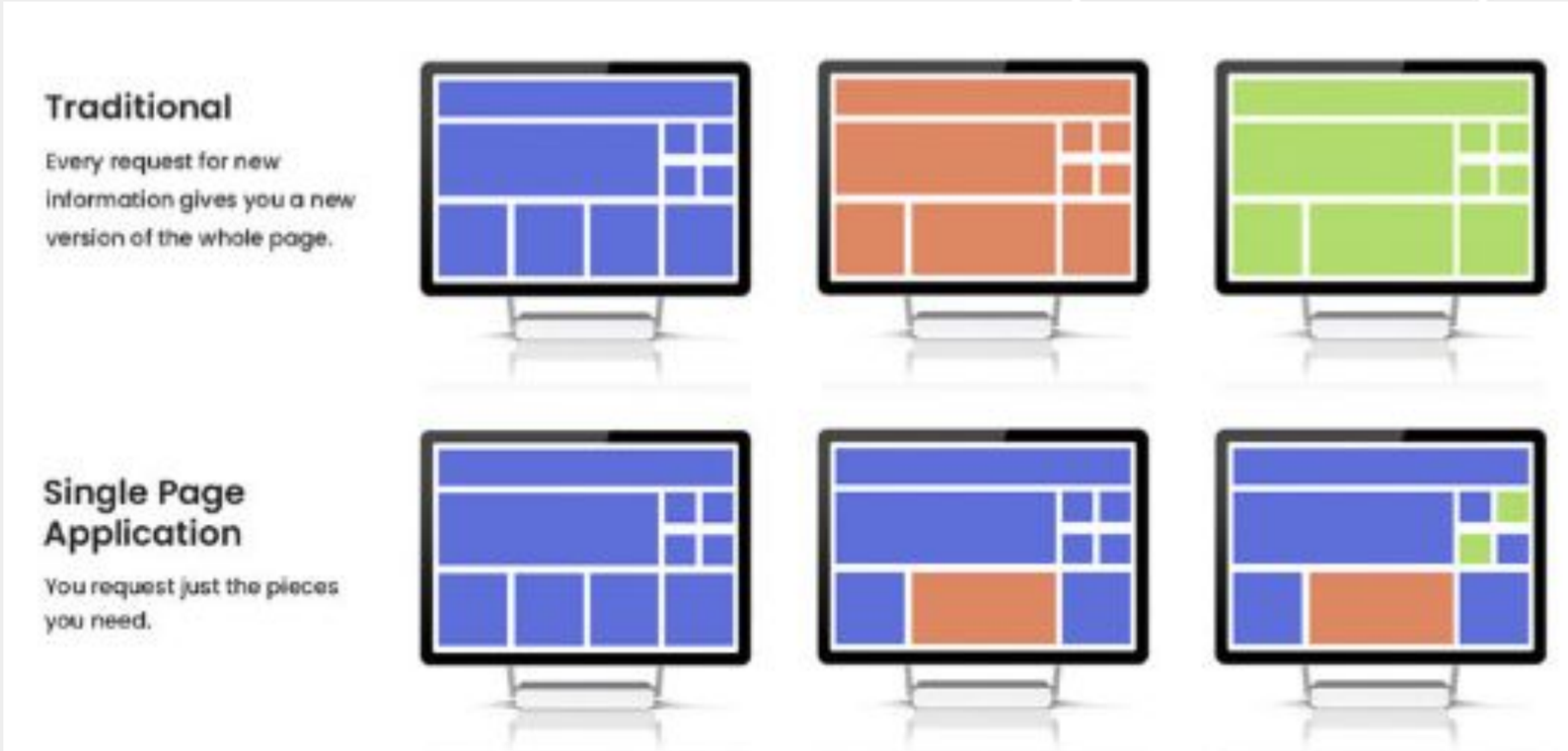
```
const PostList = () => {
  const [posts, setPosts] = useState([]);

  useEffect(() => {
    const fetchPosts = async () => {
      try {
        const response = await fetch(
          "https://jsonplaceholder.typicode.com/posts"
        );
        const data = await response.json();
        setPosts(data);
      } catch (error) {
        console.error("Failed to fetch posts:", error);
      }
    };
    fetchPosts();
  }, []);
```

Ejercicio del Async / Await

SPA

Una aplicación de página única (SPA) es una aplicación web que interactúa con el usuario reescribiendo dinámicamente el contenido de la página en lugar de cargar páginas nuevas desde un servidor.

Esto se traduce en una navegación más rápida y una experiencia de usuario más fluida, ya que la mayoría de los recursos, como HTML, CSS y JavaScript, solo se cargan una vez.



SPA - Características

Una Sola Carga Inicial: El navegador descarga todo el HTML, CSS y JavaScript necesario al principio.

Contenido Dinámico: Después de la carga inicial, todo el contenido nuevo se inyecta y elimina del DOM (Document Object Model) mediante JavaScript.

Experiencia de Usuario Fluida: La navegación es muy rápida porque no hay recargas completas de página. Se siente más como una aplicación de escritorio o móvil.

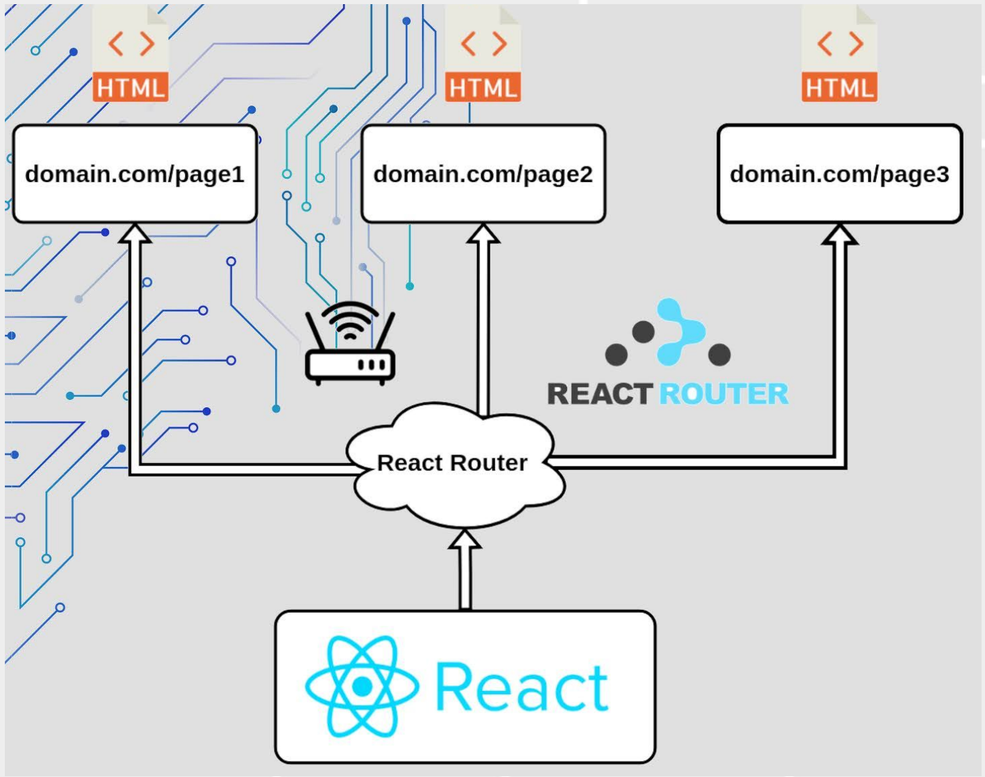
APIs Rest/GraphQL: Las SPAs suelen comunicarse con un backend a través de APIs (RESTful o GraphQL) para obtener y enviar datos, sin necesidad de recargar la página.

Gestión del Historial del Navegador: Aunque solo haya una página HTML, las SPAs utilizan la API History del navegador (ej. pushState) para manipular la URL y permitir que el botón de "atrás" y "adelante" del navegador funcionen como se espera.

