

AJUSTES CRUD – MEJORANDO PRESENTACION.

Vamos a mejorar la presentación de nuestro front agregando algunos detalles que no lo permitan y además vamos a controlar nuestros avisos.

1. Font Awesome: (<https://fontawesome.com/v5.15/how-to-use/on-the-web/using-with/react>).

Font Awesome ahora tiene un componente oficial de React que está disponible de tal forma que se pueda usar nuestros íconos en sus aplicaciones de React de manera sencilla.

Para comenzar, deberá instalar los siguientes paquetes en su proyecto utilizando un administrador de paquetes como npm e yarn. Aquí hay ejemplos que instalan todo lo que necesita y nuestro estilo sólido de íconos usando cada administrador de paquetes respectivo.

```
$ npm i --save @fortawesome/fontawesome-svg-core  
npm install --save @fortawesome/free-solid-svg-icons  
npm install --save @fortawesome/react-fontawesome
```

Usando el proyecto del CRUD de clientes agregamos al componente Clientes.js las siguientes líneas que importan los íconos a utilizar.

```
4 import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';  
5 import { faEdit, faTrashAlt } from '@fortawesome/free-solid-svg-icons';
```





Dentro de la tabla se hacen los siguientes ajustes:

```

    /* <td>
      <NavLink to={"/detalles/" + cliente._id}>Detalles</NavLink>
    </td> */
    <td>
      <NavLink to={"/update/" + cliente._id} className="btn btn-primary">
        <FontAwesomeIcon icon={faEdit}/></NavLink>
      </td>
    <td>
      <NavLink to={"/delete/" + cliente._id} className="btn btn-danger">
        <FontAwesomeIcon icon={faTrashAlt}/></NavLink>
      </td>

```

La apariencia de la pantalla en la sección de las acciones quedará como se muestra a continuación:

Teléfono	Editar	Borrar
3124567711		
3126789111		

2. Agregar ícono al menú.

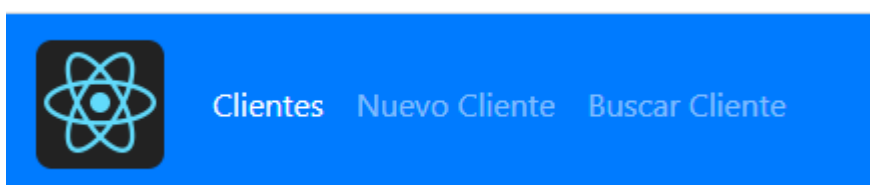
Aprovechando la existencia del ícono de React, lo agregamos al menú creado en el proyecto en las líneas en donde aparece el texto Navbar,

```

export default class MenuClientes extends Component {
  render() {
    return (
      <nav className="navbar navbar-expand-lg navbar-dark bg-primary">
        <a className="navbar-brand" href="/"></a>

```

La barra de navegación tendrá la siguiente apariencia:



3. Notificaciones al usuario.

A la hora de crear modales, popups y alertas con Javascript, pueden utilizarse diferentes bibliotecas u opciones. Una de esas bibliotecas es SweetAlert2.

SweetAlert2 es una biblioteca que está pensada para poder utilizarse junto a JavaScript Vanilla. Al estar pensado para usarse de esa manera, también funciona con React, Vue y Angular, aunque en este documento se utilizará con React.

Para saber que pasa en ciertos momentos dentro del aplicativo vamos a utilizar SweetAlert2 cuya página oficial se encuentra en:

<https://sweetalert2.github.io/recipe-gallery/sweetalert2-react.html>

Podemos encontrar en el sitio las instrucciones y algunos ejemplos para poder usarlos en nuestro proyecto, lo primero que debemos hacer es instalar el componente con la siguiente instrucción:

```
npm install --save/sweetalert2/sweetalert2-react-content
```

Una vez que se hallan terminado de instalar todas los paquetes y dependencias, ya se podrá empezar a trabajar.

