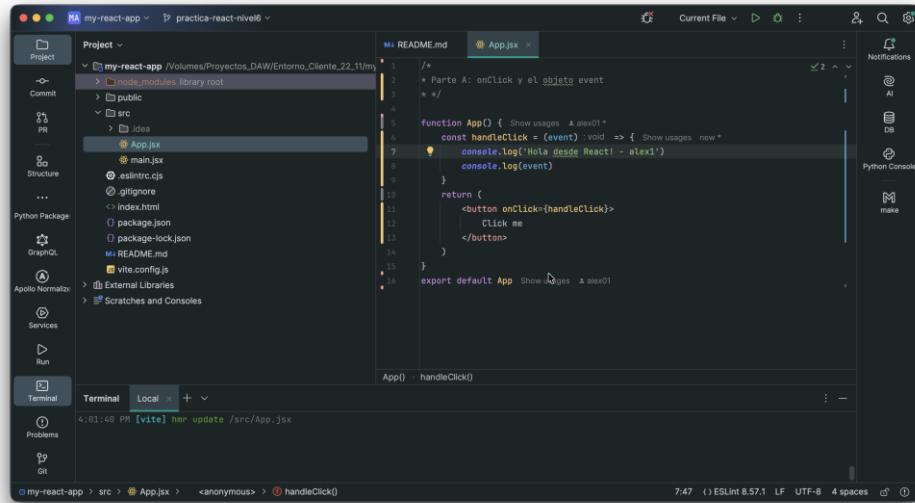


1- Parte A: onClick y el objeto event

- Código

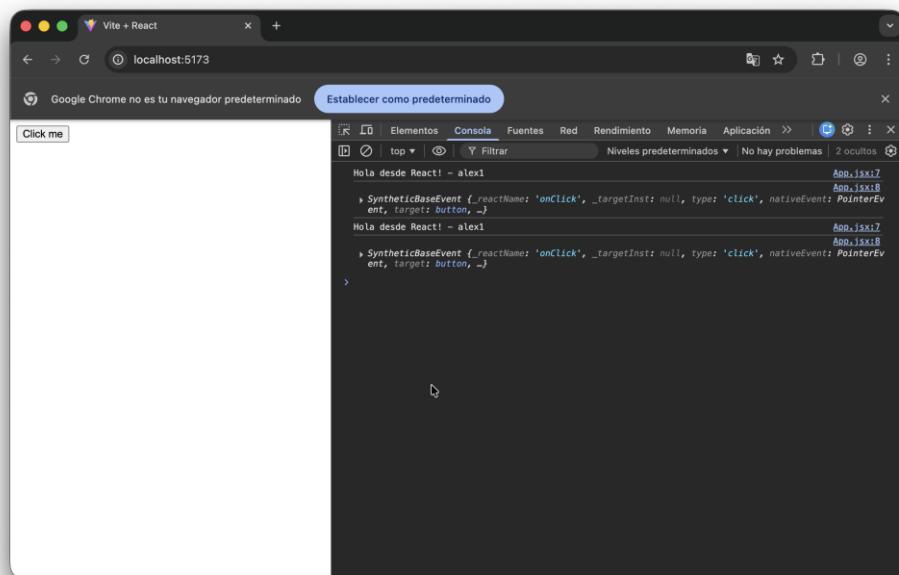


The screenshot shows a code editor with the file `App.jsx` open. The code defines a function `handleClick` that logs "Hola desde React! - alex1" and the event object to the console. It then returns a button with the `onClick` attribute set to `handleClick`, which has the value `handleClick`. The code editor interface includes a sidebar with project files like `README.md`, `index.html`, and `vite.config.js`, and a terminal at the bottom showing the command `npm update /src/App.jsx`.

```

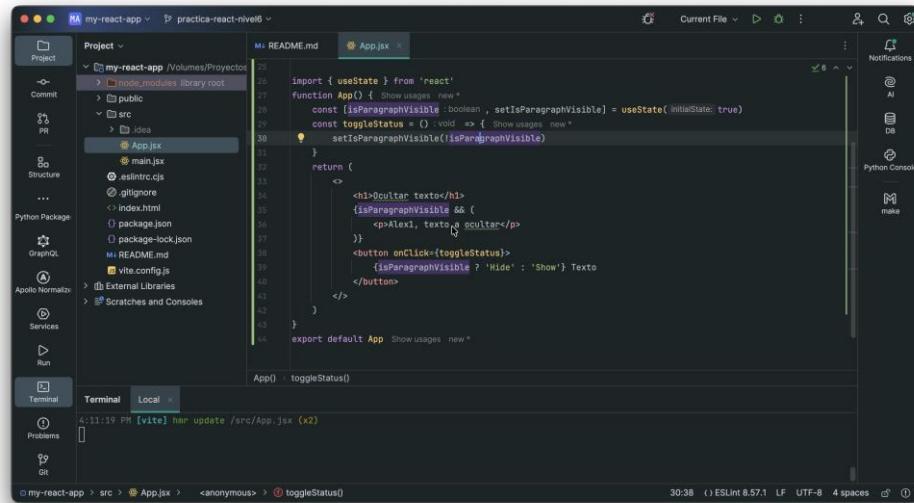
    1 // Parte A: onClick y el objeto event
    2
    3
    4 function App() { Show usages & alex0!
    5   const handleClick = (event) => { Show usages new!
    6     console.log("Hola desde React! - alex1")
    7     console.log(event)
    8   }
    9   return (
10     <button onClick={handleClick}>
11       Click me
12     </button>
13   )
14 }
15 export default App Show usages & alex0!
```