React Router Dom

React Router DOM es la librería estándar de facto para manejar el enrutamiento en aplicaciones React. Proporciona los componentes necesarios para mapear URLs a componentes de React sin recargar la página, podemos usarlo tanto en web como en móvil con React Native.

Con esta librería se obtiene un enrutamiento dinámico gracias a los componentes, en otras palabras tenemos unas rutas que renderizan un componente.



https://reactrouter.com/

```
npm install
react-router-dom@6
```

React Router Dom

RouterProvider

El envoltorio principal para tu aplicación. Gestiona el enrutamiento. Requiere un objeto router creado con createBrowserRouter

createBrowserRouter

La forma moderna de definir tus rutas. Permite rutas anidadas, manejo de errores, y carga de datos

Link

El reemplazo de <a>. Cambia la URL y renderiza el componente sin recargar. Usa la prop to.

NavLink

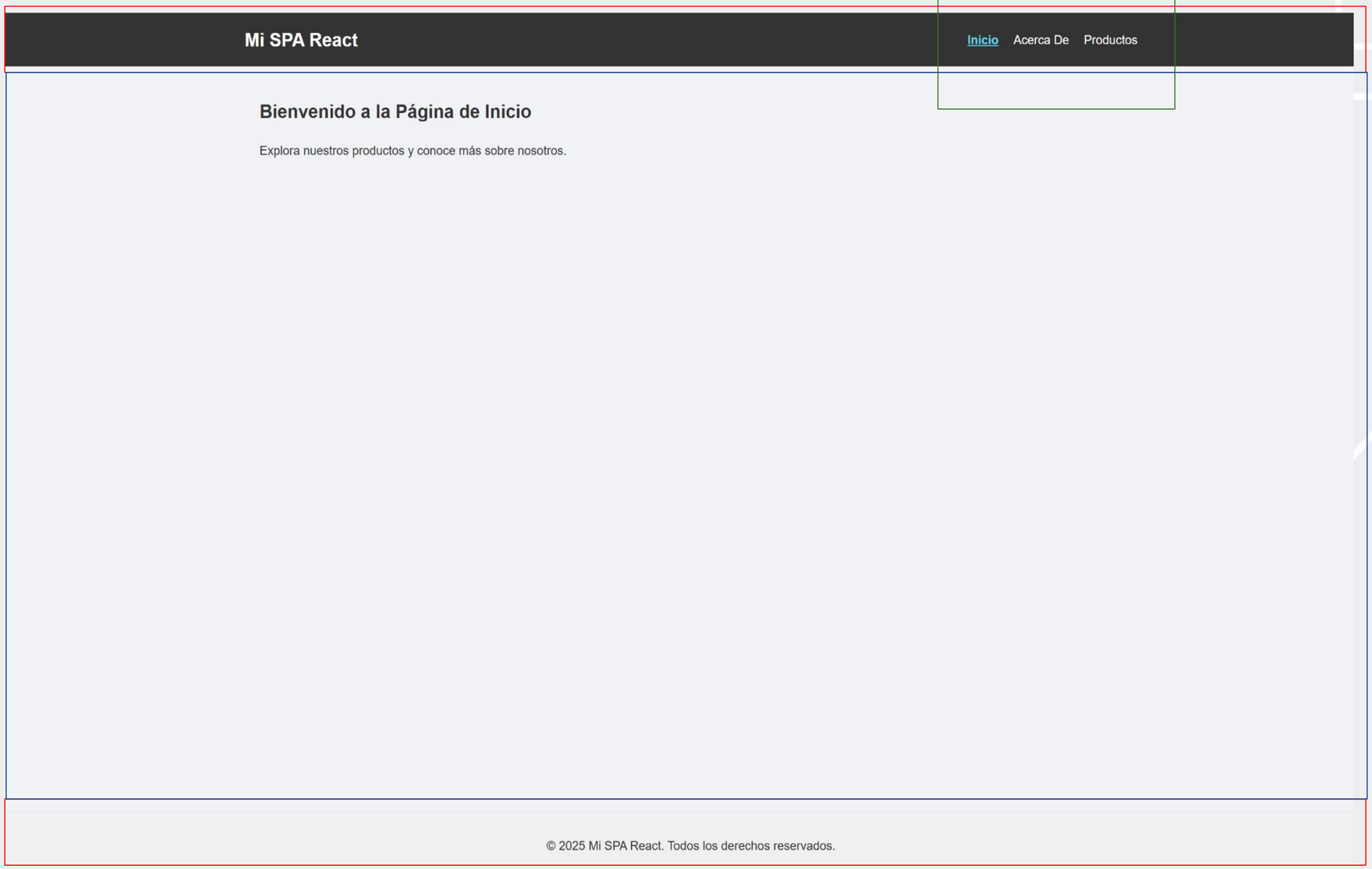
Como Link, pero añade una clase especial al enlace activo (útil para barras de navegación).

