

Git 1r_MódULO

Primeros Comandos

- Primer comando

```
> git help
```

Para info de un comando específico

```
> git --help comando
```

- De gran ayuda 😊
- Para listar info de usuario, también se puede modificar

```
> git config --global -e
```

- Esc :wq! para salir
- Para inicializar el repositorio y crearlo en el espacio local

```
> git init
```

- Se crea .git, una carpeta oculta que no debe tocarse
- Para visualizar dónde estoy y que archivos sigue o no sigue Git

```
> git status
```

- Para agregar individualmente

```
> git add nombre_del_archivo
```

- Para agregarlos todos

```
> git add .
```

- Para remover algun archivo

```
> git reset Archivo_a_borrar
```

- Ahora están listos en el stage para hacer el commit.
- Con -m agrego el mensaje de info que acompaña a la snapshot

```
> git commit -m "mi primer commit"
```

- Para ver los commits

```
> git log
```

- Para volver al estado de mi último commit de mi proyecto

```
> git checkout -- .
```

MUY ÚTIL

- Para saber en que rama estoy trabajando

```
> git branch
```

Es buena práctica que el master sólo tenga lo que va a producción, la versión final, o cambios revisados

- Es recomendable trabajar en ramas.
- Para renombrar el master a main

```
> git branch -m master main
```

- Para que lo haga por defecto

```
> git config --global init.defaultBranch main
```

- Para saltarse los pasos de actualizar con add puedo usar -a de all y -m de mensaje

```
> git commit -am "Todo agregado"
```

Sólo funciona con archivos que ya están en seguimiento

- Para coger todos los archivo html puedo usar un comodin

```
> git add *.html
```

- Tengo que estar en el mismo directorio

```
> git add src/js/*.js
```

- Si la carpeta no contiene nada git no lo reconoce en el sistema

Puedo ponerle el archivo .gitkeep dentro para que lo haga, pesa 0Bytes

```
> git add src/.gitkeep  
> git commit -m "carpeta añadida"
```

Diferentes formas de añadir al stage

- Para añadir todo un directorio al stage

```
> git add css/
```

- Para añadir todos los archivos

```
> git add .
```

Hay una manera gráfica de añadir, hacer cambios y más cosas en vscode, dándole al icono de Git de la esquina inferior izquierda. Bonus

Alias

Si quiero un resumen de la info

```
> git status --short
```

- Pero es mucho código para escribir. Yo puedo crear mis propios alias

```
> git config --global alias.s status  
> git s
```

- Para entrar en modo edición del global

```
> git config --global -e
```

- Tecla a para modificar---> añadido a s = status --short
- Esc :wq! para salir

Para hacerlo directamente lo debo entrecomillar: `git config --global alias.s "status --short"`

- Para leer la forma corta del hash y un formato más amigable

```
> git log --oneline
```

- Puedo crearme los alias que quiera concatenando los parámetros
> git config --global alias.lg "log --oneline --decorate --all --graph"

Instalando Tailwind

```
> npm i autoprefixer postcss tailwindcss  
> npx tailwindcss init -p
```

- En tailwind.config.js le digo que el archivo purge: ["index.html", ./src/**/*.*.jsx]
- Uso los comodines para buscar todos los archivos acabados en jsx
- En index.css importo
 - @tailwind base;
 - @tailwind components;
 - @tailwind utilities;

En el body voy a agregar un par de clases - minimo el height tome toda la altura - el background sea un gris

```
<body class="min-h-screen bg-gray-100">
```

Instalando React Router 6

- Con el Routing puedo tener varias url por cada componente/página que quiero mostrar
- También puedo restringir el acceso
- React Router es la más popular
- Gatsby y Next incorporan su propio router

```
> npm -i react-router-dom
```

Lo importo en App.jsx

```
import {BrowserRouter, Routes, Route} from 'react-router-dom'
```

Definiendo Rutas

Creo la carpeta Layouts con dos archivos: IniciarSesión.jsx Layout.jsx Creo la carpeta llamada páginas con varios componentes: - Inicio.jsx - EditarCliente.jsx - NuevoCliente.jsx - LoginForm

- App es el componente principal, ahí instalo Browser Router
- Dentro de Routes estarán todos mis rutas, cada una en un Route
- En React-Router6 se pueden usar nested routes
- Tengo un >Route de apertura y cierre que significa un grupo de Rutas y dentro un Route solo de cierre, que significa una sola ruta
- Le digo que cuando visite la página principal en la ruta "/" muestre el elemento IniciarSesion
- Creo otro grupo de rutas con /clientes para que muestre el Layout Lo que coloque dentro de ellos va a ser rutas anidadas
- Coloco inicio sin path dentro de la ruta /clientes
- Donde importe Outlet de 'react-router-dom' lo mostrará
- Importo {Outlet} en el componente Layouts y lo coloco debajo del h1
- Cómo puede haber varias rutas anidadas, pero hay que especificar cual es la del index

```

return (
  <BrowserRouter>
    <Routes>
      <Route path="/" element={<IniciarSesion />}>
      </Route>

      <Route path="/clientes" element={<Layout />}>
        <Route index element={<Inicio />} />
      </Route>
    </Routes>
  </BrowserRouter>
)

```

```

import {Outlet} from 'react-router-dom'

const Layouts = () => {
  return (
    <div>
      <h1>Layouts</h1>
      <Outlet />
    </div>
  )
}

export default Layouts