- Resultado



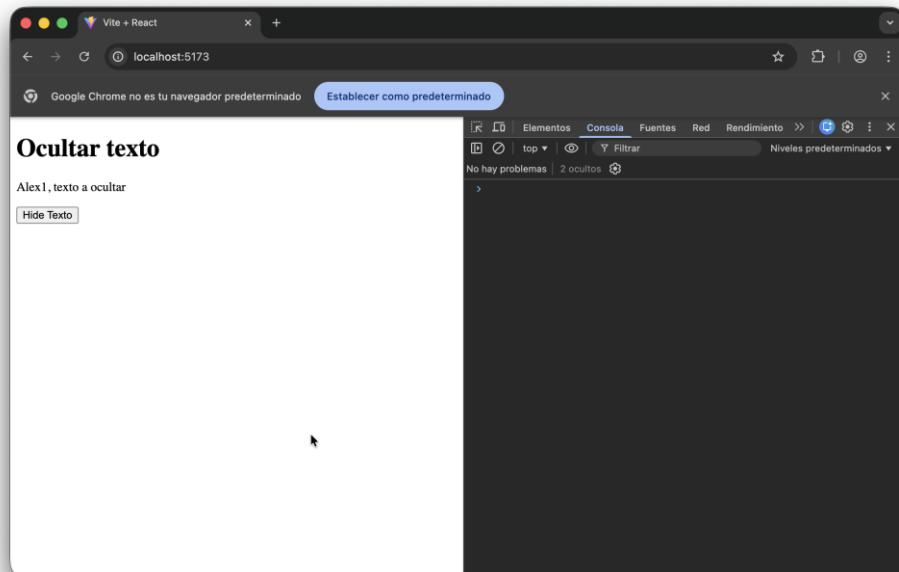
2- Parte B: Cambiar la UI con estado + evento

- La función que tiene el botón funciona en base al estado, de inicio empieza en true, con el texto en pantalla, al pulsar el botón cambia, ya que la función para a ser false

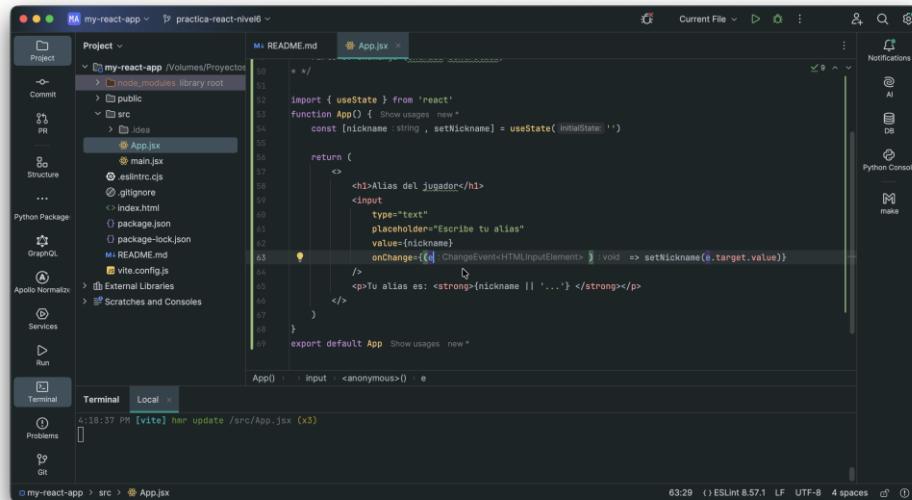


The screenshot shows the VS Code interface with the following details:

- Project Explorer:** Shows the project structure with files like README.md, App.jsx, main.jsx, .eslintrc.js, .gitignore, index.html, package.json, package-lock.json, .vite.config.js, and .idea.
- Code Editor:** Displays the `App.jsx` file containing React code. The code uses the `useState` hook to manage the visibility of a paragraph. It includes a button that toggles the state between "Show" and "Hide".
- Terminal:** Shows the command `vite update /src/App.jsx` being run at 4:11:19 PM.
- Status Bar:** Shows the time as 30:38, ESLint version 8.57.1, LF, UTF-8, 4 spaces, and a help icon.



