ALCYON
IT Servicios

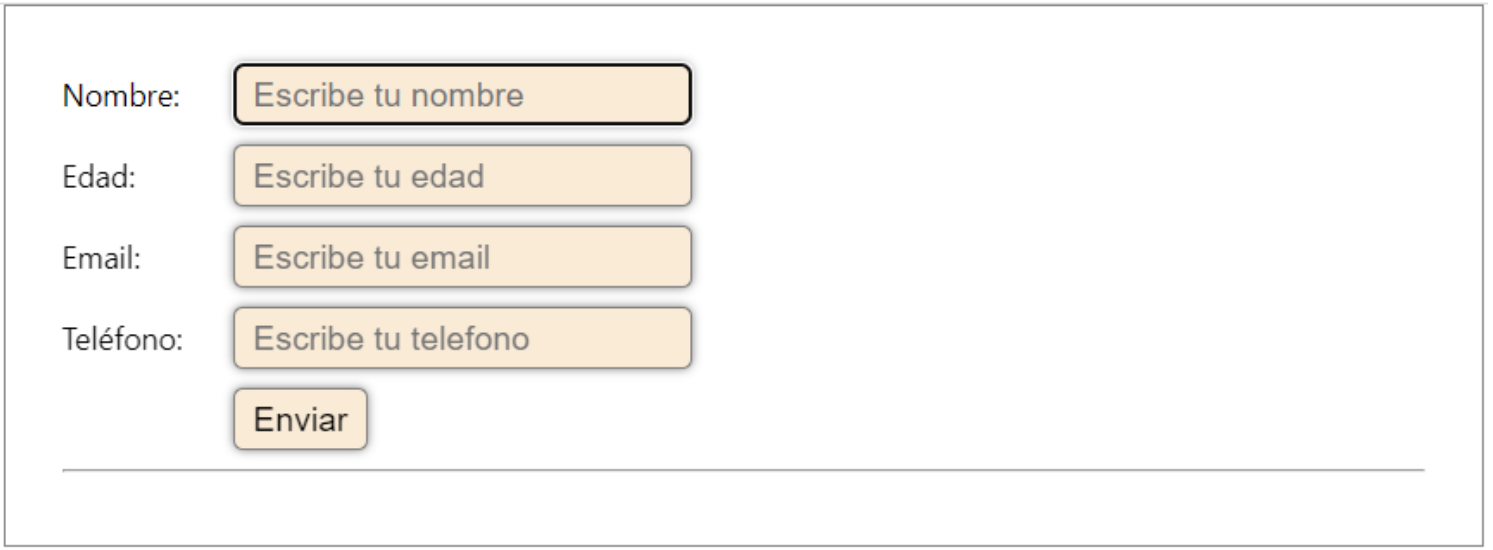
PARTE IX: Formularios y validación

ALCYON
IT Servicios

INTRODUCCIÓN

Hasta ahora hemos trabajado de forma muy superficial con controles input de formulario pero no hemos confeccionado un formulario completo.

Vamos a ver un ejemplo de como trabajar con formularios de forma que podamos recoger datos del mismo (sin utilizar referencias o el objeto event.target) y como validar estos datos sin utilizar estructuras JavaScript



ALCYON
IT Servicios

INTRODUCCIÓN

Creación del proyecto

Crearemos un nuevo proyecto desde el cmd o el terminal de VSC:

create-react-app formularios

Instalar la librería react para trabajar con formularios

Necesitamos instalar la libería REACT para poder utilizar los componentes y hooks de formularios:

npm install react-hook-form

Preparar el entorno

Vamos a editar el fichero **src/App.css** y borraremos el contenido

Editamos el fichero **src/App.js** y vaciamos el contenido del return de la función App()

ALCYON
IT Servicios

CREACION DE COMPONENTES

No vamos a necesitar más componentes que el que viene por defecto al crear el proyecto: **App.js** dentro del cual crearemos el formulario que utilizaremos de ejemplo

