

YUP

Se usa la siguiente sintaxis para usarlo

```
import * as Yup from 'yup'
```

Despues hay que crear un schema. Es básicamente un objeto que tiene toda la forma, con todos los campos que vas a tener, y que forma deben tener esos campos

- shape va a ser la forma que tienen los datos cuando vaya a crear un nuevo cliente
- Va a recibir los datos de initialValue(del formulario)

```
const nuevoClienteSchema= Yup.object().shape({  
  nombre: Yup.string().required('El nombre del cliente es requerido'),  
  empresa: '',  
  email:'',  
  telefono:'',  
  notas:''  
})
```

A Formik tambien le agrego el validationSchema con los valoresde nuevoClienteSchema

```
<Formik  
  initialValues ={{  
    nombre: '',  
    empresa: '',  
    email:'',  
    telefono:'',  
    notas:''  
  }}  
  onSubmit={values=>{  
    handleSubmit(values)  
  }}  
  validationSchema={nuevoClienteSchema}  
>
```

Para ver la data del formulario que esta envuelto en un arrow function: - Pongo data de argumento - En lugar del return implicito con parentesis, lo mantengo pero envuelvo todo el form entre llaves. - Pongo un console.log de data y agrego el return

```
{{data)}=>{ console.log(data)  
  return (  
    <Form className="mt-10">
```

```
<div className="mb-4">
  <label
    className='text-gray-800'
    htmlFor='nombre'>Nombre:</label>
  <Field
    id="nombre"
    type="text"
    className="mt-2 block w-full p-3 bg-gray-100"
    placeholder="Nombre del cliente"
    name="nombre"/>
</div>
<div className="mb-4">
  <label
    className='text-gray-800'
    htmlFor='empresa'>Empresa Cliente:</label>
  <Field
    id="empresa"
    type="text"
    className="mt-2 block w-full p-3 bg-gray-100"
    placeholder="Nombre de la empresa"
    name="empresa"/>
</div>
<div className="mb-4">
  <label
    className='text-gray-800'
    htmlFor='email'>Email</label>
  <Field
    id="email"
    type="email"
    className="mt-2 block w-full p-3 bg-gray-100"
    placeholder="Email del Cliente"
    name="email"/>
</div>
<div className="mb-4">
  <label
    className='text-gray-800'
    htmlFor='telefono'>Teléfono</label>
  <Field
    id="telefono"
    type="tel"
    className="mt-2 block w-full p-3 bg-gray-100"
    placeholder="Teléfono del Cliente"
    name="telefono"/>
</div>
<div className="mb-4">
  <label
    className='text-gray-800'
    htmlFor='notas'>Notas</label>
  <Field
    as="textarea"
    id="notas"
    type="text"
    className="mt-2 block w-full p-3 bg-gray-100 h-20"
    placeholder="Notas"
```

```

        name="notas"/>
      </div>
      <input
        type="submit"
        value="Agregar Cliente"
        className="mt-5 w-full bg-blue-800 p-3 text-white uppercase font-bold
text-lg"/>
    </Form>
  )}}

```

- Dentro del objeto en consola puedo ver que en errors, si no puse el nombre en el formulario y cliqué en el submit, aparece un objeto con la frase de campo requerido

De esta forma puedo extraer el error y mostrarlo en pantalla

- Puedo usar desestructuración en la data abriendo llaves y colocando errors
- Ahora puedo usarlo para mostrar el mensaje. Le pongo una segunda condición, con touched(que también saco de la desestructuración) tengo validación en tiempo real. -una vez clico en nombre, si salgo sin poner nada el touched actúa

Si estos dos existen, regresame el error, si no nada

```

{errors.nombre && touched.nombre ? (
  <div>
    {errors.nombre}
  </div>
): null}

```

- Por supuesto le puedo añadir estilos al div
- Pero también puedo crear un componente de esta alerta y que tome children para reutilizarlo

```

import React from 'react'

const Alerta = ({children}) => {
  return (
    <div className="text-center my-4 bg-red-800 text-white
font-bold p-3 uppercase">
      {children}
    </div>
  )
}

export default Alerta

```

- Puedo escribir min() en la validación para especificar un minimo de caracteres
- Introduzco n como minimo seguido de una coma y el mensaje entre comillas

```
nombre: Yup.string().min(3, 'El nombre es muy corto')
  .required('El nombre del cliente es requerido'),
```

- Puedo hacer lo mismo pero a la inversa con max
- El email tiene su propio método. Si no escribo @*.com no será válido
- Cómo teléfono no lo voy a hacer obligatorio, escribo mi error con `typeError` y que solo sean números enteros

```
const nuevoClienteSchema= Yup.object().shape({

  nombre: Yup.string().min(3, 'El nombre es muy corto')

    .required('El nombre del cliente es requerido'),

  empresa: Yup.string().required('El nombre de la empresa es obligatorio'),

  email: Yup.string().email('Email no válido').required('El email es obligatorio'),

  telefono: Yup.number().integer('Número no válido').positive('Número no válido').typeError('No es válido'),

  notas: ''
})
```