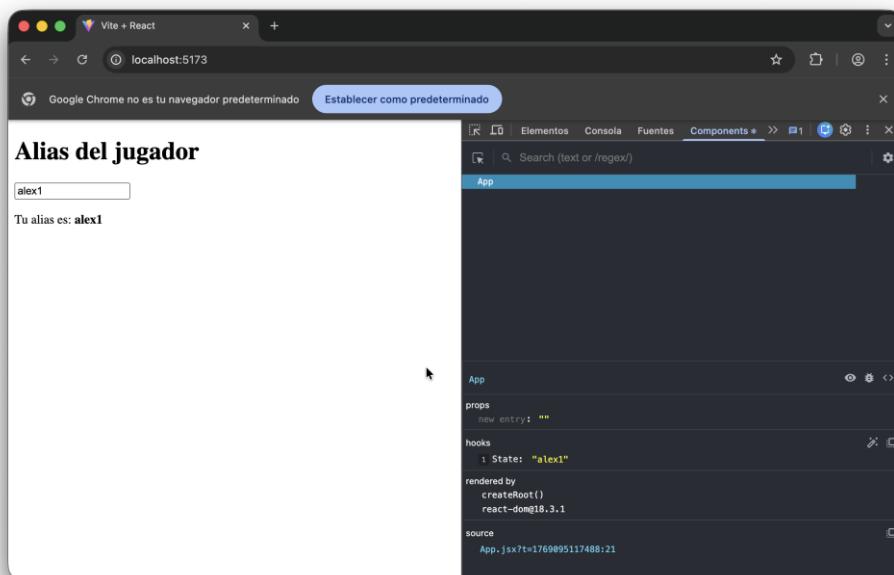
3- Parte C: onChange (entrada controlada)



```

1 // ...
2 import { useState } from 'react';
3 function App() {
4   const [nickname, setNickname] = useState('InitState');
5
6   return (
7     <>
8       <h1>Alias del jugador</h1>
9       <input
10         type="text"
11         placeholder="Escribe tu alias"
12         value={nickname}
13         onChange={({ target }) => setNickname(target.value)}
14       />
15       <p>Tu alias es: <strong>{nickname} || '...'"</strong></p>
16     </>
17   );
18 }
19 export default App;
20 
```

The screenshot shows the VS Code interface with the project structure on the left and the code editor on the right. The code editor displays an `App.jsx` file using the `useState` hook to manage a nickname and an `<input>` element to update it.



4- Parte D: onSubmit (formulario)

- He agregado una nueva variable, de **msj** para que cuando mande el correo, también reciba el **msj** que se le ha escrito.

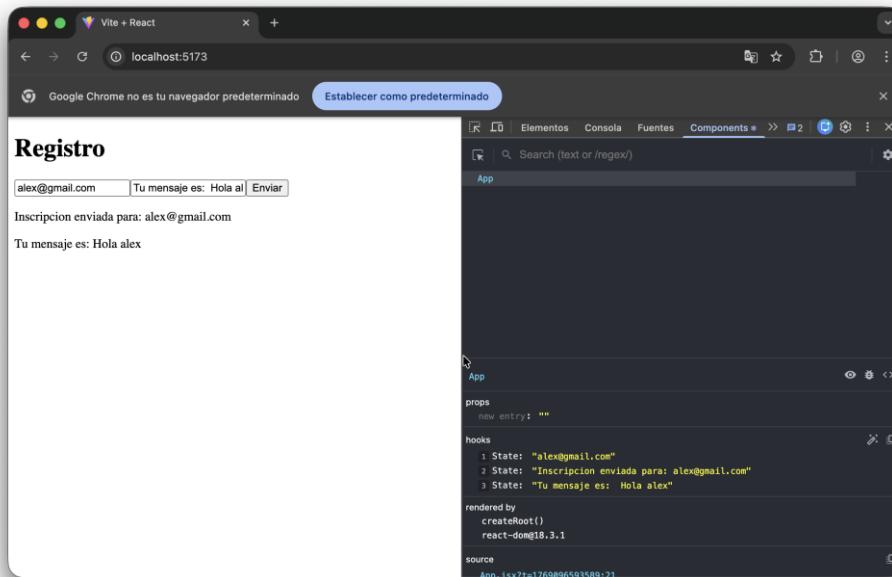
```

function App() {
  const [msj, setMsj] = useState(initialState)
  ...
  return (
    <div>
      <h1>Registro</h1>
      <form onSubmit={handleSubmit}>
        <input
          type="email"
          placeholder="tu@email.com"
          value={email}
          onChange={(e: ChangeEvent<HTMLInputElement>) => setEmail(e.target.value)}
          required
        />
        <input
          type="msg"
          placeholder="Escribe tu msj"
          value={msj}
          onChange={(e: ChangeEvent<HTMLInputElement>) => setMsj(e.target.value)}
          required
        />
        <button type="submit">Enviar</button>
      </form>
    </div>
  )
}

const handleSubmit = (e: void) => {
  e.preventDefault()
  setMsg('Inscripción enviada para: ' + email)
  setMsj('Tu mensaje es: ' + msj)
}

```

- Resultado



5- Mini reto

- He usado tres variables **hideText**, **contador** y **modo**
- **HideText:** Este es simple, método que variable en función al botón de true o false muestra el texto oculto.

- **Contador:** En el botón, he añadido llamado a setContador y que vaya suman de 1 en 1 para que sea incrementable.
- **Modo:** He creado el método **msj**, para para luego llamarlo en el input y he llamado la variable al final del Form **{modo && <p>{modo}</p>}**
- Para que cuando escribas en el input, se vea el texto cambiando.

```

function App() {
  const [modo, setModo] = useState("modo");
  const [contador, setContador] = useState(0);
  const [hideText, setHideText] = useState(true);

  const handleTextChange = (e) => {
    setModo(e.target.value);
  };

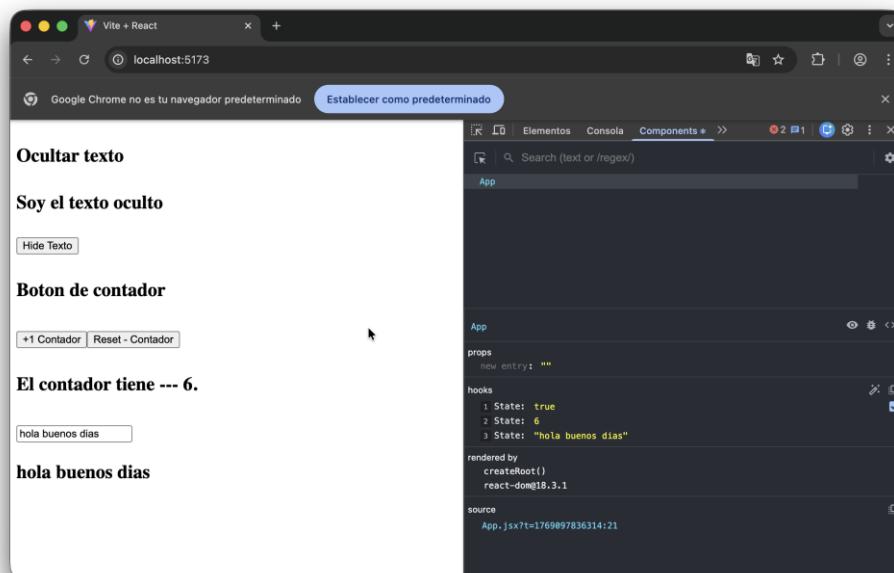
  const handleCounterClick = () => {
    setContador(contador + 1);
  };

  const handleTextClick = () => {
    setHideText(!hideText);
  };

  return (
    <div>
      <h4>Boton de contador</h4>
      <button onClick={() => setContador(contador + 1)}>+1 Contador</button>
      <button onClick={() => setContador(0)}>Reset - Contador</button>
      <br/>
      <h4>El contador tiene --- {contador}</h4>
      <br/>
      <h4>Formulario</h4>
      <form onSubmit={msj}>
        <input type="modo" placeholder="modo" value={modo} onChange={(e) => handleTextChange(e)} required />
      </form>
      {modo && <p>{modo}</p>}
    </div>
  );
}

export default App;
  
```

- Resultado



PREGUNTAS EXTRA:

1. ¿Qué diferencia hay entre onClick y onSubmit?

- OnClick: Al ejecutarlo llama a un elemento ya sea un div, o alguna acción
- OnSubmit: Se utiliza cuando se envía un formulario, este es recomendable para formularios.

2. ¿Por qué usamos e.preventDefault() en un formulario?

- Por defecto, a la hora de hacer una acción de mandar un formulario la página se recarga, con esto se evita que pase eso

3. ¿Qué es una "entrada controlada" y por qué usamos value + onChange?

- Por ejemplo, un input, ya que es controlada por React mediante su estado, esto hace que nosotros tengamos el control.

4. En tu mini-reto, que estado(s) manejas y que evento(s) los actualizan?

- Los estados que utilizo son, hideText, Contador y modo
- Los eventos:
- Onclick para mostrar u ocultar el texto.
- OnChange actualiza el modo mientras escribes.