



ACTIVIDAD FINAL: CRUD COMPLETO

Desarrollo web



Alumno: Navarrete Diaz Luis Alejandro

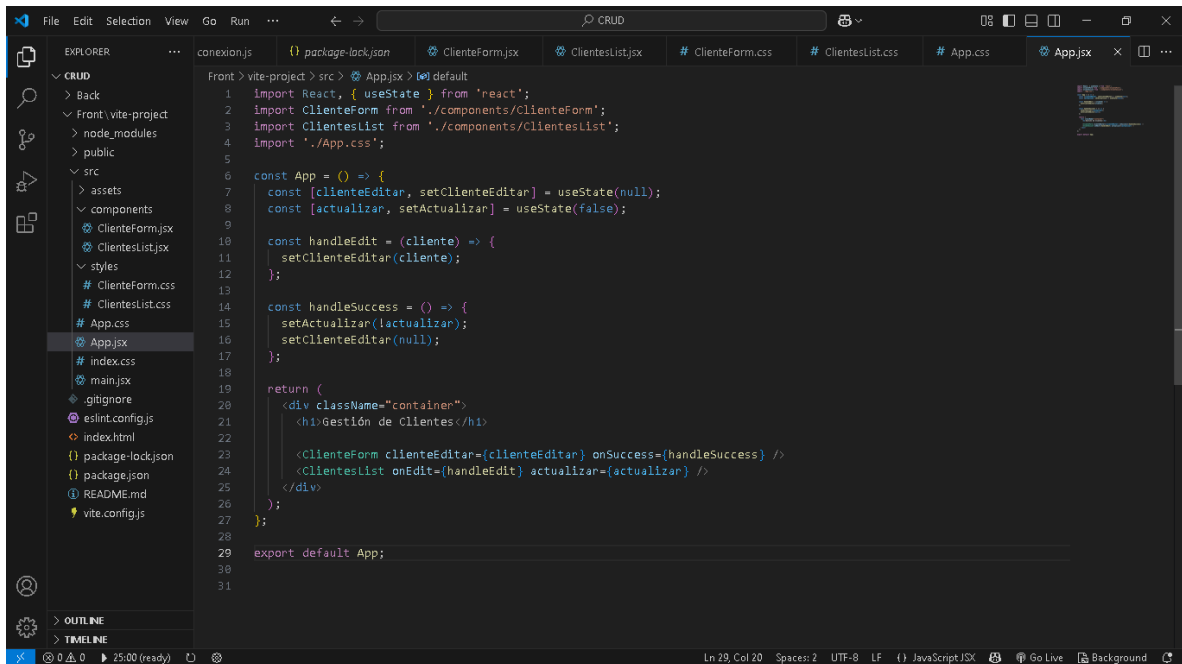
Grupo: 2-3

Maestro: José Manuel Cazarez Alderete

**30 DE MAYO DE 2025
FACULTAD DE INFORMATICA
UAS**

Documentación de Back.

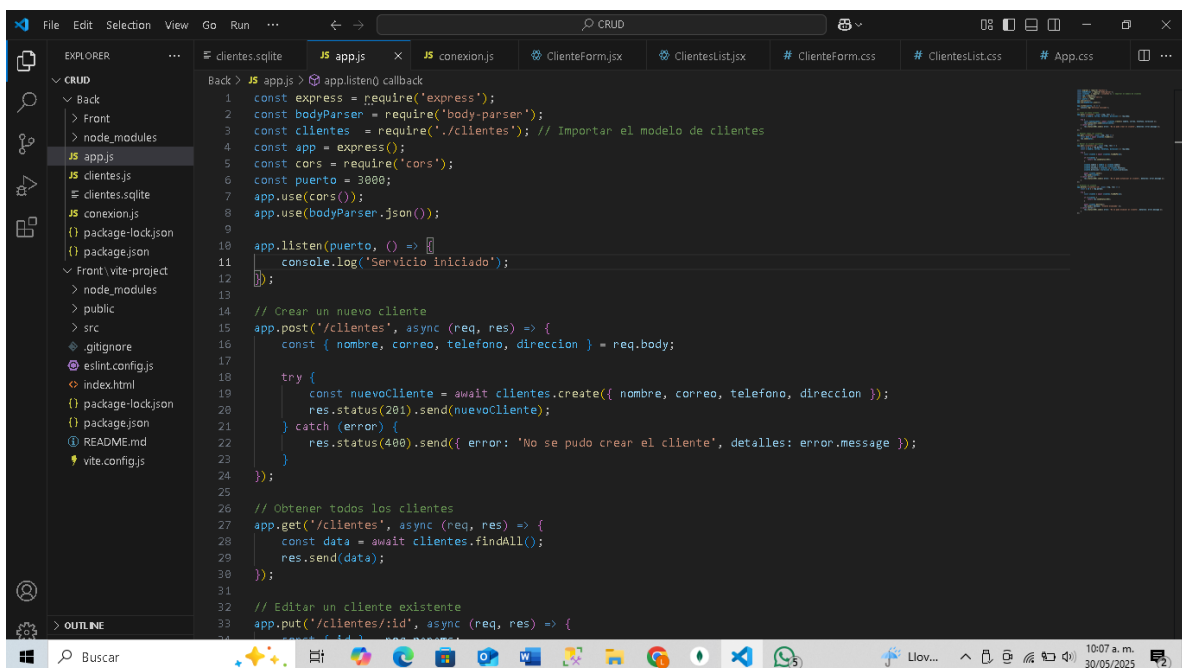
-Estructura de carpetas.



The screenshot shows the VS Code interface with the Explorer sidebar on the left. The file explorer shows a project structure with folders like 'Back', 'Front', 'node_modules', 'public', 'src', 'assets', 'components', 'styles', and 'App.jsx'. The main editor displays the content of 'App.jsx', which is a React component. The code includes imports for React, useState, ClienteForm, ClientesList, and App.css. It defines a functional component 'App' that uses useState to manage 'clienteEditar' and 'actualizar' states. It also defines 'handleEdit' and 'handleSuccess' functions. The component renders a 'div' with a 'container' class, containing a 'h1' tag with the text 'Gestión de Clientes', a 'ClienteForm' component, and a 'ClientesList' component. The 'App' component is exported as the default export.

```
1 import React, { useState } from 'react';
2 import ClienteForm from './components/ClienteForm';
3 import ClientesList from './components/ClientesList';
4 import './App.css';
5
6 const App = () => {
7   const [clienteEditar, setClienteEditar] = useState(null);
8   const [actualizar, setActualizar] = useState(false);
9
10  const handleEdit = (cliente) => {
11    setClienteEditar(cliente);
12  };
13
14  const handleSuccess = () => {
15    setActualizar(actualizar);
16    setClienteEditar(null);
17  };
18
19  return (
20    <div className="container">
21      <h1>Gestión de Clientes</h1>
22
23      <ClienteForm clienteEditar={clienteEditar} onSuccess={handleSuccess} />
24      <ClientesList onEdit={handleEdit} actualizar={actualizar} />
25    </div>
26  );
27
28
29 export default App;
```