Para trabajar con ello, es necesario importar en el componente la biblioteca, tal y como se muestra a continuación:

```
import Swal from "sweetalert2"
```

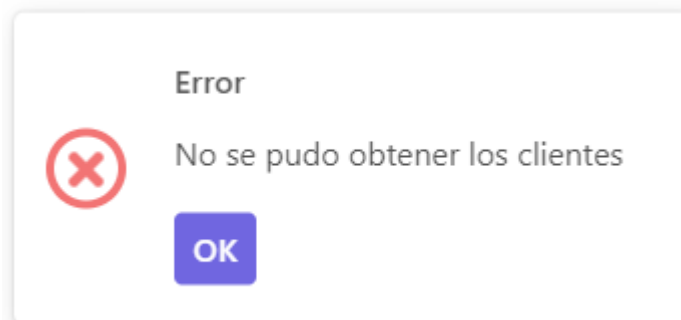
Vamos a hacer una prueba sobre el cargue de clientes ajustando el componente Clientes.js de la siguiente manera:

```

cargarClientes = () => {
  var url = Global.urlclientes;
  var request = "/clientes";
  axios.get(url + request).then((res) => {
    this.setState({
      clientes: res.data
      , status: true
    });
  }).catch(error=>{
    Swal.fire({
      icon: 'error',
      title: 'Error',
      text: 'No se pudo obtener los clientes',
      toast: true
    });
    console.log(error);
  });
}

```

Al suspender la ejecución del api y recargar el front nos muestra un mensaje como el que sigue:



4. Ajustar todos los elementos del CRUD para poder ajustar los mensajes solicitados en las pruebas.

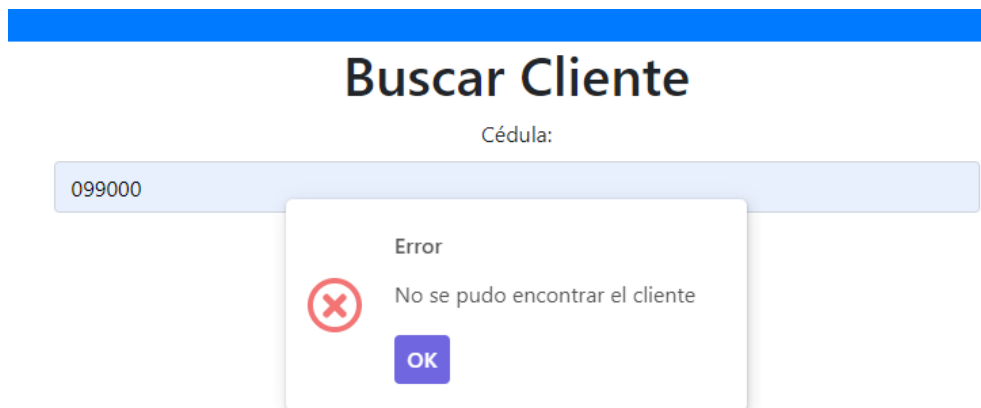
Por ejemplo al momento de consultar un cliente y no encontrarlo debe aparecer un mensaje de error, debemos ajustar el componente `BuscarCliente.js`.

```

buscarCliente = (e) => {
  e.preventDefault();
  var ced = parseInt(this.cajaCedRef.current.value);
  console.log({ced});
  var request = `/buscar/${ced}`;
  var url = Global.urlclientes + request;
  axios.get(url).then(res => {
    this.setState({
      cliente: res.data,
      encontrado: true
    , status: true
    });
  }).catch(error=>{
    Swal.fire({
      icon: 'error',
      title: 'Error',
      text: 'No se pudo encontrar el cliente',
      toast: true
    });
    this.setState({
      encontrado: false
    , status: false
    });
    console.log(error);
  });
}

```

El resultado en el navegador sería como el se aprecia en la siguiente gráfica:



Modificamos el componente UpdateCliente.js de tal manera que si la actualización es exitosa de un aviso sobre ello:

```
modificarCliente = (e) => {  
  e.preventDefault();  
  var ced = this.cajaCedRef.current.value;  
  var nom = this.cajaNomRef.current.value;  
  var dir = this.cajaDirRef.current.value;  
  var ema = this.cajaEmaRef.current.value;  
  var tel = this.cajaTelRef.current.value;  
  var cliente = {  
    cedula: ced,  
    nombre: nom,  
    direccion: dir,  
    email: ema,  
    telefono: tel  
  };  
  var request = "/clientes/" + this.props._id;  
  var url = Global.urlclientes + request;  
  axios.put(url, cliente).then(res => {  
    this.setState({status: true});  
    Swal.fire({  
      icon: 'sucess',  
      title: 'Actualizado',  
      text: 'El cliente se Actualizó con Exito',  
      toast: true  
    });  
  });  
};
```

El resultado de ejecutar la actualización se muestra con la siguiente figura:

