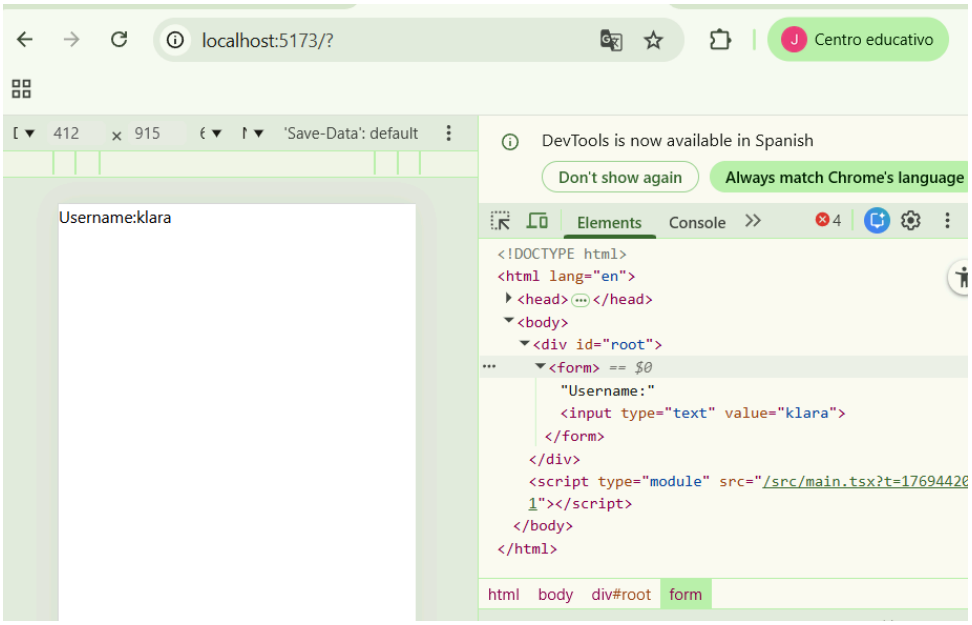
Juan Antonio Zamora Nieto |26/01/2026

REACT NIVEL 8

PARTE A: Input controlado (Controlled Component):

```
src > App.tsx > default
1  import { useState } from 'react';
2
3  function App() {
4    const [username, setUsername] = useState('');
5
6    return (
7     <form>
8        Username:
9        <input type='text' value={username} onChange={(e) => setUsername(e.target.value)} />
10    </form>
11   );
12 }
13
14 export default App;
```

(Ilustración 1. Captura de pantalla del código en el cual creamos un estado "`username`" y lo vinculamos al `input`, por lo que coge su valor. Elaboración propia.)



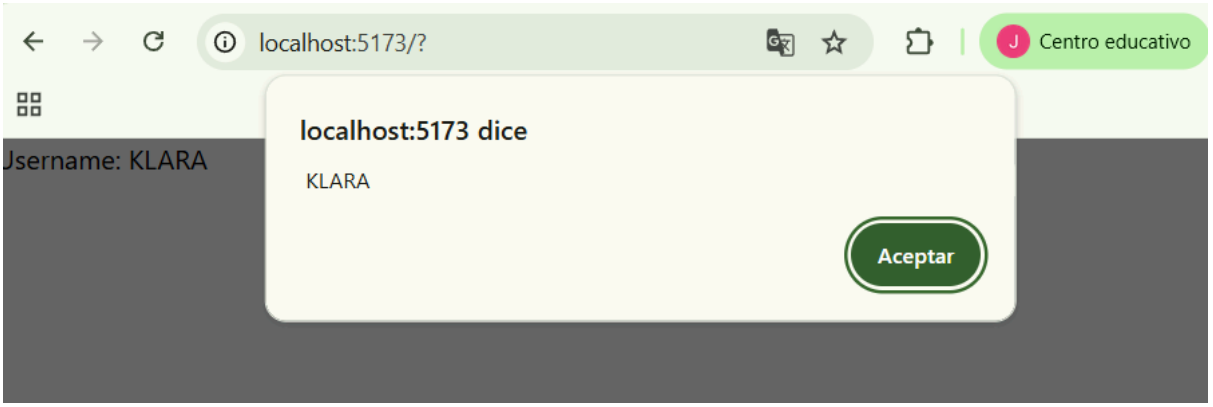
(Ilustración 2. Captura de pantalla de la App en la que ponemos el nombre de usuario, y en la consola de nuestro navegador podemos observar el "`value = 'klara'`", justo el nombre puesto en el input, verificando que están conectados. Elaboración propia.)

PARTE B: Envío del formulario (onSubmit):

src > App.tsx > default

```jsx
import { useState } from 'react';

function App() {
  const [username, setUsername] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(username);
  };

  return (
    <form onSubmit={handleSubmit}>
      Username:
      <input
        type="text"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <button type="submit">Submit</button>
    </form>
  );
}

export default App;
```

(Ilustración 3. Captura de pantalla del código, añadiendo en este caso una variable que evita el refresco del navegador y nos muestra el valor actual. Lo añadimos en el "`form`" "`onSubmit`" y no se refrescará, viendo el "`alert`" cuando le damos a "`Submit`" . Elaboración propia.)

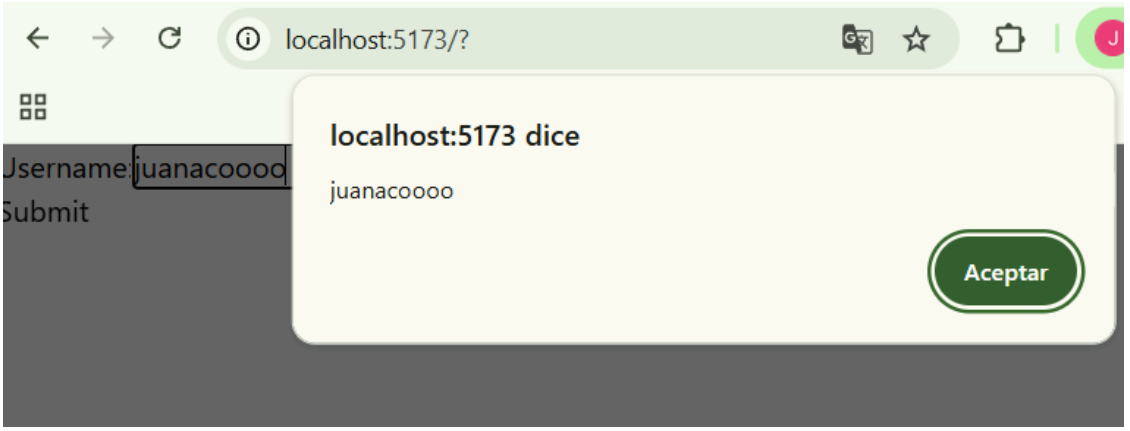


(Ilustración 4. Captura de pantalla de la App mostrando como la página no se refresca, y nos muestra el valor cuando le damos a "`Submit`". Elaboración propia.)

PARTE C: Validación básica:

(Ilustración 5. Captura de pantalla del código mostrando la pequeña validación añadida que funciona a tiempo real gracias a “onChange”, mostrando que tiene menos de 6 letras de longitud. Elaboración propia.)

(Ilustración 6. Captura de pantalla de la App mostrándonos como la variable "`usernameError`" es `true`, por lo que se nos muestra en pantalla que no se puede enviar. De fondo se puede observar cómo sale :"El username debe tene…r más de 6 carácteres". Elaboración propia.)



(Ilustración 7. Captura de pantalla de la App en la que una vez cumplido el requisito de tener los carácteres mínimos de longitud, nos dejaría enviar el formulario. Elaboración propia.)

PARTE D: Código completo de referencia:

App.tsx > App

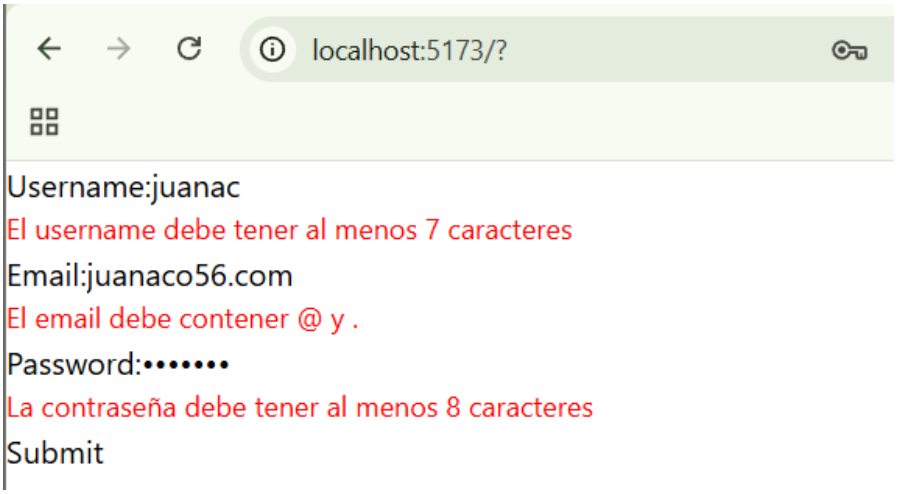
```tsx
import { useState } from 'react';

function App() {
  const [username, setUsername] = useState('');
  const [usernameError, setUsernameError] = useState('');

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

    setUsername(value);

    if (value.length <= 6) {
      setUsernameError('El username debe tener más de 6 caracteres');
    } else {
      setUsernameError('');
    }
  };

  const handleSubmit = (e) => {
    e.preventDefault();

    if (usernameError) {
      alert('No se puede enviar: el formulario contiene errores');
    } else {
      alert(username);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      Username:
      <input
        type="text"
        value={username}
        onChange={handleUsername}
      />
      <p>{usernameError}</p>

      <button type="submit">Submit</button>
    </form>
  );
}

export default App;
```

(Ilustración 8. Captura de pantalla del código de la parte anterior, sin haber necesitado el de la profesora. Elaboración propia.)

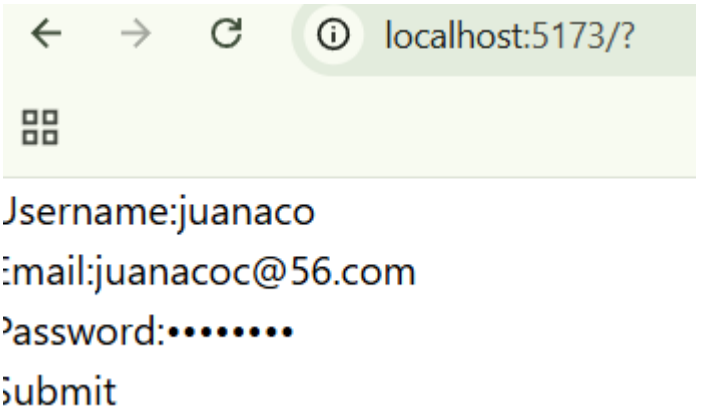
MINIRETILLO:

```tsx
App.tsx > ...
import { useState } from 'react';

function App() {
  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  const [usernameError, setUsernameError] = useState('');
  const [emailError, setEmailError] = useState('');
  const [passwordError, setPasswordError] = useState('');

  const handleUsername = (e) => {
    const value = e.target.value;
    setUsername(value);

    if (value.length < 7) {
      setUsernameError('El username debe tener al menos 7 caracteres');
    } else {
      setUsernameError('');
    }
  };

  const handleEmail = (e) => {
    const value = e.target.value;
    setEmail(value);

    if (!value.includes('@') || !value.includes('.')) {
      setEmailError('El email debe contener @ y .');
    } else {
      setEmailError('');
    }
  };

  const handlePassword = (e) => {
    const value = e.target.value;
    setPassword(value);

    if (value.length < 8) {
      setPasswordError('La contraseña debe tener al menos 8 caracteres');
    } else {
      setPasswordError('');
    }
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    alert('Formulario enviado correctamente');
  };

  const hasErrors = usernameError || emailError || passwordError;

  return (
    <form onSubmit={handleSubmit}>
      <div>
        Username:
        <input type="text" value={username} onChange={handleUsername} />
        <p className="error">{usernameError}</p>
      </div>

      <div>
        Email:
        <input type="text" value={email} onChange={handleEmail} />
        <p className="error">{emailError}</p>
      </div>

      <div>
        Password:
        <input type="password" value={password} onChange={handlePassword} />
        <p className="error">{passwordError}</p>
      </div>

      <button type="submit" disabled={hasErrors}>
        Submit
      </button>
    </form>
  );
}

export default App;
```

(Ilustración 10. Captura de pantalla del código, haciendo lo mismo que previamente, pero validando el correo, la contraseña, y deshabilitamos el botón "`Submit`" si hay algún error. Mostramos los errores en rojo gracias a que en "`Main.tsx`" tenemos importado el css. Elaboración propia.)



(Ilustración 11. Captura de pantalla de la App, donde mostramos que nos saltan los errores en rojo cuando no los tenemos bien completados. Elaboración propia.)



(Ilustración 12. Captura de pantalla de la App, donde mostramos ahora sí, los campos rellenos y completamente funcional. Elaboración propia.)

(Ilustración 13. Captura de pantalla del código `.css` que nos da el color rojo en nuestros errores. Elaboración propia.)