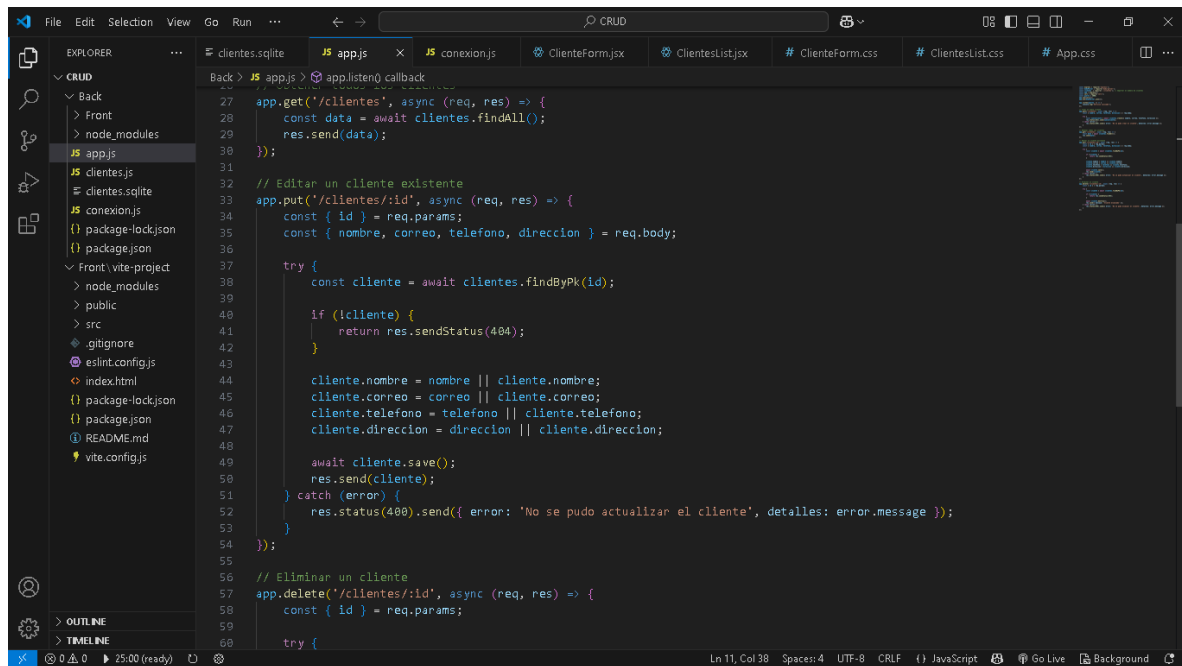
Codigo App.js de express. Utilice la base de datos de clientes que antes habíamos trabajo donde se uso un get /clientes para obtener un listado de todos los clientes que están en la base de datos. Con el codigo findAll() funciona el proceso. Tambien esta la ruta post que es para crear los clientes y verificar que este correcto el proceso.



The screenshot shows the VS Code interface with the Explorer sidebar on the left. The file explorer shows a project structure with folders like 'Back', 'Front', 'node_modules', 'public', 'src', 'assets', 'components', 'styles', and 'App.jsx'. The main editor displays the content of 'app.js', which is a Node.js application using Express. The code includes imports for express, body-parser, clientes, and cons. It defines a functional application 'app' that uses body-parser and cons. It also defines 'app.listen' and 'app.post' methods. The 'app.post' method is used to create a new client, and the 'app.get' method is used to get all clients. The 'app.put' method is used to edit an existing client. The application is started by calling 'app.listen'.

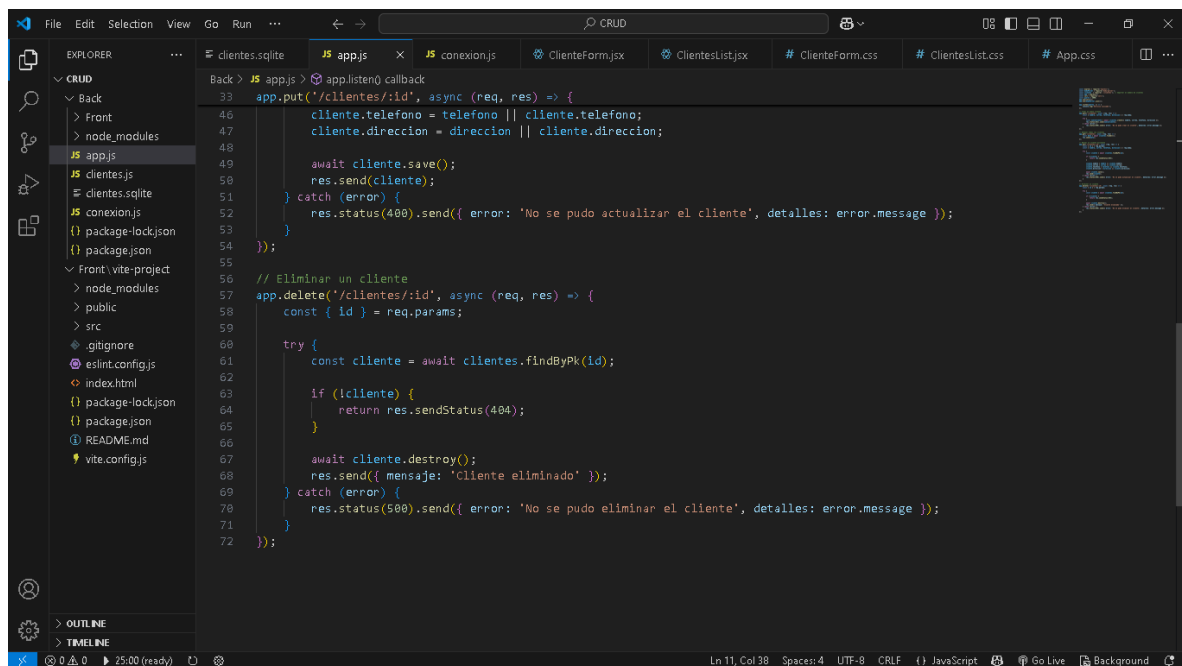
```
1 const express = require('express');
2 const bodyParser = require('body-parser');
3 const clientes = require('./clientes'); // Importar el modelo de clientes
4 const app = express();
5 const cons = require('cons');
6 const puerto = 3000;
7 app.use(cons());
8 app.use(bodyParser.json());
9
10 app.listen(puerto, () => {
11   console.log('Servicio Iniciado');
12 });
13
14 // Crear un nuevo cliente
15 app.post('/clientes', async (req, res) => {
16   const { nombre, correo, telefono, direccion } = req.body;
17
18   try {
19     const nuevoCliente = await clientes.create({ nombre, correo, telefono, direccion });
20     res.status(201).send(nuevoCliente);
21   } catch (error) {
22     res.status(400).send({ error: 'No se pudo crear el cliente', detalles: error.message });
23   }
24 });
25
26 // Obtener todos los clientes
27 app.get('/clientes', async (req, res) => {
28   const data = await clientes.findAll();
29   res.send(data);
30 });
31
32 // Editar un cliente existente
33 app.put('/clientes/:id', async (req, res) => {
34   const { id } = req.params;
```

Esta la ruta put('/clientes/:id') que como parámetro se requiere el id para saber que cliente se va actualizar, como variables se utiliza el nombre, correo,telefono y direccion y se necesita rellenar todos los campos.



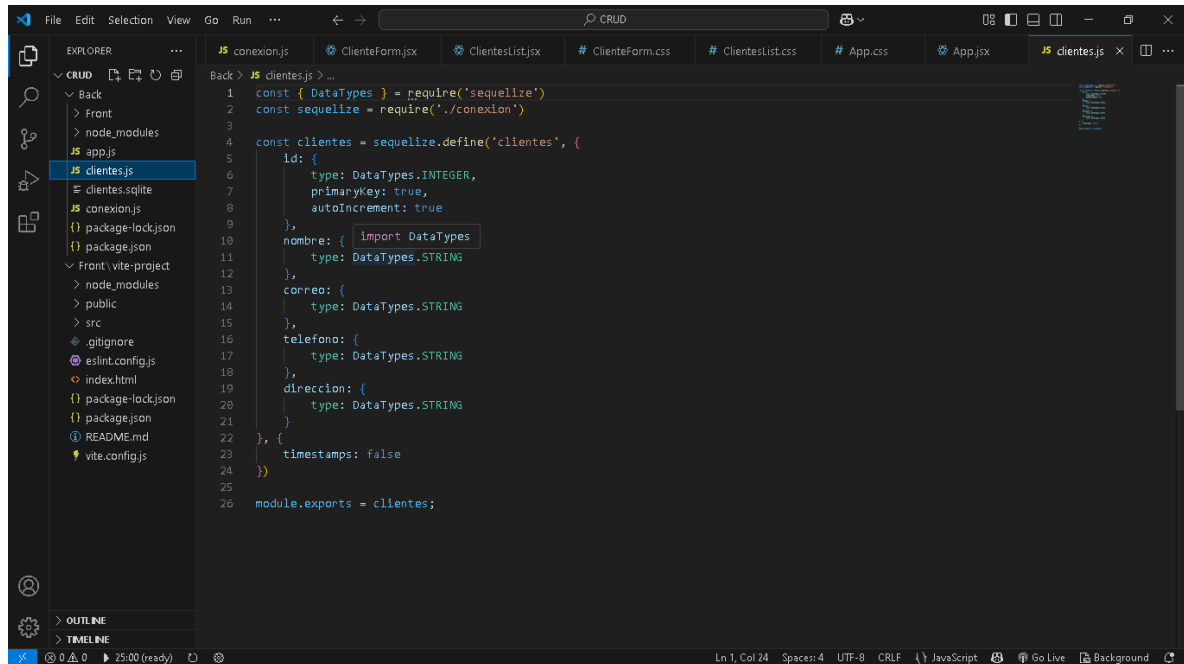
```
Back > JS appjs > app.listen() callback
27 app.get('/clientes', async (req, res) => {
28   const data = await clientes.findAll();
29   res.send(data);
30 });
31
32 // Editar un cliente existente
33 app.put('/clientes/:id', async (req, res) => {
34   const { id } = req.params;
35   const { nombre, correo, telefono, direccion } = req.body;
36
37   try {
38     const cliente = await clientes.findByIdPk(id);
39
40     if (!cliente) {
41       return res.sendStatus(404);
42     }
43
44     cliente.nombre = nombre || cliente.nombre;
45     cliente.correo = correo || cliente.correo;
46     cliente.telefono = telefono || cliente.telefono;
47     cliente.direccion = direccion || cliente.direccion;
48
49     await cliente.save();
50     res.send(cliente);
51   } catch (error) {
52     res.status(400).send({ error: 'No se pudo actualizar el cliente', detalles: error.message });
53   }
54 });
55
56 // Eliminar un cliente
57 app.delete('/clientes/:id', async (req, res) => {
58   const { id } = req.params;
59
60   try {
```