Outlet

Se usa en componentes de layout (padres) para renderizar las rutas hijas.

```
const router = createBrowserRouter([
 {
   path: '/',
   element: <Layout />,
   errorElement: <NotFound />,
children: [
    {
      index: true,
      element: <Home />,
    },
    {
      path: 'about',
      element: <About />,
    }
  ],
 },
  {
    path: '/login',
    element: <LoginPage />,
  },
]);
```

React Router Dom



React Router Dom

RouterProvider

El envoltorio principal para tu aplicación. Gestiona el enrutamiento. Requiere un objeto router creado con createBrowserRouter

createBrowserRouter

La forma moderna de definir tus rutas. Permite rutas anidadas, manejo de errores, y carga de datos

Link

El reemplazo de <a>. Cambia la URL y renderiza el componente sin recargar. Usa la prop to.

NavLink

Como Link, pero añade una clase especial al enlace activo (útil para barras de navegación).

Outlet

Se usa en componentes de layout (padres) para renderizar las rutas hijas.

```
<header>
  <nav>
    <Link to="/">
      Mi SPA React
    </Link>
    <ul >
      <li>
        <NavLink to="/">
          Inicio
        </NavLink>
      </li>
      <li>
        <NavLink to="/about">
          Acerca De
        </NavLink>
      </li>
      <li>
        <NavLink to="/products">
          Productos
        </NavLink>
      </li>
    </ul>
  </nav>
</header>
```

React Router Dom - Hooks

Su propósito principal es simplificar y hacer más legible el código de enrutamiento, permitiéndote interactuar con la URL, navegar programáticamente y obtener información de la ruta directamente dentro de tus componentes funcionales.

useLocation

Permite acceder al objeto location actual, que contiene información sobre la URL actual (como pathname, search, hash, etc.). Es útil para saber dónde estás en la aplicación y reaccionar a cambios en la URL, como mostrar un mensaje de error específico para una ruta no encontrada.

useParams

Permite acceder a los parámetros dinámicos de la URL definidos en tu ruta, como :userId o :productId.
Es ideal para páginas de detalles de productos, perfiles de usuario, o cualquier vista que dependa de un identificador en la URL

```
import { useParams } from "react-router-dom";
import Products from "../components/Products";

function ProductDetail() {

  let { id } = useParams();
   const product = Products.find(product =>
String(product.id) === id);

   return (
     <>
       <section key={id} className="details-section">
         <img src={product.image} alt=""></img>
         <div>
           <h3>{product.title}</h3>
           <p>{product.description}</p>
         </div>
       </section>
     </>
   )
}

export default ProductDetail;
```

React Router Dom

- ➔ Establecer rutas en nuestra aplicación ej: Home, About, User.
- ➔ Realizar redirecciones
- ➔ Acceso al historial del navegador
- ➔ Manejo de rutas con parámetros
- ➔ Páginas para el manejo de errores como 404

ROUTE

Con Route podemos definir las rutas de nuestra aplicación. Cuando definimos una ruta con Route le indicamos que componente debe renderizar.

```
import React from 'react';
import './App.css';
import {BrowserRouter as Router,Route
                } from "react-router-dom";
import Home from './pages/Home'

function App() {
  return (
    <Router>
      <div className="App">
         <Route exact path="/" component={Home} />
      </div>
    </Router>
  );
}
export default App;
```

React Router Dom

Path: la ruta donde debemos renderizar nuestro componente podemos pasar un string o un array de string.

