



```
src/pages/_app.tsx
  7 import { Container } from '@material-ui/core'; 63.32 KB (gzip: 20.38 kB)
  8 import { useApollo } from '../graphql/client';
  9
 10 import { lightTheme, darkTheme } from '../utils/theme';
 11 import useLocalStorage from '../hooks/useLocalStorage';
 12
 13 import NavBar from '../components/NavBar';
 14
 15 function App({ Component, pageProps }: AppProps) {
 16   const [currentTheme, setCurrentTheme] = useLocalStorage('theme-value', initialValue: 'light');
 17   const apolloClient = useApollo(pageProps.initialApolloState);
 18
 19   useEffect(() => {
 20     const jssStyles = document.querySelector(selectors: '#jss-server-side');
 21     if (jssStyles) {
 22       jssStyles.parentElement.removeChild(jssStyles);
 23     }
 24   }, [deps]);
 25
 26   return (
 27     <>
 28       <Head>
 29         <title>ECU-DEV</title>
 30         <meta name="viewport" content="minimum-scale=1, initial-scale=1, width=device-width, height=device-height, user-scalable=no" />
 31     </>
 32   );
 33 }
 34
 35 export default App;
```

# Event handlers

Date	@February 21, 2023 → February 25, 2023
Assign	
Status	
Tasks	

Los Event Handlers son funciones que creamos que se van a llamar como respuesta a una interaccion de un usuario, estas pueden ser clicks, hovers, focus y muchas otras, las mismas que en el DOM.

- Mientras que en el dom se escriben en lowercase, en react lo hacemos en camelCase.
- En React, cuando queremos evitar el comportamiento default debemos llamar preventDefault dentro del event handler.

```
const handleSubmit = (e) => {
  e.preventDefault();
  // TODO: handle submit data
}
```

- No es necesario que agreguemos un event listener con addEventListener como lo hacemos en JS, solo tenemos que entregarle la funcion al renderizar nuestro componente.

```
// class components
<button onClick={this.handleClick}> Esto es un Boton. </button>

// Functional components
<form onSubmit={handleSubmit}>
  <button type="submit">Submit</button>
</form>
```

- Nuestros event handlers pueden recibir argumentos cuando los llamamos, esto lo hacemos como se haria con cualquier otra funcion.

```
<button onClick={(e) => handleEditRow(row, e)}>Edit Row</button>
```

- Recuerden que las funciones que usamos como event handlers no deben ser llamadas.

```
// correcto
<button onClick={handleClick}>Click</button>
// incorrecto
<button onClick={handleClick()}>
```

- En nuestros componentes hay event propagation, es decir que nuestros eventos empiezan en el componente donde fueron ejecutados pero despues sube por el DOM tree, esto puede generar efectos indeseados, en casos donde no queremos que esto pase, como por ejemplo tenemos un boton dentro de un componente padre que tiene un onClick, debemos usar stopPropagation().

## Ejercicio en clase:

Crear un ejemplo donde usemos componentes para mostrar una tabla de estudiantes con el nombre, edad, Carrera que estudia y el semestre actual, cada uno de esos estudiantes va a tener un boton para eliminarlo, uno para sumarle 1 al semestre y otro

para restarle 1 al semestre, al dar click a cualquier parte del row menos los botones se abre un modal para modificar ese estudiante, ademas debe existir un boton para agregar un nuevo estudiante a la tabla y se tiene que actualizar de forma automatica al dar click en el boton de guardar.

Cualquier modificacion que se haga al estudiante debe verse reflejada en la interfaz mediante el uso de un estado global (redux o context).