Formularios en React

Módulo 7

Elaborado por: Jesua Luján

Jesua Hadai Luján

DEV.F:

DEV.F:: | 🟢 Módulo 1: Envío de formularios en React 🔥

🎯 ***Objetivo:*** Dominar la creación y gestión de formularios en React.

🎓 Contenido:

✅ Entender la relevancia de los formularios. 🤔

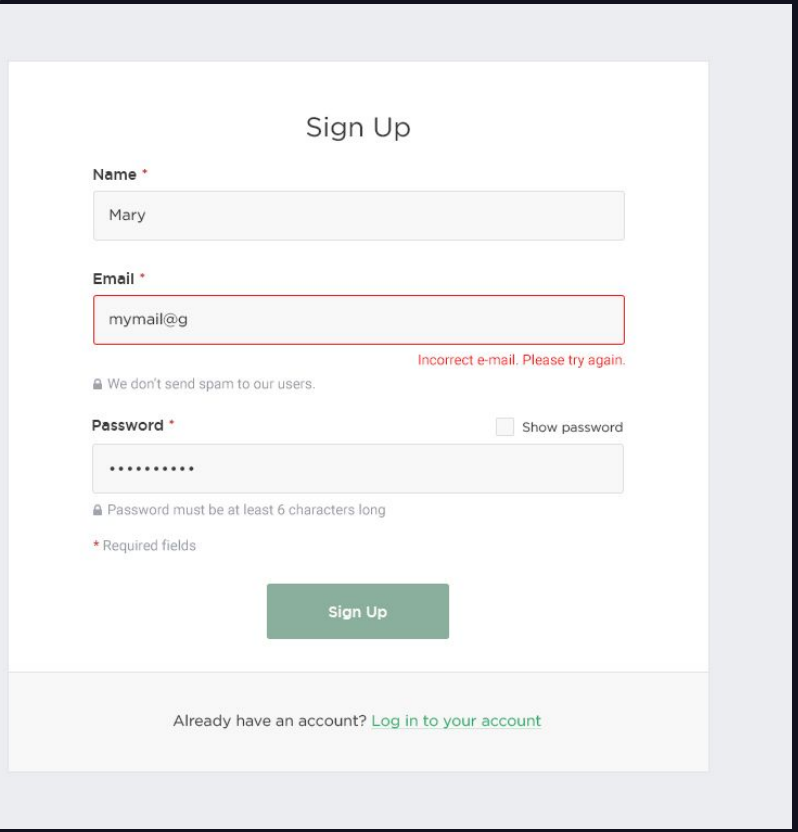
✅ Ver Formulario no Controlado (html-tradicional) vs Formulario Controlado (react).🤦‍♀️

✅ Aprender a evitar que un formulario recargue la página (*e.prevent.default*).🤦‍♀️

✅ Implementar librerías adecuadas para el manejo de formularios.😈

ESCUELA DE HACKERS

🚀 Relevancia de Formularios



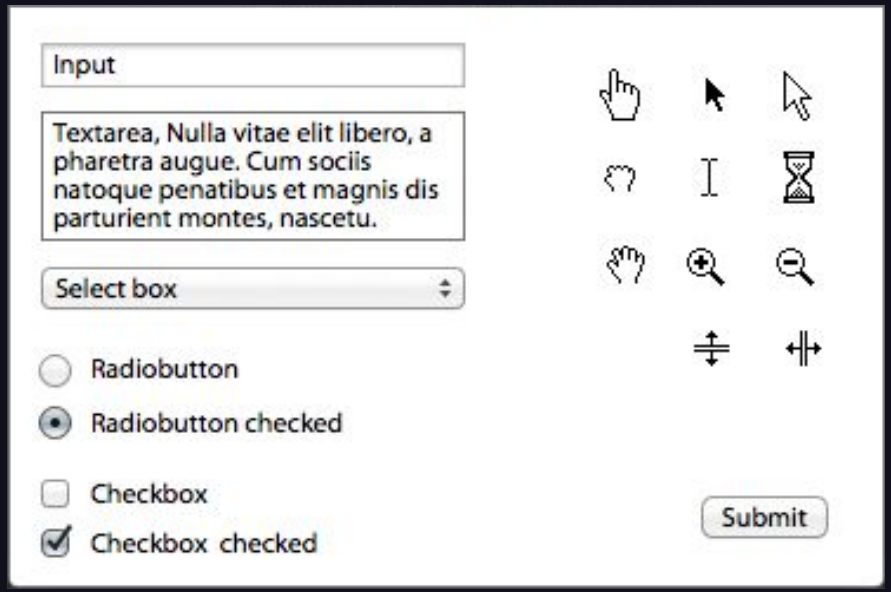
Los formularios son uno de los elementos más usados hoy en día en cualquier sistema, incluyendo por supuesto las aplicaciones web.