Exact: Solo vamos a mostrar nuestro componente cuando la ruta sea exacta. Ej: /home === /home.

Strict: Solo vamos a mostrar nuestro componente si al final de la ruta tiene un slash. Ej: /home/ === /home/

Sensitive: Si le pasamos true vamos a tener en cuenta las mayúsculas y las minúsculas de nuestras rutas. Ej: /Home === /Home

Component: Le pasamos un componente para renderizar solo cuando la ubicación coincide. En este caso el componente se monta y se desmonta no se actualiza.

Render: Le pasamos una función para montar el componente en línea.

```
import React from 'react';
import './App.css';
import {BrowserRouter as Router,Route
                } from "react-router-dom";
import Home from './pages/Home'

function App() {
  return (
    <Router>
      <div className="App">
         <Route exact path="/" component={Home} />
      </div>
    </Router>
  );
}
export default App;
```

React Router Dom

LINK

Con Link vamos a poder navegar por nuestra aplicación, este componente recibe las siguientes propiedades.

To: le podemos pasar un string, object o una function para indicarle la ruta a la cual queremos navegar.

Replace: cuando es verdadero, y hacemos clic en el enlace reemplazará la entrada actual en la pila del historial en lugar de agregar una nueva.

```
<nav>
 <ul>
  <li>
    <Link to="/">Home</Link>
  </li>
 <li>
  <Link to="/about">About</Link>
 </li>
 <li>
  <Link to="/users">Users</Link>
 </li>
 <li>
  <Link to="/hola-mundo">Hello</Link>
 </li>
 </ul>
</nav>
```