Igual tenemos la ruta delete('/clientes/:id') que sirve mas que nada para eliminar el cliente por lo que seria el id, se necesita para eliminarlos.



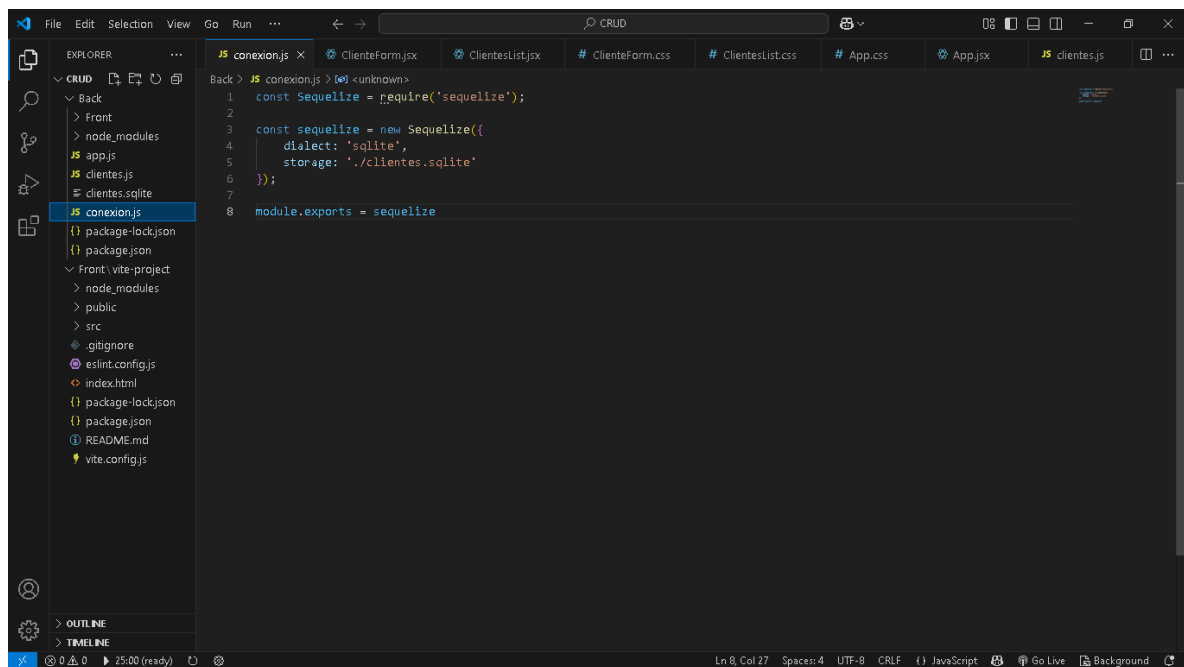
```
Back > JS appjs > app.listen() callback
33 app.put('/clientes/:id', async (req, res) => {
46   cliente.telefono = telefono || cliente.telefono;
47   cliente.direccion = direccion || cliente.direccion;
48
49   await cliente.save();
50   res.send(cliente);
51 } catch (error) {
52   res.status(400).send({ error: 'No se pudo actualizar el cliente', detalles: error.message });
53 }
54 });
55
56 // Eliminar un cliente
57 app.delete('/clientes/:id', async (req, res) => {
58   const { id } = req.params;
59
60   try {
61     const cliente = await clientes.findByIdPk(id);
62
63     if (!cliente) {
64       return res.sendStatus(404);
65     }
66
67     await cliente.destroy();
68     res.send({ mensaje: 'cliente eliminado' });
69   } catch (error) {
70     res.status(500).send({ error: 'No se pudo eliminar el cliente', detalles: error.message });
71   }
72 });
```

El modelo clientes que es la tabla y la declaración de campos. Se requiere tambien la conexión a la base de datos y el sequelize.



```
Back > JS clientes.js > ...
1  const { DataTypes } = require('sequelize')
2  const sequelize = require('./conexion')
3
4  const clientes = sequelize.define('clientes', {
5
6    id: {
7      type: DataTypes.INTEGER,
8      primaryKey: true,
9      autoIncrement: true
10   },
11   nombre: {
12     type: DataTypes.STRING
13   },
14   correo: {
15     type: DataTypes.STRING
16   },
17   telefono: {
18     type: DataTypes.STRING
19   },
20   direccion: {
21     type: DataTypes.STRING
22   },
23   timestamps: false
24 }, {
25   module.exports = clientes;
```

El modelo conexión, que es para enlazar la base de datos de clientes y su modelo sqlite. Los demás archivos son de express, sequelize y sqlite3.

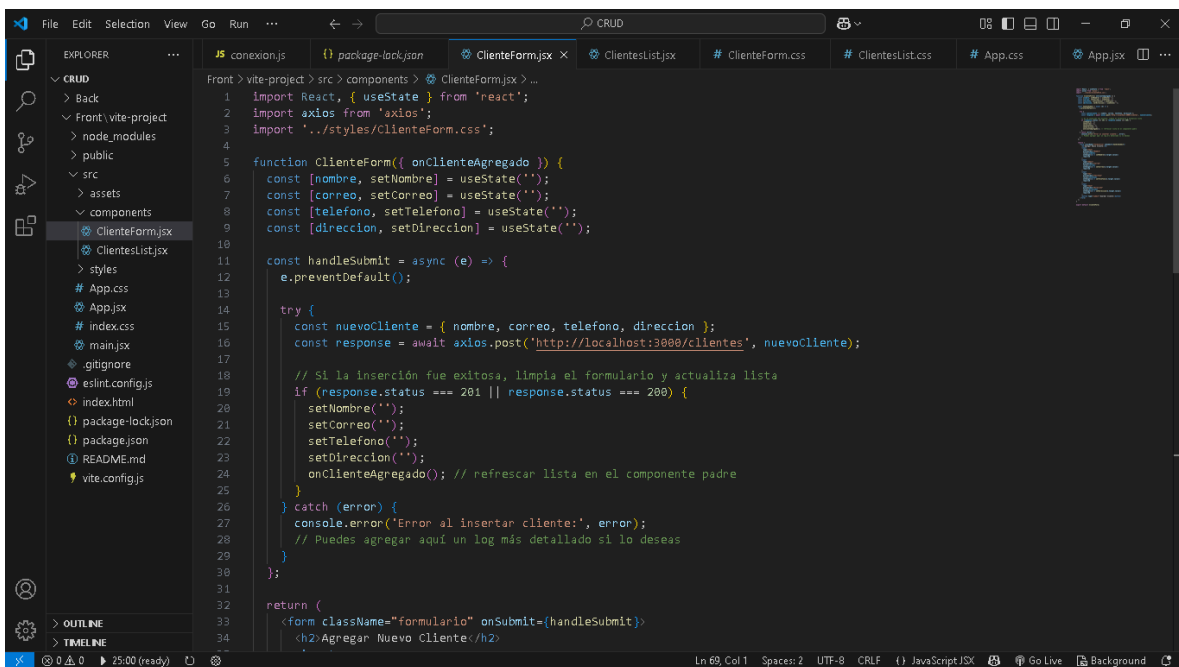


```
Back > JS conexion.js > <unknown>
1  const Sequelize = require('sequelize');
2
3  const sequelize = new Sequelize({
4    dialect: 'sqlite',
5    storage: './clientes.sqlite'
6  });
7
8  module.exports = sequelize
```

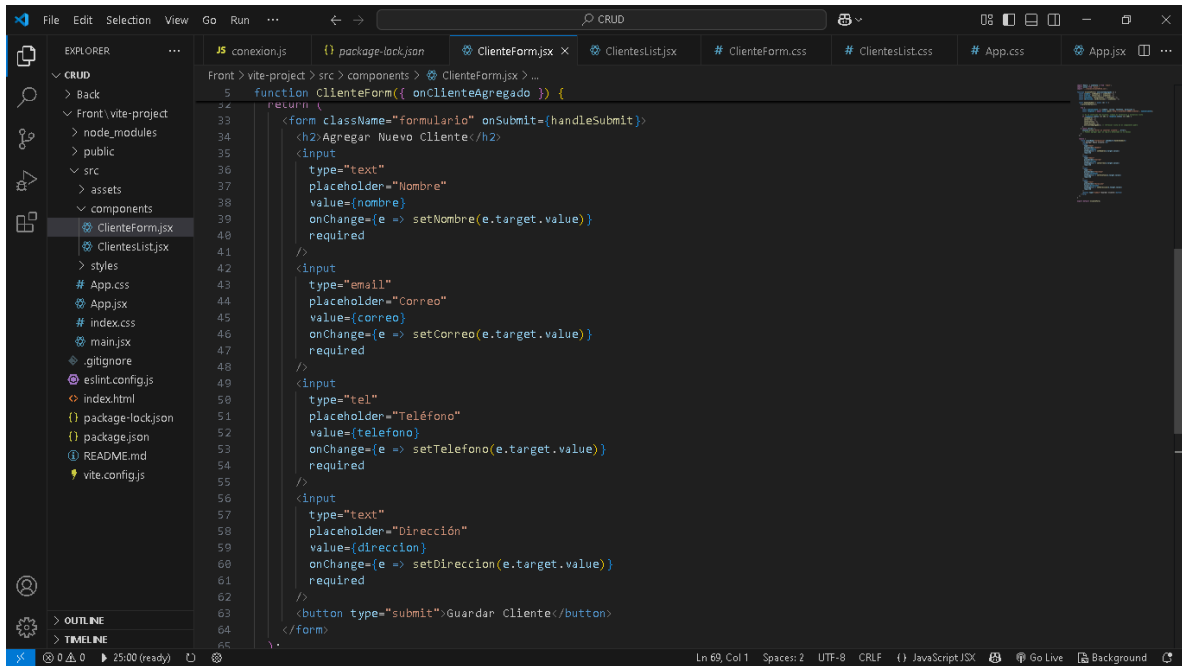
Documentación de Front.

Se necesitaron dos componentes que es el formulario de clientes y el de la lista de clientes.

Este componente ClienteForm en React permite a los usuarios agregar nuevos clientes a través de un formulario. Utiliza el hook useState para manejar el estado local de cada campo del formulario (nombre, correo, teléfono y dirección). Al enviar el formulario, se ejecuta la función handleSubmit, que previene el comportamiento por defecto del navegador y envía los datos ingresados a un servidor mediante una solicitud POST usando axios. Si la solicitud es exitosa, los campos del formulario se vacían y se ejecuta una función (onClienteAgregado) pasada desde el componente padre para actualizar la lista de clientes. El formulario está estilizado con una clase CSS externa y todos los inputs están controlados, lo que significa que su valor depende del estado del componente. Finalmente, el componente se exporta para ser utilizado en otras partes de la aplicación.



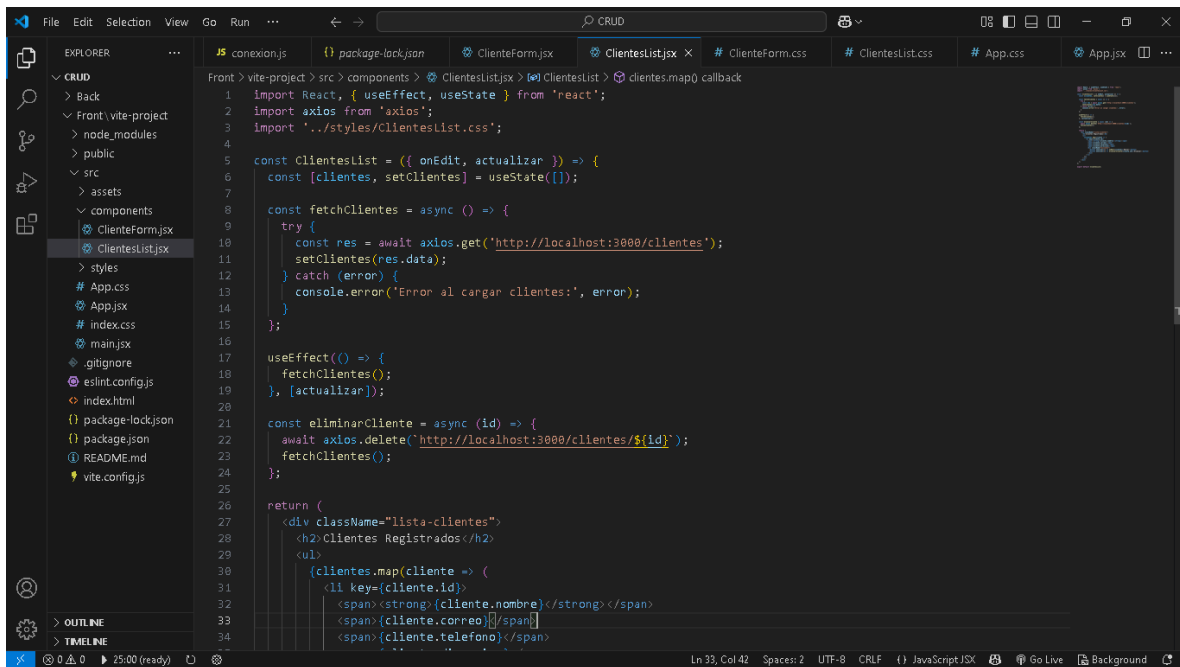
```
1 import React, { useState } from 'react';
2 import axios from 'axios';
3 import '../styles/ClienteForm.css';
4
5 function ClienteForm({ onClienteAgregado }) {
6   const [nombre, setNombre] = useState('');
7   const [correo, setCorreo] = useState('');
8   const [telefono, setTelefono] = useState('');
9   const [direccion, setDireccion] = useState('');
10
11   const handleSubmit = async (e) => {
12     e.preventDefault();
13
14     try {
15       const nuevoCliente = { nombre, correo, telefono, direccion };
16       const response = await axios.post('http://localhost:3000/clientes', nuevoCliente);
17
18       // Si la inserción fue exitosa, limpia el formulario y actualiza lista
19       if (response.status === 201 || response.status === 200) {
20         setNombre('');
21         setCorreo('');
22         setTelefono('');
23         setDireccion('');
24         onClienteAgregado(); // refrescar lista en el componente padre
25       }
26     } catch (error) {
27       console.error('Error al insertar cliente:', error);
28       // Puedes agregar aquí un log más detallado si lo deseas
29     }
30   };
31
32   return (
33     <form className="formulario" onSubmit={handleSubmit}>
34       <h2>Agregar Nuevo Cliente</h2>
```



```
Front > vite-project > src > components > ClienteForm.jsx > ...
5 function ClienteForm({ onClienteAgregado }) {
24   return (
33     <form className="formulario" onSubmit={handleSubmit}>
34       <h2>Agregar Nuevo Cliente</h2>
35       <input
36         type="text"
37         placeholder="Nombre"
38         value={nombre}
39         onChange={e => setNombre(e.target.value)}
40         required
41       />
42       <input
43         type="email"
44         placeholder="Correo"
45         value={correo}
46         onChange={e => setCorreo(e.target.value)}
47         required
48       />
49       <input
50         type="tel"
51         placeholder="Teléfono"
52         value={telefono}
53         onChange={e => setTelefono(e.target.value)}
54         required
55       />
56       <input
57         type="text"
58         placeholder="Dirección"
59         value={direccion}
60         onChange={e => setDireccion(e.target.value)}
61         required
62       />
63       <button type="submit">Guardar Cliente</button>
64     </form>
65   );
}
```

CienteList.

El componente ClientesList en React se encarga de mostrar una lista de clientes previamente registrados, recuperados desde un backend. Utiliza el hook `useState` para almacenar los datos de los clientes y `useEffect` para cargar automáticamente la lista al montar el componente o cuando cambie la prop `actualizar`. La función `fetchClientes` obtiene los clientes con una solicitud GET a un servidor local utilizando `axios`, y almacena los resultados en el estado. También incluye una función `eliminarCliente`, que permite eliminar un cliente del servidor mediante una solicitud DELETE, seguida de una recarga de la lista. El componente recibe dos props: `onEdit`, que se ejecuta al hacer clic en el botón “Editar” para pasar los datos del cliente al componente padre, y `actualizar`, que actúa como señal para recargar la lista. Finalmente, la lista se muestra en la interfaz con opciones para editar o eliminar cada cliente, con estilo definido por una hoja de CSS externa.



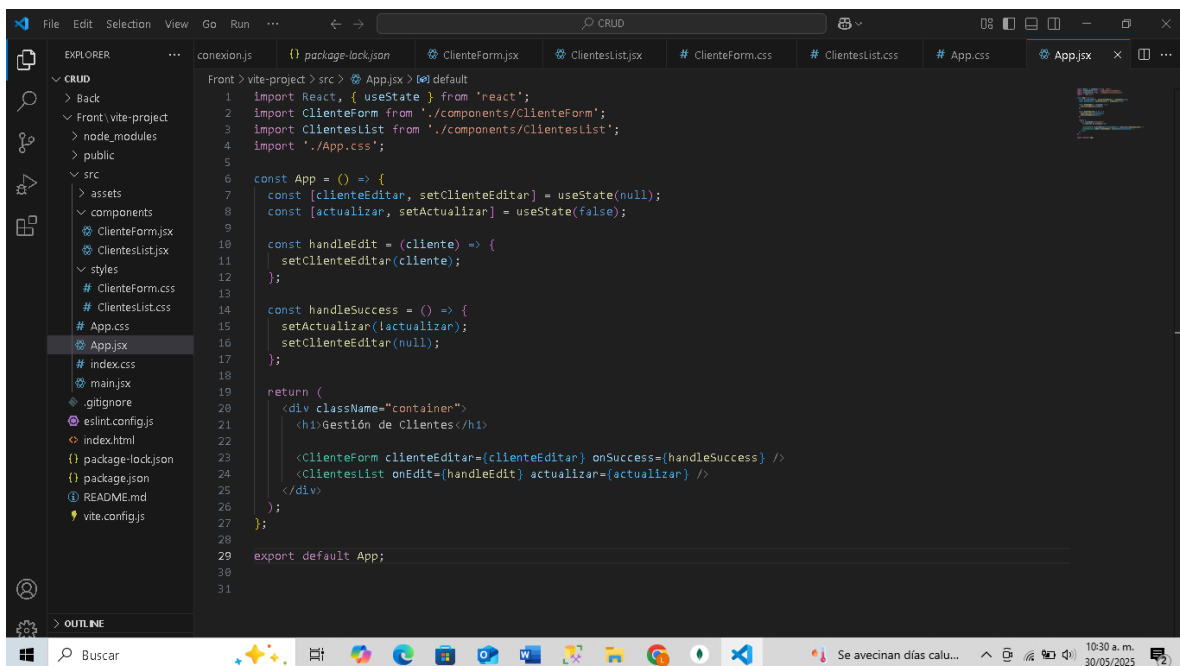
```
1 import React, { useEffect, useState } from 'react';
2 import axios from 'axios';
3 import '../styles/ClientesList.css';
4
5 const ClientesList = ({ onEdit, actualizar }) => {
6   const [clientes, setClientes] = useState([]);
7
8   const fetchClientes = async () => {
9     try {
10       const res = await axios.get('http://localhost:3000/clientes');
11       setClientes(res.data);
12     } catch (error) {
13       console.error('Error al cargar clientes:', error);
14     }
15   };
16
17   useEffect(() => {
18     fetchClientes();
19   }, [actualizar]);
20
21   const eliminarCliente = async (id) => {
22     await axios.delete('http://localhost:3000/clientes/${id}');
23     fetchClientes();
24   };
25
26   return (
27     <div className="lista-clientes">
28       <h2>Clientes Registrados</h2>
29       <ul>
30         {clientes.map(cliente => (
31           <li key={cliente.id}>
32             <span><strong>{cliente.nombre}</strong></span>
33             <span>{cliente.correo}</span>
34             <span>{cliente.telefono}</span>
```

```
File Edit Selection View Go Run ...  
conexion.js package-lock.json ClienteForm.jsx ClientesList.jsx ClienteForm.css ClientesList.css App.css App.jsx  
Front > vite-project > src > components > ClientesList.jsx > ClientesList > clientes.map() callback  
5 const ClientesList = ({ onEdit, actualizar }) => {  
21 const eliminarCliente = async (id) => {  
22   await axios.delete(`http://localhost:3000/clientes/${id}`);  
23   fetchClientes();  
24 };  
25  
26 return (  
27   <div className="lista-clientes">  
28     <h2>Clientes Registrados</h2>  
29     <ul>  
30       {clientes.map(cliente => (  
31         <li key={cliente.id}>  
32           <span><strong>{cliente.nombre}</strong></span>  
33           <span>{cliente.correo}</span>  
34           <span>{cliente.telefono}</span>  
35           <span>{cliente.direccion}</span>  
36           <div className="acciones">  
37             <button onClick={() => onEdit(cliente)}>Editar</button>  
38             <button onClick={() => eliminarCliente(cliente.id)}>Eliminar</button>  
39           </div>  
40         </li>  
41       )>  
42     </ul>  
43   </div>  
44 );  
45 };  
46  
47 export default ClientesList;  
48
```

Ln 33, Col 42 Spaces: 2 UTF-8 CRLF JavaScript/JSX Go Live Background

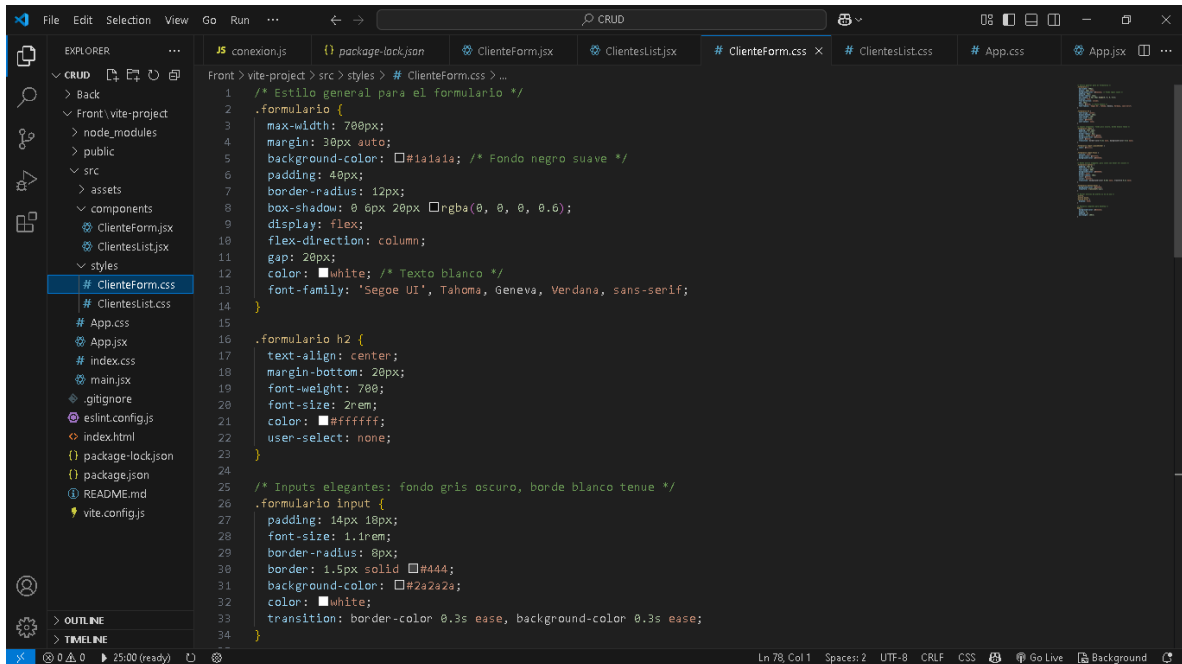
App.jsx

Este componente App actúa como el contenedor principal de la aplicación de gestión de clientes. Utiliza el hook `useState` para manejar dos estados: `clienteEditar`, que contiene la información del cliente que se desea editar, y actualizar, una bandera booleana que se usa para forzar la recarga de la lista de clientes tras una operación exitosa (como agregar o eliminar un cliente). Incluye dos funciones: `handleEdit`, que se activa al seleccionar un cliente para editar y guarda sus datos en el estado, y `handleSuccess`, que se llama después de una operación exitosa en el formulario y se encarga de recargar la lista y limpiar el formulario. Dentro del JSX, se renderizan dos componentes hijos: `ClienteForm`, que permite agregar o editar clientes y recibe como props el cliente a editar y la función de éxito; y `ClientesList`, que muestra la lista de clientes registrados y recibe como props la función para activar la edición y la bandera de actualización. Todo está envuelto en un contenedor estilizado con una clase CSS. Así, este componente coordina el flujo de datos y eventos entre el formulario y la lista de clientes, funcionando como el núcleo de la aplicación.