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 ítem a la **base de datos**
- Visualizar la información capturada y modificarla
- Formularios de contacto, entre otros

DEV.F.:

🚀 ¿Cómo funciona un Formulario?



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.

(1)

ESCUELA DE HACKERS

🚀 ¿Cómo funciona un Formulario?

```
> 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"
```

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

DEV.F::


```
<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?

Para indicarle a un formulario la acción que 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 que ocurre una recarga de la página. Sin embargo, en React no debemos permitir que esta recarga ocurra (***e.preventDefault()***).

Controlados vs no Controlados

Formularios

DEV.F:

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

ESCUELA DE HACKERS

👩‍💻 Componentes No 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>
  );
}
```

(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 que 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::

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**.

Característica	Formularios Controlados	Formularios No Controlados
Definición	Valor gestionado por el estado de React.	Valor gestionado por el DOM.
Acceso a los valores	Directo desde el estado de React.	Usando useRef al momento del submit.
Rendimiento	Puede causar más renders.	Mejor rendimiento, sin re-renderizados innecesarios.
Simplicidad	Más código para cada input.	Menos código, más simple.
Validación	Fácil en tiempo real.	Más difícil de implementar.
Ideal para	Formularios dinámicos, validaciones complejas.	Formularios simples y estáticos.

React Hook Form

(useForm)

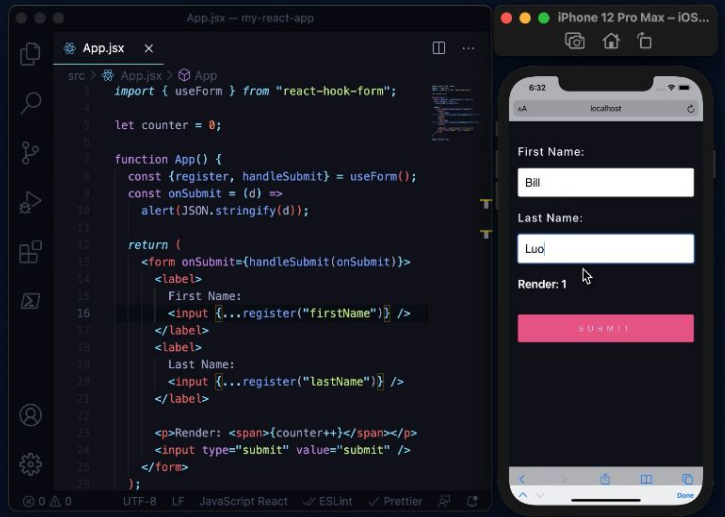
DEV.F:

ESCUELA DE HACKERS

React Hook Form

Performant, flexible and extensible forms with easy-to-use validation.