Practica # 02

React Router

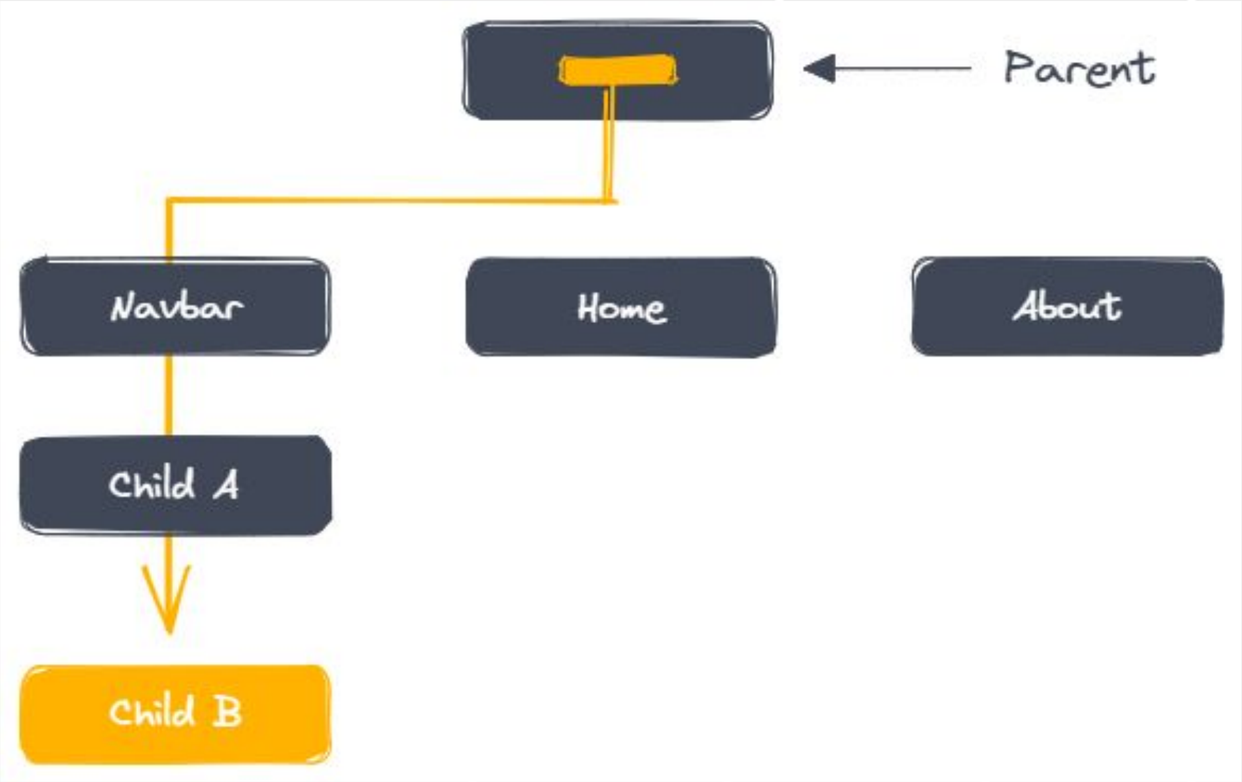
Practica # 03

Librerias UI React

React Context

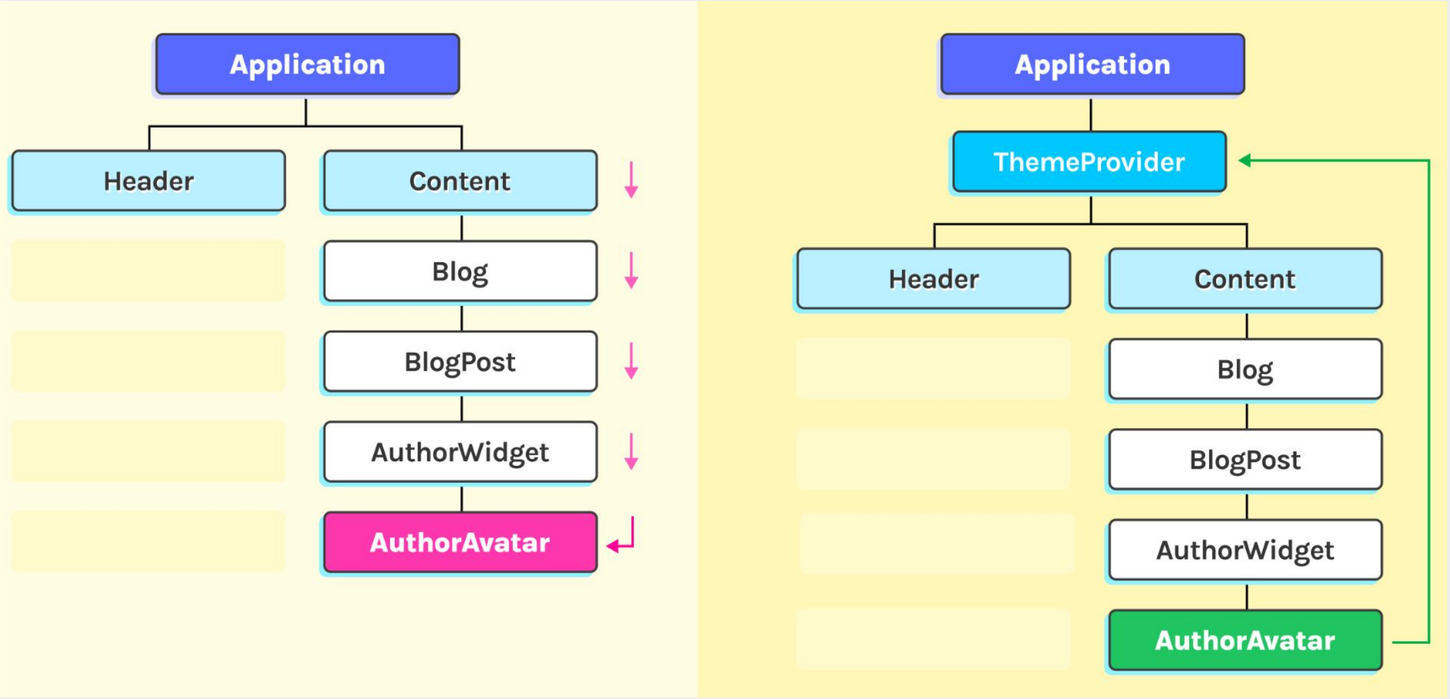
La Context API de React se utiliza para crear y compartir datos a nivel de aplicación sin necesidad de pasar props a cada componente que requiera dichos datos.

