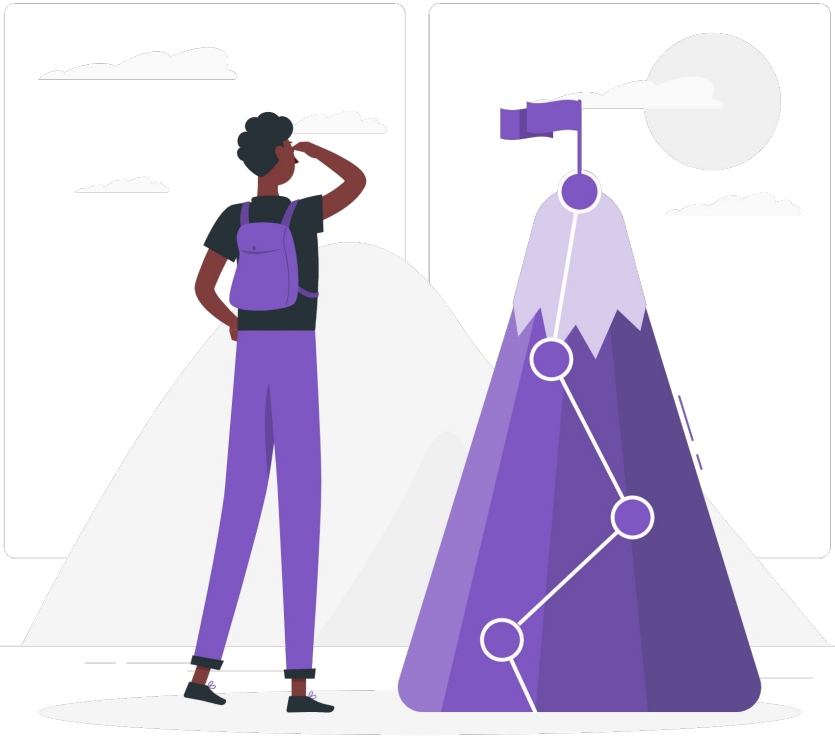

# Formularios en React

DEV.F:
DESARROLLAMOS(PERSONAS);

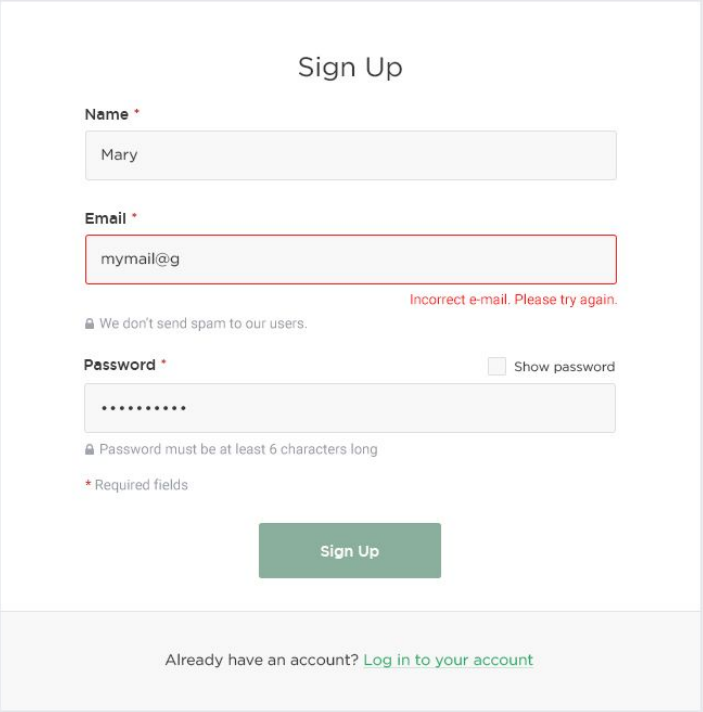
Elaborado por: César Guerra
www.cesarguerra.mx



# Objetivos

- Relevancia de los formularios
- Comprender la diferencia entre formularios tradicionales con HTML (*no controlados*) vs los formularios en React (*controlados*)
- Aprender a evitar que un formulario recargue la página (*prevent.default*)
- Entender el flujo del uso de formularios en React.
- Implementar librerías adecuadas para el manejo de formularios.