ALCYON
IT Servicios

CREAR FORMULARIO

Como primer paso, crearemos el formulario

```
<div className="App">
    <form>
      <div className='pregunta'>
        <label htmlFor='nombre'>Nombre: </label>
        <input id='nombre' placeholder='Escribe tu nombre' autoFocus/>
      </div>
      <div className='pregunta'>
        <label htmlFor='edad'>Edad: </label>
        <input type='number' id='edad' placeholder='Escribe tu edad'/>
      </div>
      <div className='pregunta'>
        <label htmlFor='email'>Email: </label>
        <input type='email' id='email' placeholder='Escribe tu email'/>
      </div>
      <div className='pregunta'>
        <label htmlFor='telefono'>Teléfono: </label>
        <input tpye='tel' id='telefono' placeholder='Escribe tu telefono'/>
      </div>
      <div className='pregunta'>
        <label></label>
        <input type='submit'/>
      </div>
    </form>
</div>
```

ALCYON
IT Servicios

CREAR FORMULARIO

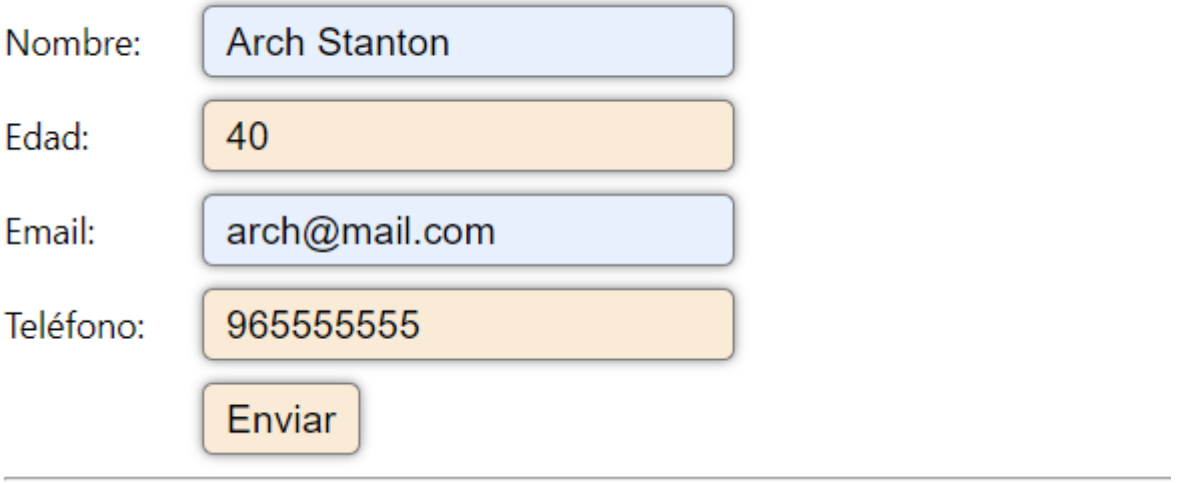
Hemos creado un formulario con los controles más utilizados para poder ver, posteriormente, varios tipos de validación

NOTA: El atributo for para la etiqueta <label> en react es htmlFor y la propiedad autofocus en react es autoFocus

CREAR FORMULARIO

Para que el formulario aparezca con un aspecto más o menos decente vamos a añadir algo de css en **App.css**

```
.pregunta {margin-bottom: 10px;}
label {display:inline-block;width:90px;}
input {padding: 5px 10px; background-color: antiquewhite;font-size: 1.1em; border: 1px solid grey; border-radius: 5px;box-shadow: 0px 0px 5px grey;}
```



ALCYON
IT Servicios

INCORPORAR HOOKS

Vamos a crear, en una constante, todos los hooks que vamos a necesitar para poder trabajar con formularios.

De momento solo incorporaremos dos y, según los vayamos necesitando, iremos incorporando el resto.