Esto se logra creando un contexto que almacena el valor y envolviendo los componentes que necesitan acceder a él dentro de un Provider.



La Context API consta de dos elementos principales:
Provider: Es el componente que proporciona el valor del contexto. Se encarga de "envolver" a los componentes que van a consumir los datos.
Consumer: Los componentes que acceden al valor del contexto.

React Context




```
export const MiContexto =createContext("");
```

```
import {createContext} from 'react';
const DataContext = createContext(“”);
export default DataContext;
```

```
const data = {
   valor1: 'Prueba',
   valor2: 'Context',
 };
 return (
   <DataContext.Provider value={data}>

   </DataContext.Provider>
 );
```

Context está diseñado para compartir datos que pueden considerarse globales para un árbol de componentes en React, como el usuario autenticado actual, el tema o el idioma preferido.

Context se usa principalmente cuando algunos datos tienen que ser accesibles por muchos componentes en diferentes niveles de anidamiento.

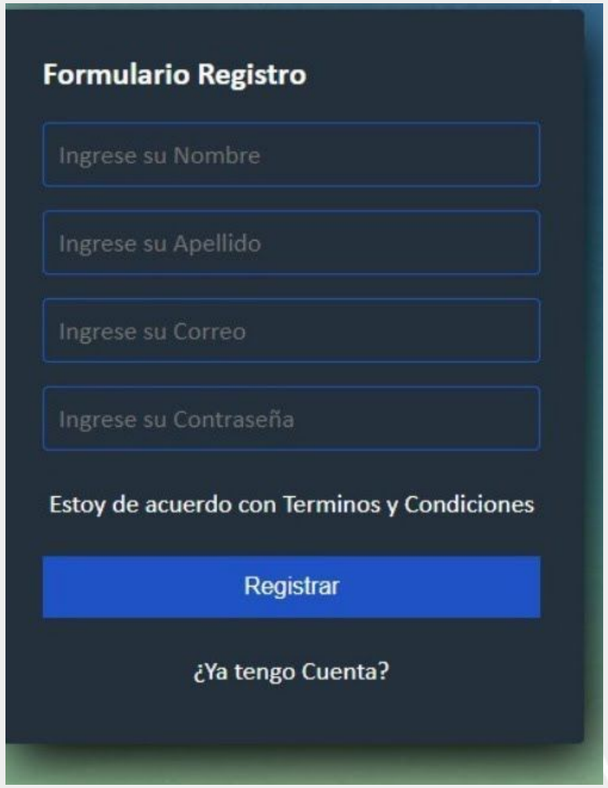
Hook useContext

```
import React, { useContext } from 'react';
import DataContext from '../store/data-context';

const PruebaContext= () => {
   const ctx = useContext(DataContext);
   return <p>{ctx.valor1}</p>;
};

export default PruebaContext;
```

Practica # 01
Context API

Registro de Usuarios con Contexto

Objetivo: Crear una app donde los usuarios puedan registrarse (nombre, email, telefono) y sus datos se guarden en un contexto global para mostrarlos en tiempo real en otra componente.



Lista de usuarios Crear usuario

ID	Nombre	Correo	Rol	Acciones
1	Abel	info@abelosh.com	Administrador	Editar | Eliminar
2	Julio Estrada	julio@gmail.com	Supervisor	Editar | Eliminar
3	Carlos Hernández	carlos@gmail.com	Vendedor	Editar | Eliminar
4	Rodrigo Estrada	rodrigo@gmail.com	Vendedor	Editar | Eliminar
5	Marta Elena Franco	marta@gmail.com	Supervisor	Editar | Eliminar
6	Matias Herrera	matias@gmail.com	Supervisor	Editar | Eliminar

PREGUNTAS Y RESPUESTAS