```

- Ahora si. Siempre que visite /clientes va a insertar este Layout y dentro va a ir agregando cada componente por separado. Separar un header, un sidebar he ir agrupándolo en este Layout.
- En todas las páginas voy a tener ese layout, que puede ser un header, un menú de navegación, lo que sea. Abajo un footer y en medio todo lo que se vaya agregando a este Outlet
- Hago lo mismo dentro de /

```

return (
  <BrowserRouter>
    <Routes>
      <Route path="/" element={<IniciarSesion />}>
      <Route index element={<LoginForm />} />
      </Route>

      <Route path="/clientes" element={<Layout />}>
        <Route index element={<Inicio />} />
      </Route>
    </Routes>
  </BrowserRouter>
)

```

```
</BrowserRouter>
```

Es importante importar el Outlet en el componente y colocarlo

```
import {Outlet} from 'react-router-dom'
```


Agrupando Rutas

- Importo NuevoCliente y EditarCliente en App.jsx
- Si coloco el Route dentro de /clientes, solo tengo que colocar nuevo en el path de NuevoCliente
- Si quiero que responda a un id le añado un placeholder con : basado en el id

```
<Route path="/clientes" element={<Layout />}>
  <Route index element={<Inicio />} />
  <Route path="nuevo" element={<NuevoCliente />}/>
  <Route path="editar/:id" element={<EditarCliente />}/>
</Route>
```

Borro el primer grupo de rutas, dejo solo /clientes Borro IniciarSesion

Layout

Le coloco clases de Tailwind al div de Layout

- Le digo que en tamaño mediano aplico flexbox
- Y que el tamaño mínimo de pantalla es toda la altura
- En div hijo, que la barra lateral tiene un width de 1/4
- En otro div 3/4

```
import {Outlet} from 'react-router-dom'

const Layouts = () => {
  return (
    <div className="md:flex md:min-h-screen">
      <div className="md:w-1/4">
        1
      </div>

      <div className="md:w-3/4">
        <Outlet />
      </div>
    </div>
  )
}

export default Layouts
```

De esta manera tengo un grid

- El 1 va a ser una barra lateral que se va a compartir en todos los componentes
- En la parte más grande de la pantalla el contenido

-
- Un padding x de 5 y un padding y de 10
 - Dentro un h2 con la clase 4xl para que sea más grande el texto, font-black para que sea en negritas
 - También un text-center para centrarlo y un text-white para darle el color blanco
 - Le añadido un nav con dos enlaces
 - Clientes
 - Nuevo Cliente
 - Le añadido las rutas

```
<div className="md:flex md:min-h-screen">
  <div className="md:w-1/4 bg-blue-900 px-5 py-10">
    <h2 className="text-4xl font-black text-center text-white">CRM-
Clientes</h2>

    <nav className="mt-10">
      <a href="/clientes">Clientes</a>
      <a href="/clientes/nuevo">Nuevo Cliente</a>
    </nav>

  </div>
```

Le añadido las clases a los enlaces. La misma para los dos

- texto color blanco, grande 2xl, display:block, un margin-top de 2 y en el hover un blue

```
className="text-white text-2xl block mt-2 hover:text-blue-300"
```

Link

- Si uso anchor tags para los links, puedo ver una recarga en el navegador al clicar sobre ellos
- Con react-router-dom se usa Link y NavLink. Debes importarlos de react-router-dom
- Tampoco se usa href, se usa to="/ruta"

```
<nav className="mt-10">
  <Link to="/clientes"
    className="text-white text-2xl block mt-2 hover:text-blue-300"
  >Clientes</Link>
  <Link to="/clientes/nuevo"
    className="text-white text-2xl block mt-2 hover:text-blue-300"
  >Nuevo Cliente</Link>
</nav>
```

useLocation Hook

- Lo importo de react-router-dom en el componente Layout.jsx.
- Le hago un console.log para ver que trae

```
const location = useLocation()

console.log(location)
```

- Tiene un hash, que sirve para obtener parámetros de la url, un key único, trae el pathname dónde está ubicado el hook, y el Search que se llena con un queryString de la barra de navegación, un parámetro de búsqueda
- Guardo el pathName en una variable
- Para que se quede fijado cuando estoy en Clientes lo meto en un template string y
 - evalúo la condición.
 - Lo meto entre strings porque si no creará que es una variable
 - Cómo es código de javascript va entre llaves

```
className={` ${urlActual === '/clientes'? 'text-blue-300': 'text-white'} text-2xl
block mt-2 hover:text-blue-300` }
```

Copio y pego en el otro link y la comparación será con /clientes/nuevo

Creando las Páginas Necesarias

- Me sitúo en NuevoCliente
- Retorno un fragment
- Le añado negrita, tamaño grande, color azul 400
- También un párrafo con padding-top de 10

```
<>
  <h1 className= "font-black text-4xl text-blue-400">Nuevo Cliente</h1>
  <p className="mt-3">Llena los siguientes campos para registrar un
cliente</p>
</>
```

- Le añado un padding al Outlet de 10

```
<div className="md:w-3/4 p-10">
  <Outlet />

</div>
```