Dentro de la función del componente **App.js**:

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

register:

Array de objetos que contendrá todos los valores que informemos en el formulario

handleSubmit

Método donde indicaremos que función se encargará de recoger los datos del formulario

ALCYON
IT Servicios

SUBMIT DEL FORMULARIO

Indicamos en la propia etiqueta <form> el evento onSubmit en donde utilizaremos el método handleSubmit para indicar la función que utilizaremos para recoger los datos del formulario

<form **onSubmit={handleSubmit(recogerDatos)}**>

Para evitar continuar con errores, vamos a crear la función recogerDatos

```
const recogerDatos = (datos) => { console.table(datos) }
```

NOTA: Esta función recibirá como parámetro todos los datos del formulario dentro de un objeto y, para poder visualizarlos, los mostraremos en la consola en formato tabla

ALCYON
IT Servicios

RECOGER DATOS DE LOS INPUT

Ahora vamos a modificar los inputs del formulario para utilizar el elemento register que será quien se encargue de recoger los datos que informa el usuario en cada input y enviarlos a la función recogerDatos que hemos definido antes:

```
<input id='nombre' placeholder='Escribe tu nombre' autoFocus
{ ...register('nombre')}/>

<input type='number' id='edad' placeholder='Escribe tu edad'
{ ...register('edad')}/>

<input type='email' id='email' placeholder='Escribe tu email'
{ ...register('email')}/>

<input tpye='tel' id='telefono' placeholder='Escribe tu teléfono'
{ ...register('telefono')}/>
```

NOTA: Utilizamos ...register para que no se pierdan los datos que hemos informado en los input previos

ALCYON
IT Servicios

RECOGER DATOS DE LOS INPUT

Si informamos datos en el formulario y pulsamos el botón del mismo, podremos ver en la consola como se muestran todos los datos (en formato tabla ya que hemos utilizado un console.table(datos) en la función recogerDatos

(index)	Value
nombre	'Arch Stanton'
edad	'15'
email	'arch@mail.com'
telefono	'915555555'

ALCYON
IT Servicios

VALIDAR DATOS

Vamos a ver ahora una de las funcionalidades más potentes de la librería de react para trabajar con formularios: las validaciones de datos.

Para validar datos vamos a ver el ejemplo en el primer input del nombre y, posteriormente, añadiremos las validaciones en el resto de inputs

Para validar datos necesitamos incorporar el hook formState dentro de la constante que hemos definido en la función del componente:

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

ALCYON
IT Servicios

VALIDAR DATOS

Vamos a ver como se utiliza en el input nombre:

```
<input id='nombre' placeholder='Escribe tu nombre' autoFocus
{...register('nombre',
      {
         required: true,
         maxLength: 20
      }
)}/>
```

Para el control *nombre* vamos a validar que el dato sea obligatorio (atributo required) y que tenga una longitud máxima de 20 caracteres (atributo maxLength)

ALCYON
IT Servicios

VALIDAR DATOS

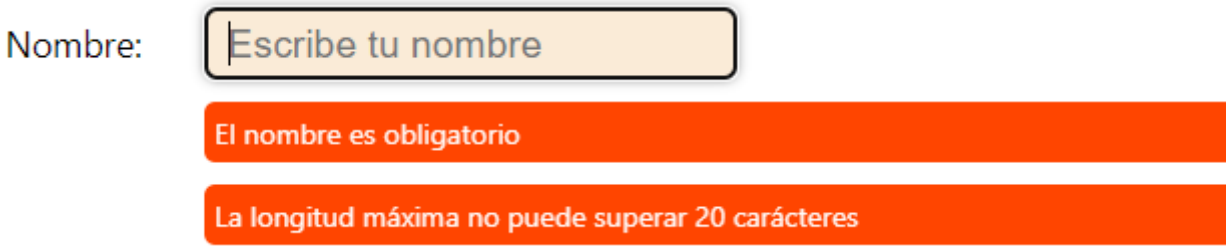
Y ahora vamos a ver como mostrar los mensajes de error asociados a las dos validaciones anteriores:

- Creamos una caja para cada una de las validaciones y las situamos debajo de la caja del control de input:

```
<div className='pregunta'> … </div>
<div className='errores'>El nombre es obligatorio</div>
<div className='errores'>La longitud máxima no puede superar 20 carácteres</div>
```

- Y añadiremos un poco de css para estas cajas

```
.errores {background-color: orangered; color: white; margin-bottom: 10px; margin-left: 90px;padding: 5px; border-radius: 5px;font-size: 0.8em;}
```



ALCYON
IT Servicios

VALIDAR DATOS

- Necesitamos un sistema que nos permita mostrar las cajas solo cuando se produzca alguno de los errores. Para ello utilizaremos el objeto errors que hemos incorporado antes en el hook formState

```
{errors.nombre?.type === 'required' &&
    <div className='errores'>El nombre es obligatorio</div>
}

{errors.nombre?.type === 'maxLength' &&
    <div className='errores'>La longitud máxima no puede superar 20 carácteres</div>
}
```

NOTA 1: Para el objeto errors preguntamos por el atributo que corresponde al nombre del input que hemos informado en register y preguntamos por el tipo de error que queremos detectar

NOTA 2: Puesto que si el contenido del input es correcto no existirán errores tenemos que comprobar cada error utilizando el operador **?** después del nombre del atributo correspondiente al nombre del input

NOTA 3: Recordad que el operador && se utiliza para ejecutar una acción si se cumple la condición situada a la izquierda del mismo