DEV.F::



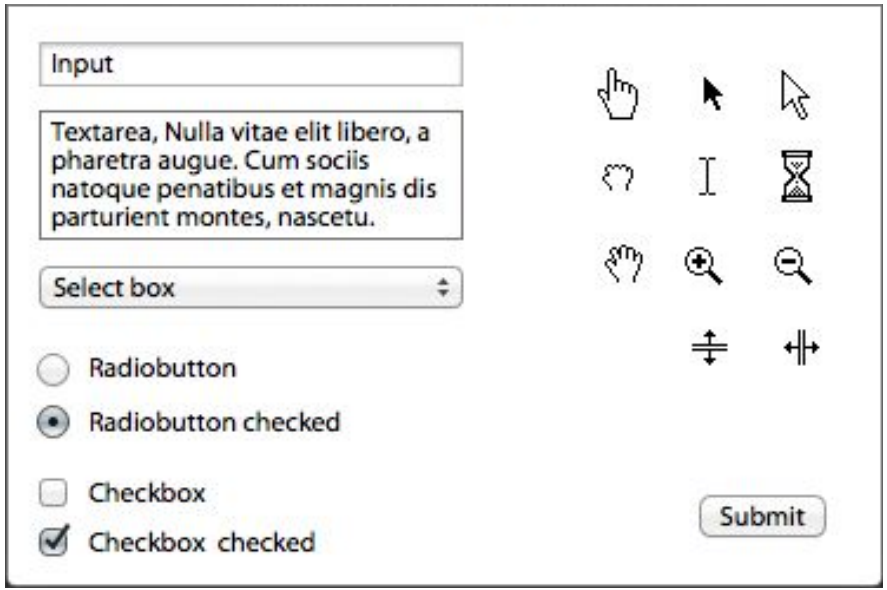

# Relevancia de Formularios

Los formularios son uno de los elementos más usados hoy en día en cualquier sistema, incluyendo por supuesto las aplicaciones web.

**Nos permiten poder introducir información al sistema** para que éste lo almacene o realice acciones, por ejemplo:

- Hacer inicio de sesión
- Registrar una cuenta
- Dar de alta algún item a la base de datos
- Visualizar la información capturada y modificarla
- Formularios de contacto, entre otros

DEV.F::




# ¿Cómo funciona un Formulario? (1)

Cada elemento de un formulario contiene un valor.

Un valor puede ser escrito (***input***, ***textarea***) o puede ser seleccionado ( ***checkbox***, ***select***, ***radiobutton***, etc.) por el usuario en el navegador.

Cuando el valor de un elemento del formulario es cambiado (por el hecho de escribir o seleccionar), su valor es actualizado.

DEV.F::

```
> b = document.querySelector('#vue-datatable-34444 > tbody > tr:nth-child(2) > td:nth-child(4) > input')
<- <input data-family="precovarejo" value="11.00" type="text" class="animated" data-coord-x="1" data-coord-y="-1" data-last-val="11.00" data-produto="25fdd2d1-619f-42d7-aa52-90df57c67972" maxlength="7">
> b.value
<- "11,00"
```

# ¿Cómo funciona un Formulario? (2)

Podemos obtener el valor de cada elemento del formulario mediante la propiedad ***.value***

Así mismo podemos usar ***.value*** para asignar un nuevo valor a ese elemento.

DEV.F::


```html
<form onSubmit="/myaction.php">
   <input type="email" name="email" id="email">
   <input type="password" name="password" id="password">
   <input type="submit" value="Submit">
</form>
```

# ¿Cómo funciona un Formulario? (3)

Para indicarle a un formulario la acción qué debe realizar, se utiliza el atributo ***onSubmit*** dentro de la etiqueta form de HTML.