- Creo la carpeta components y el componente formulario con un h1
- Lo importo en NuevoCliente y lo visualizo
- Le agrego unas clases: texto gris, bold, un poco más grande, en mayúsculas y centrado
- Lo meto en un div con las clases color de fondo blanco
 - margin-top: 10
 - padding x de 5, 10 de y
 - esquinas redondeadas medium y sombreado medium
 - Y un md de mediaquery, un width de 3/4
 - Lo centro con un margin x auto

```
<div className="bg-white mt-10 px-5 py-10 rounded-md shadow-md md:w-3/4 mx-
auto">
  <h1 className="text-gray-600 font-bold text-3xl uppercase text-center">
Formulario
  </h1>

</div>
```

- Ahora voy a incorporar una librería para manejar los formularios

Formik

- Es aconsejable usar una librería cuando hay muchos formularios, muchos campos o son complejos.
- Algunas librerías cuentan con validaciones muy robustas, otras se integran bien con laguna dependencia de validación
- Formik usa Yup y es la que voy a usar
- React Hook form es otra librería
- instalo con

```
> npm i formik yup
```

Importo {Formik, Form, Field} from 'formik' en Formulario.jsx

-Uso Formik y Form como componentes

- Coloco un label con nombre y a continuación Field todo dentro de un div
- A este field le puedo agregar atributos - Type="text" o "password", value etc
- Tiene los mismos atributos que un input
- Le añado unas clases de tailwind: -margin-top de 2, display: block width todo el ancho, padding de 3, fondo gris
- Le añado un htmlFor al label y le paso el value al field, así cuando de clic al nombre, se activa el input
- Le añado también un margin top de 10 al Form y un margin bottom al div entero

```
<Formik>
  <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"/>
    </div>
  </Form>
</Formik>
```

- Copio el código para el campo de la empresa, el mail, el telf cambiando los respectivos types htmlFor y id's

```
<Formik>
  <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"/>
    </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"/>
    </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"/>
    </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"/>
    </div>
    <div className="mb-4">
      <label
        className='text-gray-800'
        htmlFor='notas'>Notas</label>
      <Field
        id="notas"
        type="text"
        className="mt-2 block w-full p-3 bg-gray-100"
        placeholder="Notas"/>
    </div>
  </Form>
</Formik>
```

```
</Form>
```

```
</Formik>
```

- Field por defecto te coloca un input pero yo le puedo cambiar el tipo de elemento con as
- Lo hago más grande con una altura de 40

```
<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-30" //altura
    placeholder="Notas"/>
</div>
```

- Creo el boton de submit - Margin top de 5, todo el ancho, color azul, texto blanco, mayúsculas, bold, texto más grande

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

- Añado un mediaquery al div del Outlet en Layouts y le digo que con un tamaño mediano o más grande la altura va a ser lo que mida la pantalla
- Va a tomar toda la altura disponible, y puedo ver que si bajo con el scroll el azul de la columna se corta. Por eso el overflow-scroll del lado del outlet
- De esta forma tengo el scroll en la parte derecha y en la izquierda no

```
<div className="md:w-3/4 p-10 md:h-screen overflow-scroll">
  <Outlet />

</div>
```

Vite App

localhost:3000/clientes/nuevo

CRM- Clientes

Cientes

Nuevo Cliente

Nuevo Cliente

Llena los siguientes campos para registrar un cliente

AGREGAR CLIENTE

Nombre:

Empresa Cliente:

Email

Teléfono

23:20
02/05/2022

Valores iniciales para el formulario

- Formik en su API da el objeto de initialValues
- Los inicializo como un string vacío

```
<Formik
  initialValues={{
    nombre: '',
    empresa: '',
    email: '',
    telefono: '',
    notas: ''
  }}
>
```

- Añado los atributos name con el mismo nombre del objeto (el mismo que el id)

Entonces ahora cuando escriba en el input (del name), estaré llenando el del objeto initialValues

- Después del Formik, voy a meter todo el form en un arrow function dentro de llaves de javascript con el return implícito entre paréntesis
- Ahora en este arrow function hay mucha info.

Leyendo los Valores del Formulario y el Evento onSubmit

- Coloco el onSubmit en el Formik después del initialValues, con una arrow function
- Si le pongo como argumento values y pongo un console.log(values) veré un objeto con todos los name.
- Ahora mapea todos los name con el objeto initialValues y va escribiendo dentro de la variable values

```
<Formik
  initialValues={{
    nombre: '',
    empresa: '',
    email: '',
    telefono: '',
    notas: ''
  }}
  onSubmit={(values)=>{
    console.log(values)
  }}
>
```

- En lugar de un `console.log`, puedo colocarle un `handleSubmit` que todavía no he creado y pasarle los `values`
- Así ahora los tengo en la función que voy a escribir fuera del `return`
- Voy a llamarlos valores para no confundirme, y le pongo un `console.log`

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

```
const handleSubmit=(valores)=>{
  console.log(valores)
}
```

- Formik se lleva muy bien con Yup para hacer las validaciones

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: ''
})
```

JSON-SERVER

Qué es una REST API?

- REST = Representational State Transfer
- API = Application Programming Interface
- Debe responder a los request de HTTP:GET,POST,PUT,PATCH, DELETE
- Tiene una forma ordenada y estructurada de poner los recursos

Verbos HTTP:

- GET --> Obtener
- POST --> Enviar datos al Servidor/Creación
- PUT/PATCH --> Actualizar
- DELETE --> Eliminar
- Cuenta con endpoints (urls) para hacer CRUDS
- Para listar todos los clientes
 - GET /clientes
- Para obtener un solo cliente
 - GET /clientes/10 (el id)
- Crear un nuevo cliente
 - POST /clientes
- Editar un cliente
 - PUT /clientes/12 (el id)
- Borrar un cliente
 - DELETE /clientes/32 (el id)

A diferencia de una API, una REST API tiene esta estructura en sus rutas

Para la instalación hay que ejecutarlo como administrador