ALCYON
IT Servicios

VALIDAR DATOS

- Si probamos a dejar sin informar el nombre veremos

- Y si probamos a introducir un nombre de más de 20 caracteres

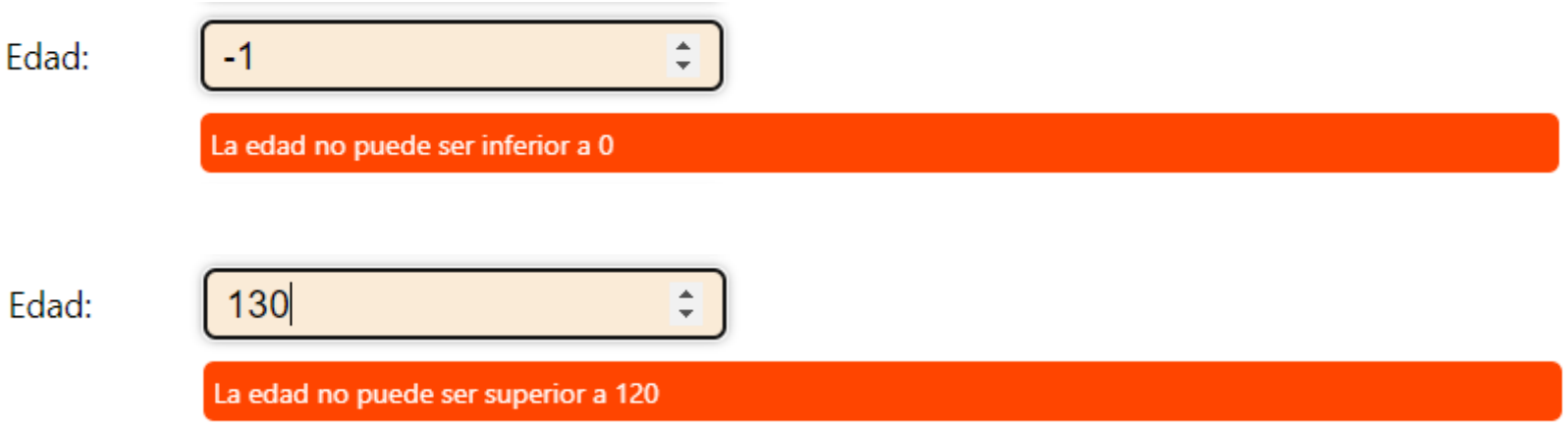
ALCYON
IT Servicios

VALIDAR DATOS

Vamos a utilizar la misma técnica de validación para el resto de inputs

Edad: como mínimo 0 y máximo 100

{...register('edad', { min: 0, max: 120 })}

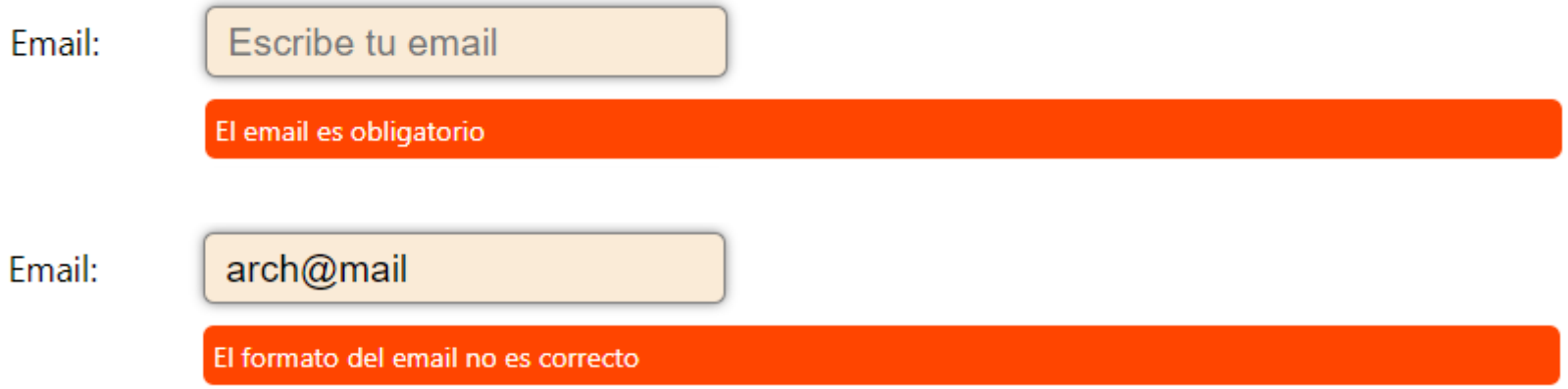


ALCYON
IT Servicios

VALIDAR DATOS

Email: obligatorio y patrón de validación del tipo *nombre@proveedor.dominio*

{...register('email', {required: true, pattern: /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i})}



ALCYON
IT Servicios

OBSERVAR DATOS

Otra de las funcionalidades de React en el tratamiento de formularios es la posibilidad de observar los datos, es decir, que éstos aparezcan en una sección diferente del componente JSC según los vayamos escribiendo.

Por ejemplo, según vayamos introduciendo los datos en los input, iremos viendo como se construye un mensaje debajo del formulario con la información que vayamos escribiendo:

ALCYON
IT Servicios

OBSERVAR DATOS

Primero tenemos que incorporar un nuevo hook en la constante que hemos creado dentro de la función del componente:

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

Incorporaremos una nueva sección justo debajo del formulario y que utilizaremos para mostrar el mensaje resumen de los datos que vayamos introduciendo

```
<form> … </form>
<div className='resumen'></div>
```

Añadiremos también algo de css en **App.css**

```
.resumen p {background-color: lightgreen; padding: 5px; border-radius: 5px;
width: fit-content;}
```

ALCYON
IT Servicios

OBSERVAR DATOS

Dentro de la sección anterior utilizaremos el método watch() de la siguiente forma:

```
<p>
    Me llamo <b>{watch('nombre')}</b>
    <span> Tengo {watch('edad')} años.</span>
    <span> Mi email es {watch('email')}</span>
    <span> y mi teléfono es {watch('telefono')}.</span>
</p>
```

Si probamos vemos que no nos acaba de convencer el resultado:

Me llamo . Tengo años. Mi email es y mi teléfono es .