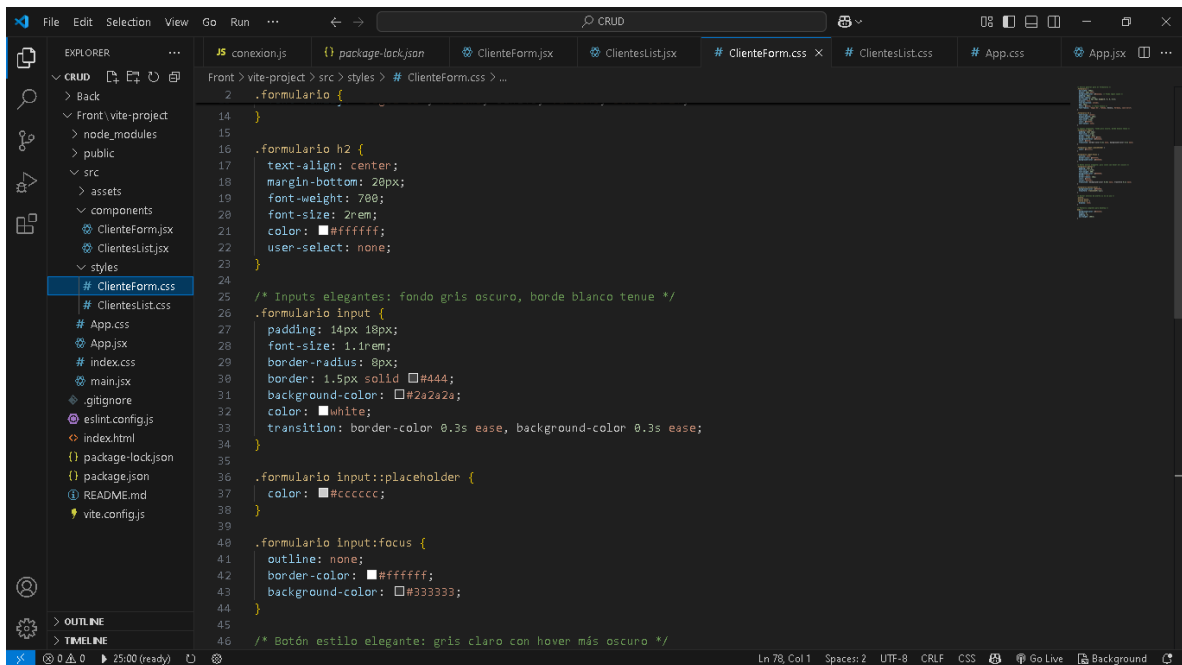


```
1 import React, { useState } from 'react';
2 import ClienteForm from './components/ClienteForm';
3 import ClientesList from './components/ClientesList';
4 import './App.css';
5
6 const App = () => {
7   const [clienteEditar, setClienteEditar] = useState(null);
8   const [actualizar, setActualizar] = useState(false);
9
10  const handleEdit = (cliente) => {
11    setClienteEditar(cliente);
12  };
13
14  const handleSuccess = () => {
15    setActualizar(!actualizar);
16    setClienteEditar(null);
17  };
18
19  return (
20    <div className="container">
21      <h1>Gestión de clientes</h1>
22
23      <ClienteForm clienteEditar={clienteEditar} onSuccess={handleSuccess} />
24      <ClientesList onEdit={handleEdit} actualizar={actualizar} />
25    </div>
26  );
27 };
28
29 export default App;
```

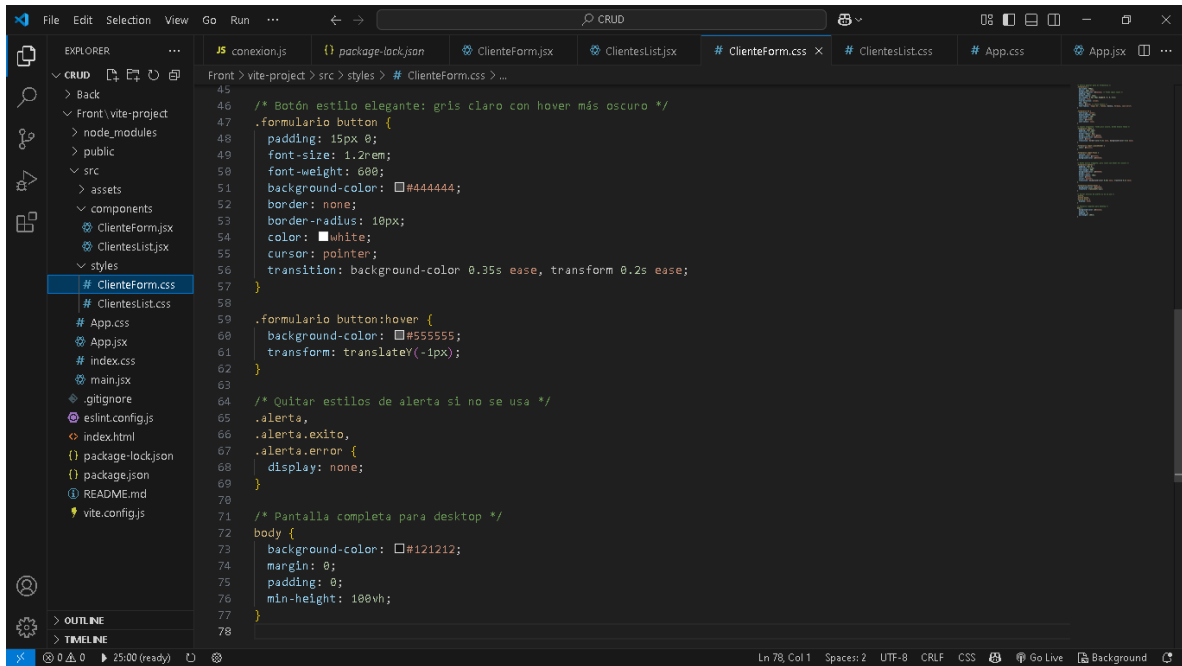
Se implemento tambien la carpeta Styles para darles estilos a la lista, formularios y a la app. Clientes forms.css



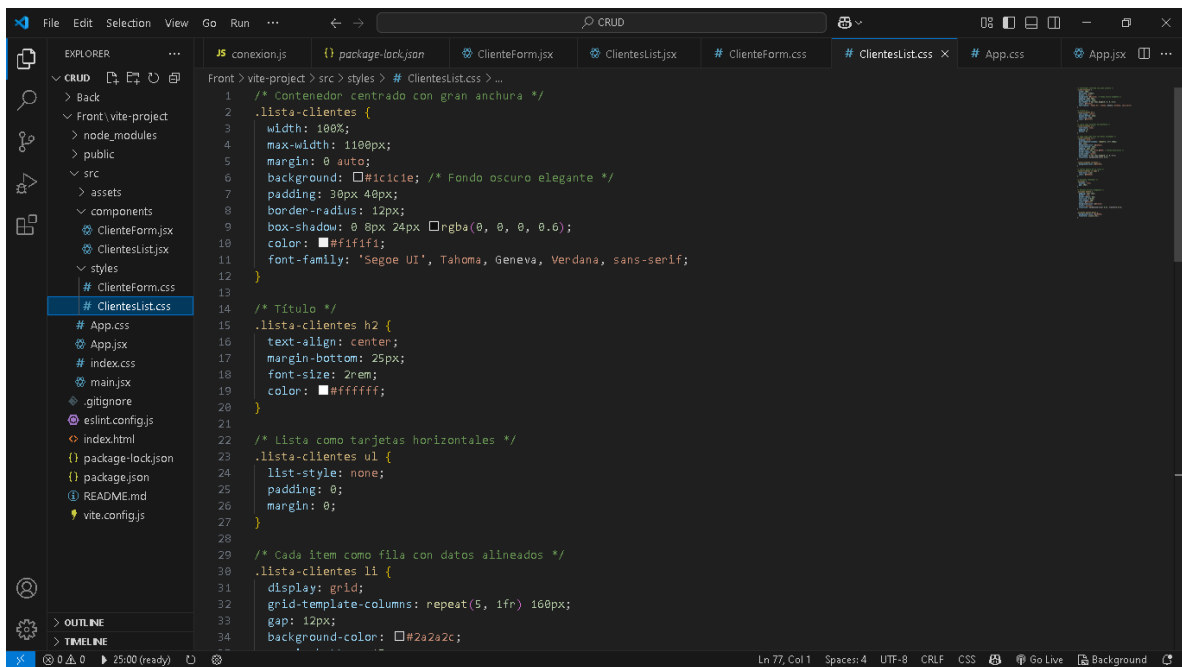
```
Front > vite-project > src > styles > # ClienteForm.css > ...
1  /* Estilo general para el formulario */
2  .formulario {
3      max-width: 700px;
4      margin: 30px auto;
5      background-color: #1a1a1a; /* Fondo negro suave */
6      padding: 40px;
7      border-radius: 12px;
8      box-shadow: 0 6px 20px rgba(0, 0, 0, 0.6);
9      display: flex;
10     flex-direction: column;
11     gap: 20px;
12     color: white; /* Texto blanco */
13     font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
14 }
15
16 .formulario h2 {
17     text-align: center;
18     margin-bottom: 20px;
19     font-weight: 700;
20     font-size: 2rem;
21     color: #ffffff;
22     user-select: none;
23 }
24
25 /* Inputs elegantes: fondo gris oscuro, borde blanco tenue */
26 .formulario input {
27     padding: 14px 18px;
28     font-size: 1.1rem;
29     border-radius: 8px;
30     border: 1.5px solid #444;
31     background-color: #2a2a2a;
32     color: white;
33     transition: border-color 0.3s ease, background-color 0.3s ease;
34 }
```



```
Front > vite-project > src > styles > # ClienteForm.css > ...
14 }
15
16 .formulario h2 {
17     text-align: center;
18     margin-bottom: 20px;
19     font-weight: 700;
20     font-size: 2rem;
21     color: #ffffff;
22     user-select: none;
23 }
24
25 /* Inputs elegantes: fondo gris oscuro, borde blanco tenue */
26 .formulario input {
27     padding: 14px 18px;
28     font-size: 1.1rem;
29     border-radius: 8px;
30     border: 1.5px solid #444;
31     background-color: #2a2a2a;
32     color: white;
33     transition: border-color 0.3s ease, background-color 0.3s ease;
34 }
35
36 .formulario input::placeholder {
37     color: #cccccc;
38 }
39
40 .formulario input:focus {
41     outline: none;
42     border-color: #ffffff;
43     background-color: #333333;
44 }
45
46 /* Botón estilo elegante: gris claro con hover más oscuro */
```



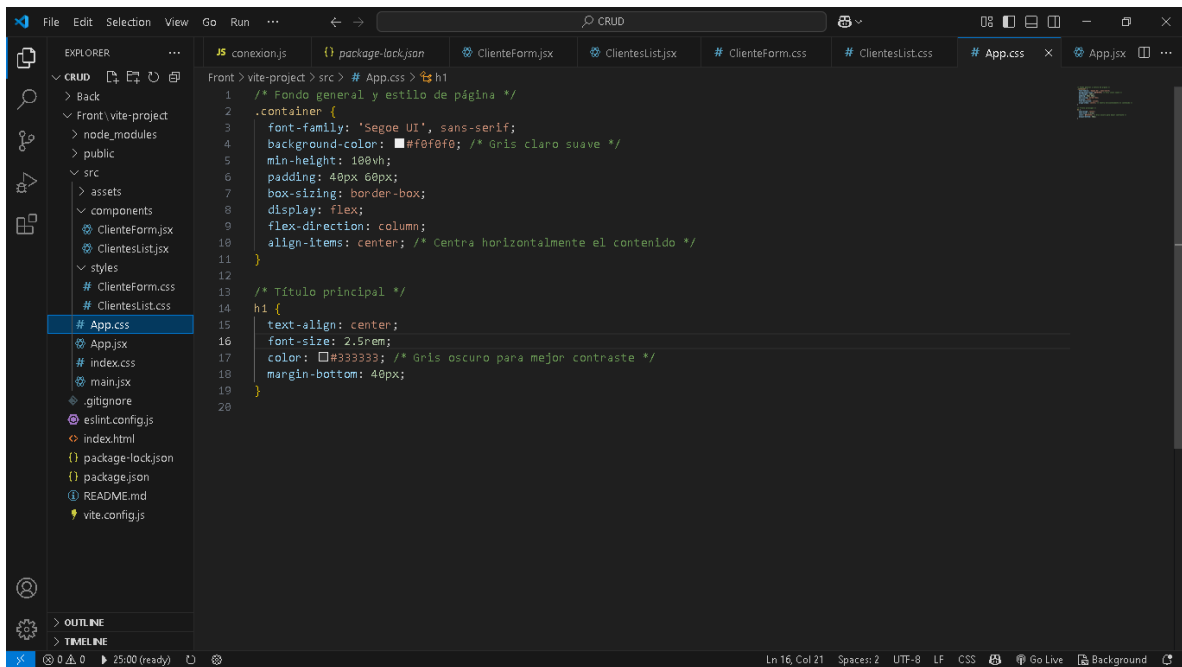
ClienteList.CSS



```
Front > vite-project > src > styles > # ClientesList.css > ...
13
14 /* Título */
15 .lista-clientes h2 {
16   text-align: center;
17   margin-bottom: 25px;
18   font-size: 2rem;
19   color: #ffffff;
20 }
21
22 /* Lista como tarjetas horizontales */
23 .lista-clientes ul {
24   list-style: none;
25   padding: 0;
26   margin: 0;
27 }
28
29 /* Cada item como fila con datos alineados */
30 .lista-clientes li {
31   display: grid;
32   grid-template-columns: repeat(5, 1fr) 160px;
33   gap: 12px;
34   background-color: #2a2a2c;
35   margin-bottom: 15px;
36   padding: 10px 22px;
37   border-left: 6px solid #444; /* Borde decorativo */
38   border-radius: 10px;
39   align-items: center;
40   box-shadow: 0 4px 12px rgba(0, 0, 0, 0.4);
41   transition: background-color 0.3s;
42 }
43
44 .lista-clientes li:hover {
45   background-color: #333335;
46 }
```

```
Front > vite-project > src > styles > # ClientesList.css > ...
74
75 /* Acciones (botones) */
76 .acciones {
77   display: flex;
78   gap: 10px;
79 }
80
81 /* Botones oscuros elegantes */
82 .acciones button {
83   padding: 10px 18px;
84   border: none;
85   border-radius: 6px;
86   font-size: 0.9rem;
87   font-weight: 600;
88   color: #ffffff;
89   background-color: #3c3c3e;
90   cursor: pointer;
91   transition: background-color 0.3s, transform 0.2s;
92 }
93
94 .acciones button:hover {
95   background-color: #505052;
96   transform: scale(1.02);
97 }
```

App.CSS



The image shows a screenshot of the Visual Studio Code editor interface. The Explorer panel on the left shows a project structure for 'crud' with folders like 'Front', 'node_modules', 'public', and 'src'. The 'App.css' file is selected in the Explorer. The main editor area displays the content of 'App.css', which includes a container style and a main heading style. The status bar at the bottom indicates the current line and column (Ln 16, Col 21) and the active language (CSS).

```
Front > vite-project > src > # App.css > h1
1  /* Fondo general y estilo de página */
2  .container {
3      font-family: 'Segoe UI', sans-serif;
4      background-color: #f0f0f0; /* Gris claro suave */
5      min-height: 100vh;
6      padding: 40px 60px;
7      box-sizing: border-box;
8      display: flex;
9      flex-direction: column;
10     align-items: center; /* Centra horizontalmente el contenido */
11 }
12
13 /* Título principal */
14 h1 {
15     text-align: center;
16     font-size: 2.5rem;
17     color: #333333; /* Gris oscuro para mejor contraste */
18     margin-bottom: 40px;
19 }
20
```

Como se ve de manera visual.