```
> npm i -g json-server
```

Creando la base de datos de mi primer REST API

Creo un archivo en la raíz llamado db.json Entre llaves y usando comillas dobles, creo el arreglo vacío clientes

```
{  
  "clientes": []  
}
```

- Para correr el servidor con el watch en el archivo por el puerto 4000

```
json-server --watch db.json --port 4000
```

- Para enviar los datos a la API, en Formulario.jsx hago async a handleSubmit
- Uso un try y un catch
- Cuando cre un registro, debo enviarlo a /clientes
- Cómo POST genera un nuevo registro.
- Por defecto, cuando hago un fetch, GET es por defecto
- Pero yo puedo especificarle el modo que quiero con esta sintaxis
 - No sé cuanto tiempo va a tomar la creación de ese nuevo registro, uso el await
 - Le estoy diciendo que a esta URL de tipo POST debo pasarle los datos del body
 - El server solo acepta strings. por eso el método stringify
 - Meto la respuesta formateada a json en una variable llamada resultado
 - Lo imprimo en consola

```
const handleSubmit=async(valores)=>{  
  try {  
    const url="http://localhost:4000/clientes"  
  
    const respuesta = await fetch(url, {  
      method: 'POST',  
      body: JSON.stringify(valores)  
    })  
  
    const resultado= await respuesta.json()  
    console.log(resultado)  
    console.log(respuesta)  
  
  } catch (error) {  
    console.log(error)  
  }  
}
```


- JSON server tiene sus reglas, y dice que cuando se usa una petición que no sea GET hay que agregarle estas líneas en el header

```
try {  
  const url="http://localhost:4000/clientes"  
  
  const respuesta = await fetch(url, {  
    method: 'POST',  
    body: JSON.stringify(valores),  
  
    headers:{  
      'Content-Type':'application/json'  
    }  
  })  
}
```

- De esta manera ya agrego con un id único que me provee json server los datos al servidor

Reiniciando Formulario

- Para limpiar el formulario, voy a Formulario.jsx
- En el onSubmit toma directamente los valores
- Puedo poner una coma y extraer con desestructuración la función resetForm
- Entonces le puedo decir que una vez que envíe los datos los borre del form
- Es un arrow function. Le puedo poner el async y que espere a enviar los datos

```
onSubmit={ async (values, {resetForm})=>{  
    await handleSubmit(values)  
    resetForm();  
}}
```

- Cuando se agrega un cliente, quiero redireccionar a Clientes para mostrar lo que tengo en la API
- Para redireccionar usaré el hook useNavigate de react-router-dom. Lo importo
- Guardo el hook en la variable navigate

```
const navigate= useNavigate()
```

- Lo coloco en el handleSubmit

```
const handleSubmit=async(valores)=>{  
    try {  
        const url="http://localhost:4000/clientes"  
  
        const respuesta = await fetch(url, {  
            method: 'POST',  
            body: JSON.stringify(valores),  
            headers:{  
                'Content-Type':'application/json'  
            }  
        })  
        const resultado = await respuesta.json()  
        navigate('/clientes')  
    } catch (error) {  
        console.log(error)  
    }  
}
```

- Reiniciar el formulario y redireccionar son buenas prácticas para no duplicar datos

Consultar la APi para obtener resultados

- Quiero mostrar los clientes en la pestaña clientes
- Usaré un useEffect en Inicio porque voy a llamar a la APi cuando el componente esté listo. Lo importo y lo coloco antes del return.
- Le paso el arreglo de dependencias vacío para que se ejecute una sola vez

```
useEffect(()=>{
  const obtenerClientesAPI = async()=>{

    try {
      const url="http://localhost:4000/clientes"

      const respuesta= await fetch(url);

      const resultado= await respuesta.json()
      console.log(resultado)

    } catch (error) {
      console.log(error)
    }
  }
  obtenerClientesAPI();
},[])
```

- Creo un state y lo llamo clientes, setClientes y lo inicio con un arreglo vacío
- En lugar del console.log pongo setClientes(resultado), lo que hace es llenar el state clientes con el resultado, que es el resultado de la promesa con los clientes

Mostrar Tabla

- Copio el fragment con los estilos de Nuevo Cliente y cambio los párrafos
- Le añado unas clases. Table-auto para cuando el resize las columnas se adapten
- Creo un thead y un tbody. El thead es la parte superior es para que corresponde cada columna. El body es dónde irán mis clientes
- Coloco un tr con varios th
- Coloco clientes entre llaves y lo mapeo. Pongo un return implícito en el .map y dentro un nuevo componente llamado Cliente
- Creo cliente, lo importo en Inicio y lo coloco en el .map con el id como key
- Le paso por props cliente al componente Cliente, de esta manera lo desestructuro y lo tengo disponible en Cliente para rellenar los campos con jsx

```
return (  
  <>  
  <h1 className= "font-black text-4xl text-blue-400">Clientes</h1>  
  <p className="mt-3">Administra tus clientes</p>  
  