Ya que los literales fijos siempre aparecen incluso aunque no informemos todavía ningún input del formulario

ALCYON
IT Servicios

OBSERVAR DATOS

Podemos mejorarlo para que el resumen solo aparezca cuando informamos el primer input obligatorio (el nombre)

```
{watch('nombre') &&
  <p>
      Me llamo <b>{watch('nombre')}</b>
      <span> Tengo {watch('edad')} años.</span>
      <span> Mi email es {watch('email')}</span>
      <span> y mi teléfono es {watch('telefono')}.</span>
  </p>
}
```

Mientas no informemos el nombre no aparecerá ningún texto, pero en cuanto informemos algo aparecerán de nuevo todos los literales, incluso aunque todavía no los hayamos informado.

Me llamo **Arch**. Tengo años. Mi email es y mi teléfono es .

ALCYON
IT Servicios

OBSERVAR DATOS

La solución pasa por comprobar, uno a uno, si se ha informado algún dato en cada uno de los input:

```
{watch('nombre') &&
  <p>
      Me llamo <b>{watch('nombre')}</b>
      {watch('edad') && <span> Tengo {watch('edad')} años.</span>}
      {watch('email') && <span> Mi email es {watch('email')}</span>}
      {watch('telefono') && <span> y mi teléfono es {watch('telefono')}.</span>}
  </p>
}
```

Me llamo **Arch Stanton**.

Me llamo **Arch Stanton**. Tengo 67 años.

Me llamo **Arch Stanton**. Tengo 67 años. Mi email es arch@mail

Me llamo **Arch Stanton**. Tengo 67 años. Mi email es arch@mail y mi teléfono es 935555555.

ALCYON
IT Servicios

RESETEAR FORMULARIO

Vamos a ver dos funcionalidades habituales en formularios cuando se gestionan con JS nativo pero que, con react, hay que gestionarlas utilizando las propias funciones del framework

- Situar el cursos en un input determinado
- Resetear el formulario cuando se recogen los datos sin errores.

Como es habitual, cualquier funcionalidad nueva hay que incluirla en la constante que hemos creado dentro dela función del componente:

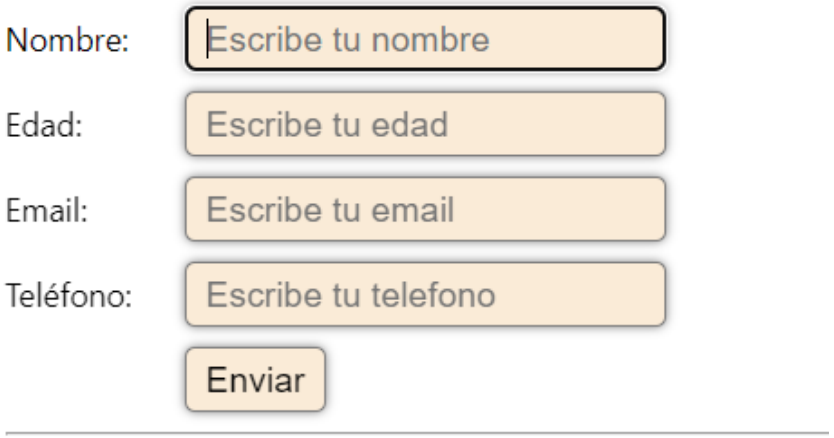
```
const {register, handleSubmit, formState: {errors}, watch, setFocus, reset} = useForm()
```

- Con setFocus podremos situar el cursos en cualquier input del formulario
- Con reset podremos limpiar el formulario de cualquier dato introducido

RESETEAR FORMULARIO

Incorporaremos ambas funcionalidades dentro de la función recogerDatos para simular que, una vez procesado el formulario, volvemos a dejarlo en su estado inicial (sin datos) y con el cursor posicionado en el primer input obligatorio (el nombre)

```
const recogerDatos = (datos) => {
    console.table(datos)
    setFocus('nombre')
    reset()
}
```