Generalmente, un botón es el que desencadena el envío del formulario, este debe tener el atributo ***type=”submit”.***

Normalmente, esta acción es procesada por otra página, por lo qué ocurre una recarga de la página. Sin embargo, en React no debemos permitir qué esta recarga ocurra (***preventDefault()***)

DEV.F::

# Formularios
# Controlled vs Uncontrolled

DEV.F:
DESARROLLAMOS(PERSONAS);

DEV.F

En React, existen al menos 2 formas de manejar la información en nuestros componentes de formulario:

1. Componentes no controlados
2. Componentes controlados

```jsx
function App() {
  function onSubmit() {
    console.log("Valor de Email: " + window.email.value);
    console.log("Valor de Password: " + window.password.value);
  }
  return (
    <form onSubmit={onSubmit}>
      <input type="email" name="email" id="email" required />
      <input type="password" name="password" id="password" required />
      <input type="submit" value="Submit" />
    </form>
  );
}
```

# Componentes No Controlados
## (uncontrolled components)

**Un componente no controlado es aquel cuya información del formulario es manejada por el propio DOM.**

Es decir, los valores del formulario son tomados directamente del valor almacenado en el DOM.

**Se dice qué es “no controlado” por el hecho de que esos componentes del formulario no son controlados por un estado de React**, por lo tanto React no tiene control sobre el valor de los mismos.

DEV.F::

```jsx
const SimpleForm = () => {
    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')

    const handleSubmit = (event) => {
        event.preventDefault()
        const submittedData = JSON.stringify({ email, password })
        console.log(submittedData)
    }

    return (
      <form className='form'>
          <label htmlFor='email'>Email</label>
              <input type='text' name='email'
               onChange={(event)=> setEmail(event.target.value)}
              />

          <label htmlFor='password'>Password</label>
              <input type='password'name='password'
              onChange={(event)=> setPassword(event.target.value)}
          />

          <button onClick={handleSubmit}> Login </button>
      </form>
    )
}
```

# Componentes Controlados
## (controlled components)

**Un componente controlado en React es aquel cuya información (data) es manejada por un estado de React (state).**

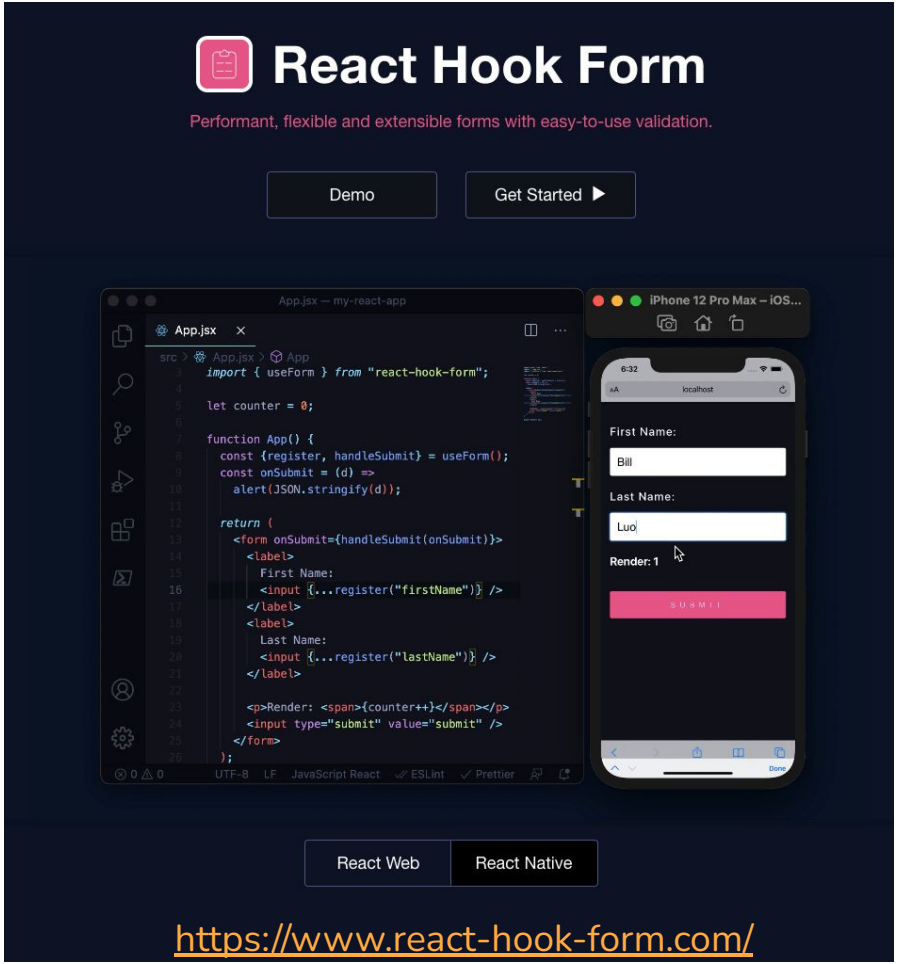
La primera consiste en usar un estado en el componente que manejara la información del formulario. Esto se le conoce como **controlled component**.

DEV.F::

| Features | Uncontrolled | Controlled |
| --- | --- | --- |
| One time value retrieval (e.g. on submit) | ✅ | ✅ |
| Validating on submit | ✅ | ✅ |
| Instant field validation | ❌ | ✅ |
| Conditionally disabling submit button | ❌ | ✅ |
| Dynamic Inputs | ❌ | ✅ |

# React Hook Form (useForm)

DEV.F:
DESARROLLAMOS(PERSONAS);



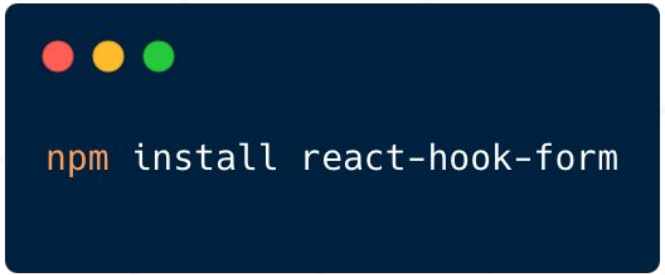

# React Hook Form

## Librería de manejo de formularios

**React Hook Form** nos ofrece la capacidad de desarrollar nuestros formularios de manera no controlada, independizando todo cambio que pueda producirse en cada uno de los elementos del formulario, evitando con ello renders innecesarios, haciendo uso de hooks y con una sencillez total.

DEV.F::

# Instalación de React Hook Form




Documentación:
Get Started | React Hook Form - Simple React forms validation (react-hook-form.com)

# Validaciones con Yup

DEV.F:: DESARROLLAMOS(PERSONAS);

```
import { object, string, number, boolean } from "yup";
const schema = object({
  name: string()
    .required("El campo nombre es obligatorio")
    .min(1, "El nombre tiene que tener al menos un carácter")
    .max(100, "El nombre no puede superar los 100 carácteres"),
  alias: string().optional(),
  age: number()
    .required("La edad es obligatoria")
    .positive("La edad tiene que ser positiva")
    .max(90, "La edad no puede superar los 90"),
  email: string()
    .required("El email es obligatorio")
    .email("El email no tiene un formato válido"),
  isChosenOne: boolean(),
});
```

Sitio Oficial: [GitHub - jquense/yup: Dead simple Object schema validation]

# ¿Qué es Yup?

**Yup es una librería para validar formularios con JavaScript**, que da soporte tanto a aplicaciones de navegador, como a servidores con NodeJs.

**Con Yup, podremos definir un conjunto de reglas, y averiguar si los datos introducidos por el usuario las cumplen, o no.** Para ello, su API provee de un sistema de creación de esquemas (schemas).

Referencia: Validar Formularios con Yup:
https://libreriasjs.com/libreria-javascript-validar-formularios-yup/

DEV.F::

# Instalación de Yup para React Hook Form

```
npm install @hookform/resolvers yup
```

Documentación:
https://www.react-hook-form.com/get-started#SchemaValidation

# ¡ Vamos al Código !


¡ GRACIAS !
CésarGuerra
www.cesarguerra.mx