  <table className="w-full mt-5 table-auto shadow bg-white"></table>  
    <thead className="bg-blue-900 text-white">  
      <tr>  
        <th className="p-2">Nombre</th>  
        <th className="p-2">Contacto</th>  
        <th className="p-2">Empresa</th>  
        <th className="p-2">Acciones</th>  
      </tr>  
    </thead>  
    <tbody>  
      {clientes.map(cliente=>(   
        <Cliente  
          key={cliente.id}  
          cliente={cliente}/>  
      ))}  
    </tbody>  
  
</>  
)
```

```
const Cliente = ({cliente}) => {  
  return (  
    <h1>{cliente.nombre}</h1>  
  )  
}
```

```
)
}
```

Mostrando la info de clientes

- Hay que extraer todas las propiedades del objeto cliente
- Construyo con html y clases el componente, con sus botones
- La clase border añade borders

```
const Cliente = ({cliente}) => {

  const {nombre, empresa, email, telefono, notas, id} = cliente

  return (
    <tr className="border-b hover:bg-gray-200 cursor-pointer">
      <td className="p-3">{nombre}</td>
      <td className="p-3">

        <p><span className="text-gray-800 uppercase font-bold">Email:
</span>{email}</p>

        <p><span className="text-gray-800 uppercase font-bold">Telefono:
</span>{telefono}</p>

      </td>

      <td className="p-3">{empresa}</td>
      <td className="p-3">

        <button
          className="bg-yellow-600 hover:bg-yellow-700 block
w-full text-white p-2 uppercase font-bold text-xs"
          type="button">Ver</button>

        <button
          className="bg-blue-600 hover:bg-blue-700 block
w-full text-white p-2 uppercase font-bold text-xs mt-3"
          type="button">Editar</button>

        <button
          className="bg-red-600 hover:bg-red-700 block
w-full text-white p-2 uppercase font-bold text-xs mt-3"
          type="button">borrar</button>

      </td>

    </tr>
  )
}
```

```
export default Cliente
```

Creando componente para ver-cliente

- Puedo usar el `useNavigate` en el `onClick` del `button` con un `arrow function` para redireccionar
- Lo hago entre `backticks` para inyectarle el `id` dinámico que estoy extrayendo de la desestructuración

```
<button
  className="bg-yellow-600 hover:bg-yellow-700 block
w-full text-white p-2 uppercase font-bold text-xs"
  type="button"
  onClick={()=>navigate(`/clientes/${id}`)}>Ver</button>
```

- Voy a `App.jsx` y en el `path` de un elemento que no he creado todavía llamado `VerCliente`, lo dejo sólo con `:id`
- Al tener esos dos puntos, lo va a tratar como una variable
- Ahora, cómo hago la consulta del `id` para asociarlo con los `id` de la API?
- Para eso hay otro `hook` llamado `useParams()`. Lo meto en la variable `params` y le hago un `console.log`

```
const params = useParams()

console.log(params)
```

- Lee de forma dinámica lo que hay en la `url`
- Me devuelve en consola un objeto con el `id`.
- Puedo extraer el `id` de `params` con desestructuración

```
const VerCliente = () => {

  const {id} = useParams()

  return (
    <h1>Ver cliente</h1>
  )
}
```

Mostrando la info de cliente por su id

- Hay que hacer de nuevo una consulta a la API desde `ver cliente` para mostrar la info
- Importo `useEffect` y el `useState` que usaré después
- Cuando se cargue el componente, cargará una vez y hará un llamado a la API, por eso el arreglo vacío
- Meto la `url` en un `templateString` para hacer dinámica la `url` con el `id`
- Pongo un `console.log` dentro del `try` con resultado, para visualizar el objeto que ya tengo

- Ese objeto es el que debo meter en el setState

```
const [cliente, setCliente] = useState({})
const {id} = useParams()
useEffect(()=>{
  const ObtenerClienteApi= async()=>{
    try {
      const url=`http://localhost:4000/clientes/${id}`
      const respuesta = await fetch(url)
      const resultado = await respuesta.json()
      setCliente(resultado)
    } catch (error) {
      console.log(error)
    }
  }
  ObtenerClienteApi()
},
[])
```

- Ahora solo tengo que darle formato con el html

```
<p className="text-2xl text-gray-700">
  <span className="uppercase font-bold">Cliente: </span>
  {cliente.nombre}
</p>
```

- Lo duplico el número de campos a mostrar

Añadiendo componente cuando no hay id

- Si me fijo, si coloco un id que no existe en consola muestra en el state un objeto vacío
- Puedo usar eso como condición en un ternario para mostrar el componente

```
Object.keys(cliente).length ===0 ? <p>No hay Resultados</p>:(
//Aquí dentro todo lo que hay en el return de Ver Cliente
```

Edición de clientes

- Voy al botón de editar y en el onClick le coloco un arrow function con un navigate con backticks para inyectarle el id


```
<button
  className="bg-blue-600 hover:bg-blue-700 block
    w-full text-white p-2 uppercase font-bold text-xs mt-3"
  type="button"
  onClick={()=>navigate(`/clientes/editar/${id}`)}>Editar</button>
```

- Copio y pego el contenido de NuevoCliente a EditarCliente
- Necesito importar useEffect para consultar la API, useState para setear el state con el resultado y useParams para leer el id que hay en la url
- Todo funciona exactamente igual que en VerCliente, porque necesito obtener el cliente que voy a editar. Copio y pego