Demo | Get Started ▶



React Web | React Native

https://www.react-hook-form.com/

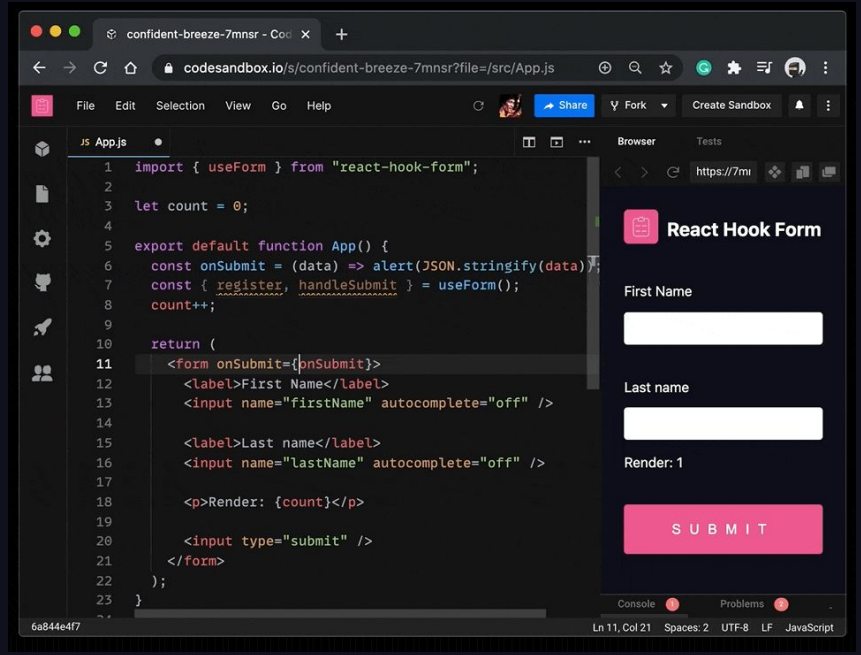
👩‍💻 React Hook Form

Librería de manejo para 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)]

Primero, necesitas instalar la librería en tu proyecto de React.

```
npm install react-hook-form
```

🧩 Implementación Básica

```tsx
import { useForm, SubmitHandler } from "react-hook-form"

type Inputs = {
  example: string
  exampleRequired: string
}

export default function App() {
  const {
    register,
    handleSubmit,
    watch,
    formState: { errors },
  } = useForm<Inputs>()
  const onSubmit: SubmitHandler<Inputs> = (data) => console.log(data)

  console.log(watch("example")) // watch input value by passing the name of it

  return (
    /* "handleSubmit" will validate your inputs before invoking "onSubmit" */
    <form onSubmit={handleSubmit(onSubmit)}>
      {/* register your input into the hook by invoking the "register" function */}
      <input defaultValue="test" {...register("example")} />

      {/* include validation with required or other standard HTML validation rules */}
      <input {...register("exampleRequired", { required: true })} />
      {/* errors will return when field validation fails  */}
      {errors.exampleRequired && <span>This field is required</span>}

      <input type="submit" />
    </form>
  )
}
```

A continuación, te muestro una implementación básica de un formulario de login usando **react-hook-form**:

DEV.F::

Yup

Validaciones

DEV.F:

Yup

```
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]

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

DEV.F::

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

Primero, necesitas instalar **Yup** junto con **@hookform/resolvers**, que es un paquete que conecta Yup con react-hook-form.

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

🧩 Implementación Básica

```
import React from 'react';
import { useForm } from 'react-hook-form';
import { yupResolver } from '@hookform/resolvers/yup';
import * as yup from 'yup';
```

Vamos a crear un formulario de ejemplo con **react-hook-form** y validación usando **Yup**.

1. Importación y Configuración

Primero, importamos **yup** y **@hookform/resolvers/yup** para integrar la validación en el formulario:

DEV.F::

🧩 Implementación Básica

```
// Esquema de validación con Yup
const schema = yup.object().shape({
  email: yup
    .string()
    .email("Email inválido")
    .required("El email es obligatorio"),
  password: yup
    .string()
    .min(6, "La contraseña debe tener al menos 6 caracteres")
    .required("La contraseña es obligatoria"),
});
```

Vamos a crear un formulario de ejemplo con **react-hook-form** y validación usando **Yup**.

2. Definición del Esquema de Validación con Yup

Definimos un esquema de validación con **Yup** para asegurarnos de que el email tenga el formato adecuado y la contraseña sea lo suficientemente segura.

DEV.F::

🧩 Implementación Básica

```jsx
function LoginForm() {
  const { register, handleSubmit, formState: { errors } } = useForm({
    resolver: yupResolver(schema), // Usamos yupResolver para integrar Yup con react-hook-form
  });

  const onSubmit = (data) => {
    console.log(data);
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <div>
        <label>Email:</label>
        <input
          type="email"
          {...register("email")}
        />
        {errors.email && <p>{errors.email.message}</p>}
      </div>

      <div>
        <label>Contraseña:</label>
        <input
          type="password"
          {...register("password")}
        />
        {errors.password && <p>{errors.password.message}</p>}
      </div>

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

Vamos a crear un formulario de ejemplo con **react-hook-form** y validación usando **Yup**.

3. Configuración del Formulario con react-hook-form

Ahora conectamos **Yup** con **react-hook-form** utilizando **yupResolver**:

DEV.F::