Gestión de Clientes

Agregar Nuevo Cliente

Nombre

Correo

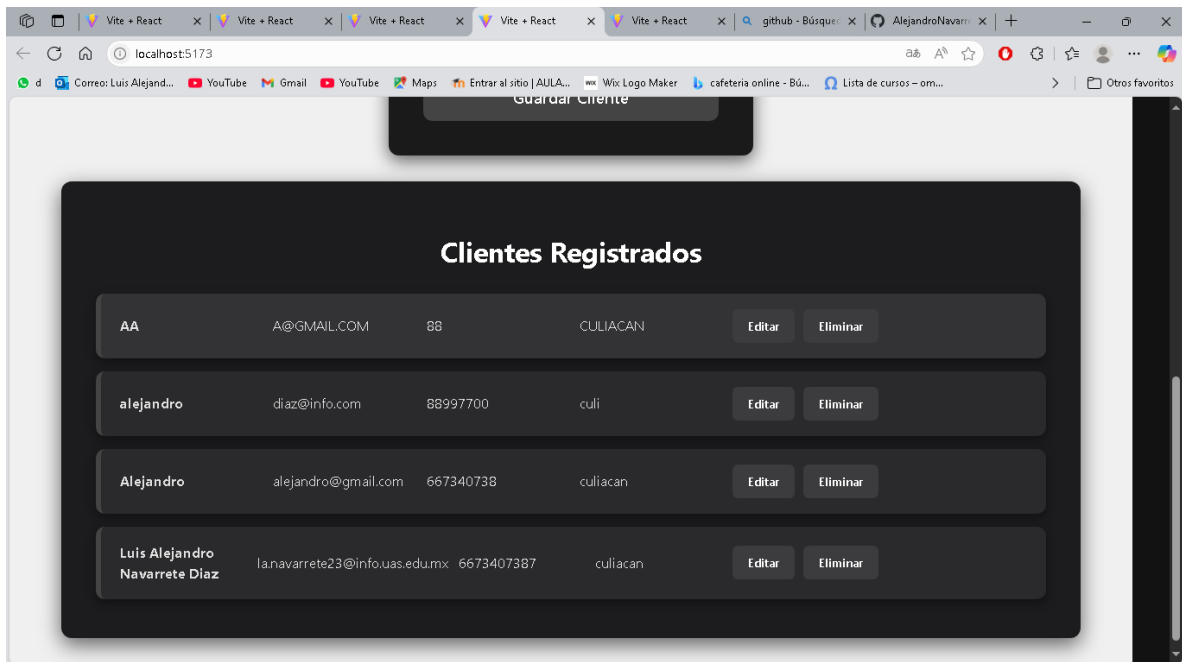
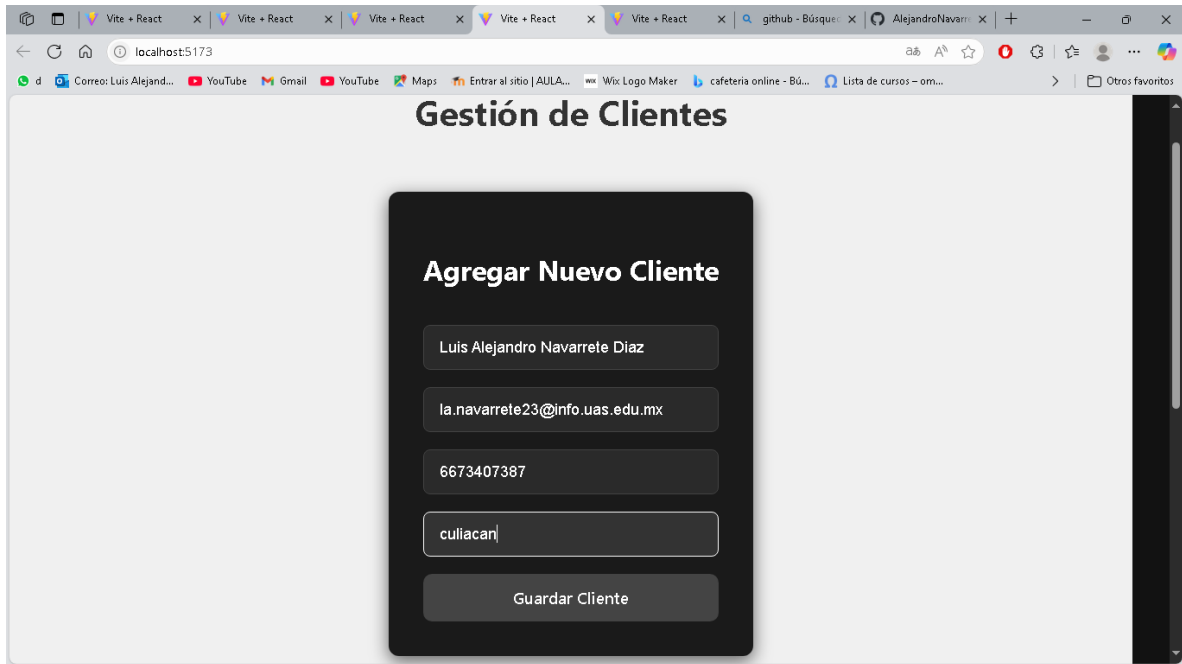
Teléfono

Dirección

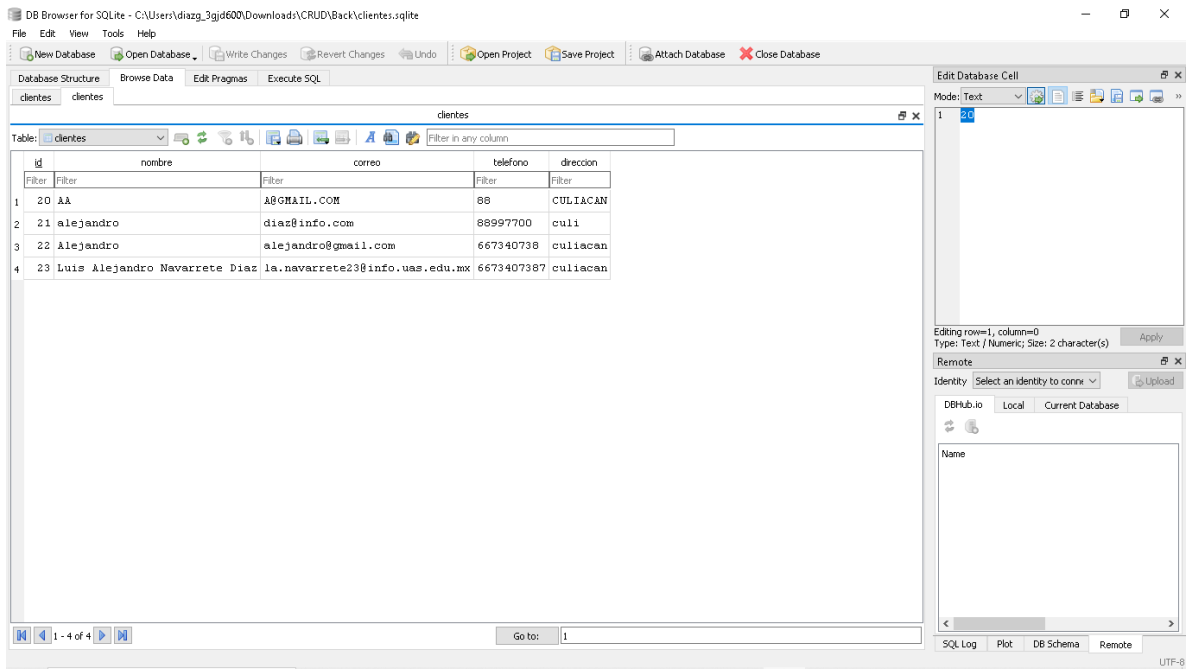
Guardar Cliente

Clientes Registrados

| | | | | | |
|-----------|--------------------|-------------|----------|--------|----------|
| alejandro | diaz@info.com | 66734073867 | culiacan | Editar | Eliminar |
| Maria | margarita@info.com | 6673408321 | culiacan | Editar | Eliminar |
| LUIS | LUIS@INFO.UAS.COM | 6675009988 | CULIACAN | Editar | Eliminar |
| AA | A@GMAIL.COM | 88 | CULIACAN | Editar | Eliminar |
| alejandro | diaz@info.com | 88997700 | culi | Editar | Eliminar |



Base de datos de como se ve el dato insertado.



Como se inserta.