```
const [cliente, setCliente] = useState({})
const {id} = useParams()

useEffect(()=>{
  const ObtenerClienteApi= async()=>{
    try {
      const url=`http://localhost:4000/clientes/${id}`
      const respuesta = await fetch(url)
      const resultado = await respuesta.json()
      setCliente(resultado)
    } catch (error) {
      console.log(error)
    }
  }
  ObtenerClienteApi()
},
[])
```

- Le paso por props a Formulario el cliente y lo extraigo en Formulario
- Formik tiene una prop llamada enableReinitialize que sirve para tomar valores iniciales que vienen desde una API. Por default es false
- La coloco en true después de initialValues
- Voy a usar defaultProps con cliente como objeto vacío para el nuevoCliente
- Funciona como los parámetros por default de las funciones

```
Formulario.defaultProps ={
  cliente: {}
}
export default Formulario
```

- Si no está presente, entran estas defaultProps
- Si ahora voy a NuevoCliente tiene un prop llamado cliente que está con un objeto vacío
- Solucionado esto, ahora si puedo usar initialValues con una condición.
- Si a initialValues le coloco cliente.nombre, lo imprime en el formulario
- Eso es gracias al enableReinitialize

Si quito enableReinitialize aparece en el objeto en consola pero no lo imprime en el campo. Por eso es importante.

- Ahora, si le doy a nuevoCliente y miro el initialValues en consola lo marca como undefined
- La idea es que inicie como un string vacío
- Para eso le añado un ternario con esta sintaxis
- Si está cliente.nombre añádelo, si no string vacío

```
<Formik
  initialValues={{
    nombre: cliente?.nombre ?? "",
    empresa: cliente?.empresa ?? "",
    email: cliente?.email ?? "",
    telefono: cliente?.telefono ?? "",
    notas: cliente?.notas ?? ""
  }}
  enableReinitialize={true}
  onSubmit={async(values, {resetForm})=>{
    await handleSubmit(values)

    resetForm();
  }}
/>
```

- Para mostrar el texto de fomra condicional

```
<h1 className="text-gray-600 font-bold text-3xl uppercase text-center">
  {cliente.nombre ? "Editar Cliente": "Agregar Cliente"}
</h1>
```

- Hago lo mismo con el botón
- Este parámetro ?? significa si el valor de la izquierda existe asignalo, y si no el de la derecha
- Coloco un ternario en EditarCliente para que no muestre el formulario si no hay cliente
 - en el caso de poner un id que no existe mostrará editar pero no el

```
return (  
  <>  
  <h1 className= "font-black text-4xl text-blue-400">Editar Cliente</h1>  
    <p className="mt-3">Utiliza este formulario para editar a un cliente</p>  
  
  {cliente.nombre ? (  
  
    <Formulario  
      cliente={cliente}/>  
  ): "No hay un cliente con ese id"  
  
  }  
</>  
)
```

Actualizando la API

- Para actualizar debo usar el método PUT
- Puedo usar la condición de si hay cliente.id para usar el método PUT
- Pongo la url con back ticks y añado el id dinámico al final, tal y como pide la API en el else pongo la petición GET. Es prácticamente el mismo código

```
const handleSubmit=async(valores)=>{
  try {
    if(cliente.id){
      const url=`http://localhost:4000/clientes/${cliente.id}`

      const respuesta = await fetch(url, {
        method: 'PUT',
        body: JSON.stringify(valores),
        headers:{
          'Content-Type':'application/json'
        }
      })
      const resultado = await respuesta.json()
      navigate('/clientes')

    }else{
      const url="http://localhost:4000/clientes"

      const respuesta = await fetch(url, {
        method: 'POST',
        body: JSON.stringify(valores),
        headers:{
          'Content-Type':'application/json'
        }
      })
      const resultado = await respuesta.json()
      navigate('/clientes')

    }
  } catch (error) {
    console.log(error)
  }
}
```

Eliminar Registro

- Me situo en Inicio.jsx ya que hay tengo el state de cliente
- Para que también desaparezca del DOM voy a usar setClientes
- Defino handleEliminar, se lo paso por props a Cliente y lo extraigo
- Lo coloco en el botón borrar en un arrow function y la invoco porque le voy a pasar un id

```
<button
  className="bg-red-600 hover:bg-red-700 block
w-full text-white p-2 uppercase font-bold text-xs mt-3"
  type="button"
  onClick={()=>handleEliminar()}>Borrar</button>
```

- Para probar si funciona, presiono eliminar y debería mostrar el console.log que le puse de prueba
- Requero el id que previamente ya lo extraí por desestructuración en Cliente
- Entonces le digo a la función en Inicio que va a llevar un id, que lo imprima para ver si funciona

```
const handleEliminar=(id)=>{
  console.log("Eliminando...", id)
}
```

- Escribo un confirm. Ahora tengo un valor de true o false si confirmo o cancelo para usarlo para llamar a la API

```
const handleEliminar=(id)=>{
  const confirmar = confirm("Deseas eliminar este cliente?")
  console.log(confirmar)
}
```

- Uso el método DELETE. La API funciona con id
- No guardo el resultado pq no me interesa, el await resp.json() lo elimina
- location.reload() refresca la página, pero no es buen método

```
const handleEliminar= async (id) => {
  const confirmar = confirm("Deseas eliminar este cliente?")
  if(confirmar){
    try {
      const url = `http://localhost:4000/clientes/${id}`
      const resp= await fetch(url,{
        method: 'DELETE'
      })
      await resp.json()
      location.reload()
    }
  }
}
```

```
    } catch (error) {  
      console.log(error)  
    }  
  }  
}
```

Es mejor actualizar el state

Le digo en el Array que quiero que me traiga a todos los clientes distintos al que estoy eliminando

```
await resp.json()  
const arrayClientes = clientes.filter(cliente=> cliente.id !== id)  
setClientes(arrayClientes)
```

```
import {Outlet, Link, useLocation} from 'react-router-dom'

const Layouts = () => {

  const location = useLocation()
  const urlActual = location.pathname

  return (
    <div className="md:flex md:min-h-screen">
      <div className="md:w-1/4 bg-blue-900 px-5 py-10">
        <h2 className="text-3xl font-black text-center text-white">CRM-
Clientes</h2>

        <nav className="mt-10">
          <Link to="/clientes"
            className={` ${urlActual === '/clientes'? 'text-blue-300': 'text-white'}
text-2xl block mt-2 hover:text-blue-300`}
            >Clientes</Link>
          <Link to="/clientes/nuevo"
            className= `${urlActual === '/clientes/nuevo'? 'text-blue-300': 'text-
white'} text-2xl block mt-2 hover:text-blue-300`}
            >Nuevo Cliente</Link>
        </nav>

      </div>

      <div className="md:w-3/4 p-10 md:h-screen overflow-scroll">
        <Outlet />

      </div>

    </div>
  )
}

export default Layouts
```

```
import {Formik, Form, Field } from 'formik'
import * as Yup from 'yup'
import Alerta from './Alerta'
import { useNavigate } from 'react-router-dom'

const Formulario = ({cliente}) => {

  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')
  })

  const navigate= useNavigate()

  const handleSubmit=async(valores)=>{
    try {
      if(cliente.id){
        const url=`http://localhost:4000/clientes/${cliente.id}`

        const respuesta = await fetch(url, {
          method: 'PUT',
          body: JSON.stringify(valores),
          headers:{
            'Content-Type':'application/json'
          }
        })
        const resultado = await respuesta.json()
        navigate('/clientes')

      }else{
        const url="http://localhost:4000/clientes"

        const respuesta = await fetch(url, {
          method: 'POST',
          body: JSON.stringify(valores),
          headers:{
            'Content-Type':'application/json'
          }
        })
        const resultado = await respuesta.json()
        navigate('/clientes')

      }
    } catch (error) {
      console.log(error)
    }
  }
}
```



```

return (
  <div className="bg-white px-5 py-10 mt-10 rounded-md shadow-md md:w-3/4 mx-
auto">

    <h1 className="text-gray-600 font-bold text-3xl uppercase text-center">
      {cliente?.nombre ? "Editar Cliente": "Agregar Cliente"}
    </h1>

    <Formik
      initialValues={{
        nombre: cliente?.nombre ?? "",
        empresa: cliente?.empresa ?? "",
        email: cliente?.email ?? "",
        telefono: cliente?.telefono ?? "",
        notas: cliente?.notas ?? ""
      }}
      enableReinitialize={true}
      onSubmit={async(values, {resetForm})=>{
        await handleSubmit(values)

        resetForm();
      }}
      validationSchema={nuevoClienteSchema}
    >
      ({({errors,touched})=>{
        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"/>
              {errors.nombre && touched.nombre? (
                <Alerta>{errors.nombre}</Alerta>
              ): null}
            </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"/>

```

```

        {errors.empresa && touched.empresa ? (
          <Alerta>{errors.empresa}</Alerta>
        ): null}
      </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"/>
        {errors.email && touched.email ? (
          <Alerta>{errors.email}</Alerta>
        ): null}

      </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"/>
        {errors.telefono && touched.telefono ? (
          <Alerta>{errors.telefono}</Alerta>
        ): null}

      </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={cliente.nombre? "Editar Cliente": "Agregar Cliente"}
        className="mt-5 w-full bg-blue-800 p-3 text-white uppercase font-bold
text-lg"/>
    </Form>
  )}}

```

```
        </Formik>

      </div>
    )
  }
  Formulario.defaultProps = {
    cliente: {}
  }
  export default Formulario
```

```
import React from 'react'

const LoginForm = () => {
  return (
    <div>
      <h1>Login Form</h1>
    </div>
  )
}

export default LoginForm
```

```
import {useParams} from 'react-router-dom'
import {useEffect, useState} from 'react'

const VerCliente = () => {
  const [cliente, setCliente] = useState({})
  const {id} = useParams()

  useEffect(()=>{
    const ObtenerClienteApi= async()=>{
      try {
        const url=`http://localhost:4000/clientes/${id}`
        const respuesta = await fetch(url)
        const resultado = await respuesta.json()
        setCliente(resultado)
      } catch (error) {
        console.log(error)
      }
    }

    ObtenerClienteApi()
  },
  [])

  return (
    Object.keys(cliente).length ===0 ? <p>No hay Resultados</p>:(
    <div>
      <h1 className="font-black text-4xl text-blue-800">Ver Cliente:
{cliente.nombre}</h1>
      <p>Información del cliente</p>

      <p className="text-2xl text-gray-700 mt-4">
        <span className="uppercase font-bold">Cliente: </span>
        {cliente.nombre}
      </p>
      <p className="text-2xl text-gray-700 mt-4">
        <span className="uppercase font-bold">Email: </span>
        {cliente.email}
      </p>
      <p className="text-2xl text-gray-700 mt-4">
        <span className="uppercase font-bold">Telefono: </span>
        {cliente.telefono}
      </p>
      <p className="text-2xl text-gray-700 mt-4">
        <span className="uppercase font-bold">Empresa: </span>
        {cliente.empresa}
      </p>
      {cliente.notas && (
      <p className="text-2xl text-gray-700 mt-4">
        <span className="uppercase font-bold">Notas: </span>
        {cliente.notas}
      </p>
    )
  )
    )
  )
}
```

```
        </p>
    })

    </div>)
}

export default VerCliente
```

```
import React from 'react'

const LoginForm = () => {
  return (
    <div>
      <h1>Login Form</h1>
    </div>
  )
}

export default LoginForm
```

```
import {Formik, Form, Field } from 'formik'
import * as Yup from 'yup'
import Alerta from './Alerta'
import { useNavigate } from 'react-router-dom'

const Formulario = ({cliente}) => {

  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')
  })

  const navigate= useNavigate()

  const handleSubmit=async(valores)=>{
    try {
      if(cliente.id){
        const url=`http://localhost:4000/clientes/${cliente.id}`

        const respuesta = await fetch(url, {
          method: 'PUT',
          body: JSON.stringify(valores),
          headers:{
            'Content-Type':'application/json'
          }
        })
        const resultado = await respuesta.json()
        navigate('/clientes')

      }else{
        const url="http://localhost:4000/clientes"

        const respuesta = await fetch(url, {
          method: 'POST',
          body: JSON.stringify(valores),
          headers:{
            'Content-Type':'application/json'
          }
        })
        const resultado = await respuesta.json()
        navigate('/clientes')

      }
    } catch (error) {
      console.log(error)
    }
  }
}
```



```

return (
  <div className="bg-white px-5 py-10 mt-10 rounded-md shadow-md md:w-3/4 mx-
auto">

    <h1 className="text-gray-600 font-bold text-3xl uppercase text-center">
      {cliente?.nombre ? "Editar Cliente": "Agregar Cliente"}
    </h1>

    <Formik
      initialValues={{
        nombre: cliente?.nombre ?? "",
        empresa: cliente?.empresa ?? "",
        email: cliente?.email ?? "",
        telefono: cliente?.telefono ?? "",
        notas: cliente?.notas ?? ""
      }}
      enableReinitialize={true}
      onSubmit={async(values, {resetForm})=>{
        await handleSubmit(values)

        resetForm();
      }}
      validationSchema={nuevoClienteSchema}
    >
      {({errors,touched})=>{
        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"/>
              {errors.nombre && touched.nombre? (
                <Alerta>{errors.nombre}</Alerta>
              ): null}
            </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"/>

```

```
        {errors.empresa && touched.empresa ? (  
          <Alerta>{errors.empresa}</Alerta>  
        ): null}  
      </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"/>  
          {errors.email && touched.email ? (  
            <Alerta>{errors.email}</Alerta>  
          ): null}  
  
      </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"/>  
          {errors.telefono && touched.telefono ? (  
            <Alerta>{errors.telefono}</Alerta>  
          ): null}  
  
      </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={cliente.nombre? "Editar Cliente": "Agregar Cliente"}  
        className="mt-5 w-full bg-blue-800 p-3 text-white uppercase font-bold  
text-lg"/>  
    </Form>  
  )}}
```

```
      </Formik>

    </div>
  )
}
Formulario.defaultProps = {
  cliente: {}
}
export default Formulario
```

```
import {useState,useEffect} from 'react'
import { useParams } from 'react-router-dom'
import Formulario from '../components/Formulario'

const EditarCliente = () => {

  const [cliente, setCliente] = useState({})
  const {id} = useParams()

  useEffect(()=>{
    const ObtenerClienteApi= async()=>{
      try {
        const url=`http://localhost:4000/clientes/${id}`
        const respuesta = await fetch(url)
        const resultado = await respuesta.json()
        setCliente(resultado)
      } catch (error) {
        console.log(error)
      }
    }
    ObtenerClienteApi()
  },
  [])
  return (
    <>
    <h1 className= "font-black text-4xl text-blue-400">Editar Cliente</h1>
    <p className="mt-3">Utiliza este formulario para editar a un cliente</p>

    {cliente.nombre ?(

      <Formulario
        cliente={cliente}/>
    ): "No hay un cliente con ese id"

    }
    </>
  )
}

export default EditarCliente
~~~
```

```
import {BrowserRouter, Routes, Route} from 'react-router-dom'
import Layout from '../src/layouts/Layouts'
import Inicio from '../src/paginas/Inicio'
import NuevoCliente from '../src/paginas/NuevoCliente'
import EditarCliente from '../src/paginas/EditarCliente'
import VerCliente from '../src/paginas/VerCliente'

function App() {

  return (
    <BrowserRouter>
      <Routes>

        <Route path="/clientes" element={<Layout />}>
          <Route index element={<Inicio />} />
          <Route path="nuevo" element={<NuevoCliente />} />
          <Route path="editar/:id" element={<EditarCliente />} />
          <Route path=":id" element={<VerCliente />} />
        </Route>
      </Routes>

    </BrowserRouter>
  )
}

export default App
```