REACT HOOK FORM

La creación y validación de formularios es una tarea bastante simple pero que puede resultar monótona y en ocasiones difícil de gestionar, no por complejidad pero sí por tener que estar pendiente de muchos factores simultáneamente, lo cual puede llegar a hacer que nuestros componentes queden muy “sucios” teniendo que recurrir a custom hooks y reducers para liberar carga lógica y simplificar su lectura.

REACT HOOK FORM

Éste fragmento de código suele repetirse por cada formulario que tenemos para poder controlar cada campo y reaccionar a los cambios y validaciones, además en cada campo del formulario tenemos que añadir el value y el onChange para ir actualizando la información en tiempo real y por supuesto sin olvidarnos de nuestro querido e.preventDefault()

```
const [userInfo, setUserInfo] = useState({
    name: '',
    email: ''
});

const handleChange = event => {
    const { name, value } = event.target;
    setUserInfo({ ...userInfo, [name]: value });
};
```

```
<form onSubmit={e => e.preventDefault()}>
    <div>
        <label htmlFor='name'>Nombre completo</label>
        <input
            id='name'
            type='text'
            name='name'
            value={userInfo.name}
            onChange={handleChange}
        />
    </div>
```

REACT HOOK FORM - Instalación

Para todas estas gestiones existen múltiples librerías que nos ayudan y la que vamos a ver es, desde mi punto de vista, la mejor que existe actualmente, React Hooks Forms.

El primer paso, como en cualquier librería, es instalarlo en nuestro proyecto, al ser una librería que necesita nuestro proyecto para funcionar la instalaremos sin ninguna flag.

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

REACT HOOK FORM - Primeros pasos

Después nos traemos a nuestro formulario las dos funciones básicas que necesitamos para que funcione.

```
const { handleSubmit, register } = useForm();
```

La función handleSubmit la pondremos en nuestro evento submit del formulario, y como parámetro recibe el callback al que llamaremos cuando se envíe el formulario.

De forma automática se envían los datos del formulario y el evento, y ya no es necesario hacer e.preventDefault(), porque la librería lo implementa automáticamente.

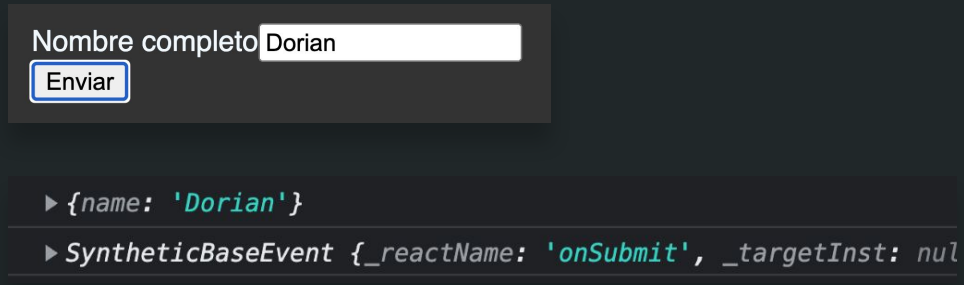
```
<form onSubmit={handleSubmit(onSubmit)}>
```

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

REACT HOOK FORM - Registrar campos

El siguiente paso es registrar nuestros campos de formulario para que pueda obtener los datos y realizar las validaciones, para hacerlo simplemente usaremos la función register pasándole como parámetro el name de nuestro campo, y automáticamente los datos llegan al campo data sin tener que hacer nada más.

```
<input
    id='name'
    type='text'
    name='name'
    {...register('name')}
/>
```



REACT HOOK FORM - Validaciones

Ahora es el turno de la validación del formulario, para ello os recomiendo la creación de tres objetos en el archivo de constantes.

```
const messages = {
    name: 'El formato introducido no es el correcto',
    requireName: 'El nombre es obligatorio',
    email: 'Debes introducir una dirección correcta',
    requireEmail: 'El email es obligatorio'
};

const patterns = {
    name: /^[A-Za-z]*$/,
    email: /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/
};

export const FORM_VALIDATIONS = {
    name: {
        require: messages.requireName,
        pattern: patterns.name,
        message: messages.name
    },
    email: {
        require: messages.requireEmail,
        pattern: patterns.email,
        message: messages.email
    }
};
```

REACT HOOK FORM - Uso de validaciones

Para usar la validación en nuestros campos del formulario sólo tenemos que añadir un objeto a la función register.

```
<input
    id='name'
    type='text'
    name='name'
    {...register('name', {
        required: FORM_VALIDATIONS['name'].require,
        pattern: {
            value: FORM_VALIDATIONS['name'].pattern,
            message: FORM_VALIDATIONS['name'].message
        }
    })}
/>
```

REACT HOOK FORM - Manejo de errores

Para el manejo de errores añadiremos formState: {errors} a la extracción de valores del hook useForm()

```
const {register, handleSubmit, formState: { errors }} = useForm();
```