ALCYON
IT Servicios

VALORES POR DEFECTO

Imaginemos que tenemos un formulario donde queremos que, cuando se cargue el componente, aparezcan unos valores por defecto procedentes, por ejemplo, de una consulta previa a una base de datos

Para ello necesitamos un objeto con los datos a mostrar y, para ello, podemos utilizar el hook useState

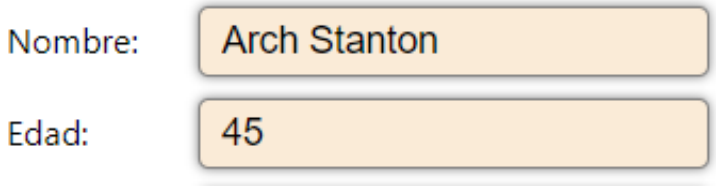
```
const [persona, setPersona] = useState(
    {
        nombre: 'Arch Stanton',
        edad: 45,
        email: 'arch@mail.com',
        telefono: '66677889'
    })
```

ALCYON
IT Servicios

VALORES POR DEFECTO

Y, para mostrar los valores en el formulario podríamos utilizar el atributo value de HTML (veamos el ejemplo con nombre y edad)

```
<div className='pregunta'>
  <label htmlFor='nombre'>Nombre: </label>
  <input id='nombre' placeholder='Escribe tu nombre' autoFocus
    value={persona.nombre} />
</div>
<div className='pregunta'>
  <label htmlFor='edad'>Edad: </label>
  <input type='number' id='edad' placeholder='Escribe tu edad'
    value={persona.edad}/>
</div>
```



ALCYON
IT Servicios

VALORES POR DEFECTO

Pero ahora vemos que los dos inputs han quedado bloqueados para poder modificar los datos visualizados

Para poder habilitarlos no podemos utilizar el atributo value sino que tenemos que utilizar el atributo específico de react defaultValue

```
<div className='pregunta'>
   <label htmlFor='nombre'>Nombre: </label>
   <input id='nombre' placeholder='Escribe tu nombre' autoFocus
      defaultValue={persona.nombre} />
 </div>
 <div className='pregunta'>
   <label htmlFor='edad'>Edad: </label>
   <input type='number' id='edad' placeholder='Escribe tu edad'
      defaultValue={persona.edad}/>
 </div>
```

ALCYON
IT Servicios

VALORES POR DEFECTO EN MODIFICACION

Pero ¿que pasaría si se tratara de un formulario donde, además de mostrar los datos del usuario, quisiéramos modificarlos, añadiendo un botón de modificación?

Veríamos que, si no modificamos todos los datos aparecerá el error de validación en aquellos input que no hemos tocado.

Para evitarlo vamos a informar los valores por defecto del formulario utilizando el método setValue en el hook useForm:

```
const {register, handleSubmit, formState: {errors}, watch, setFocus, reset, setValue} = useForm()
```

ALCYON
IT Servicios

VALORES POR DEFECTO EN MODIFICACION

Y, posteriormente, informamos los datos del formulario utilizando este método:

```
setValue("nombre", persona.nombre);
setValue("edad", persona.edad);
setValue("email", persona.email);
setValue("telefono", persona.telefono)
```

ALCYON
IT Servicios

INPUTS DE TIPO FILE

Para recuperar archivos utilizando un input de tipo file realizaríamos los siguientes pasos:

- Creamos en la plantilla jsx el input:

```
<input type="file" {...register('imagen')} accept='image/*'/>
```

- En la función donde recogemos los datos del formulario:

```
const recogerDatos = (datos) => {
    const nombreImagen = datos.imagen[0].name
    const imagen = datos.imagen[0]
}
```

Aviso Legal

David Alcolea
david-alcolea@alcyon-it.com
www.alcyon-it.com