



Universidad Autónoma de Sinaloa

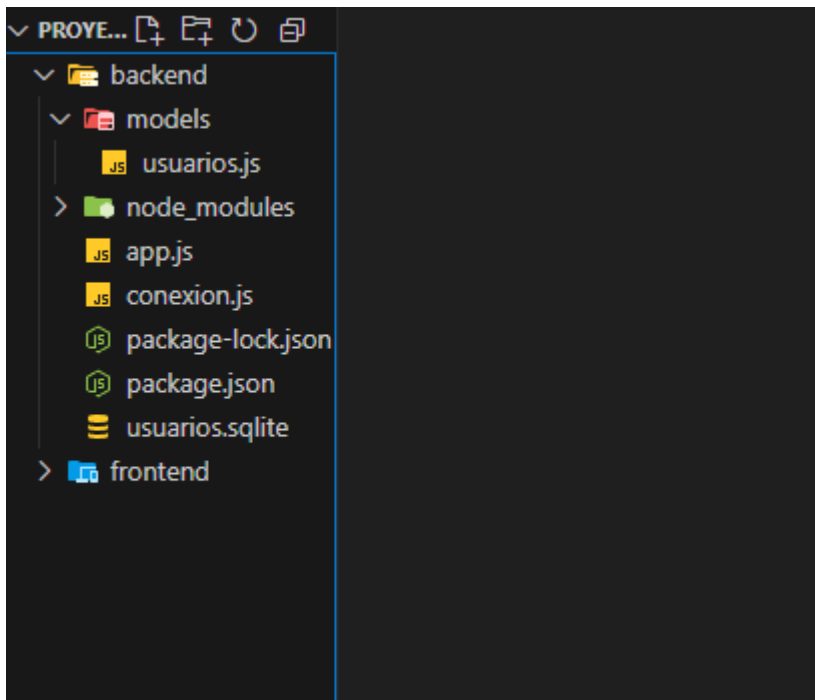
Materia: Desarrollo web del lado del
servidor

Tema: Actividad final: CRUD
completo

Docente: José Manuel Cazarez
Alderete

Alumno: Uribe Quintero Maximiliano
2-3

Código del Backend

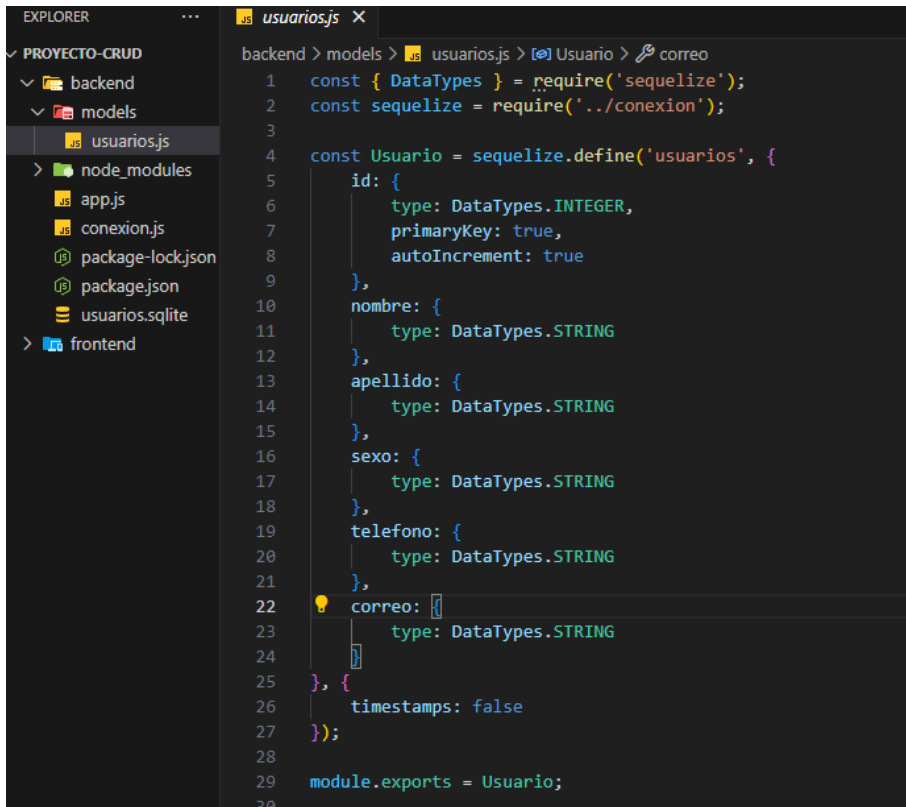


Aquí se encuentra todo el código y recursos que se usó para la elaboración y funcionamiento del Backend para el CRUD de usuarios, y que de acuerdo a las indicaciones se desarrollado mediante Node.js, Express y Sequelize con la base de datos “DB Browser for SQLite”.

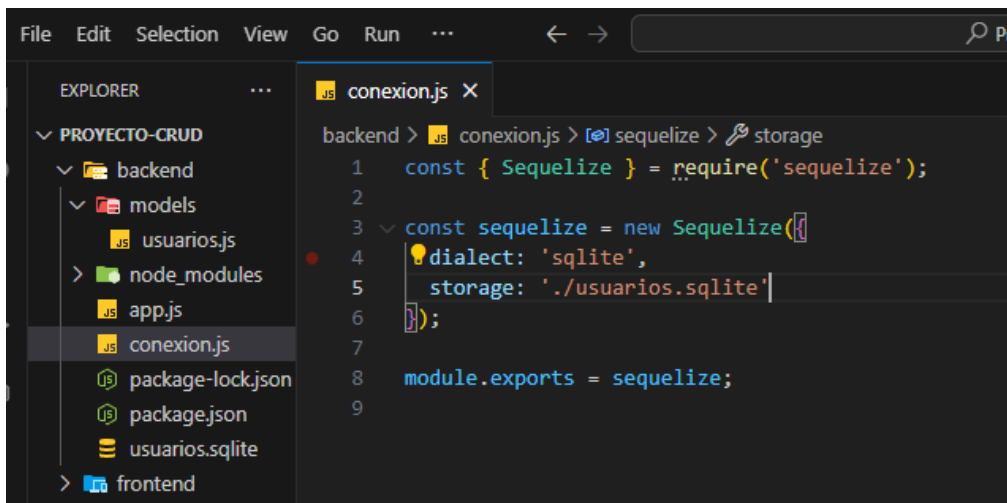
Este Backend proporciona una API REST que permite gestionar registros de usuarios a través de operaciones crear, leer, actualizar y eliminar (CRUD).

Este Backend sirve como base sólida y escalable para conectar el FRONTEND con la lógica de persistencia de datos, garantizando así una comunicación fluida entre la interfaz de usuario y la base de datos.

Modelo y Conexión de la BD “usuarios.js”, “conexión.js”



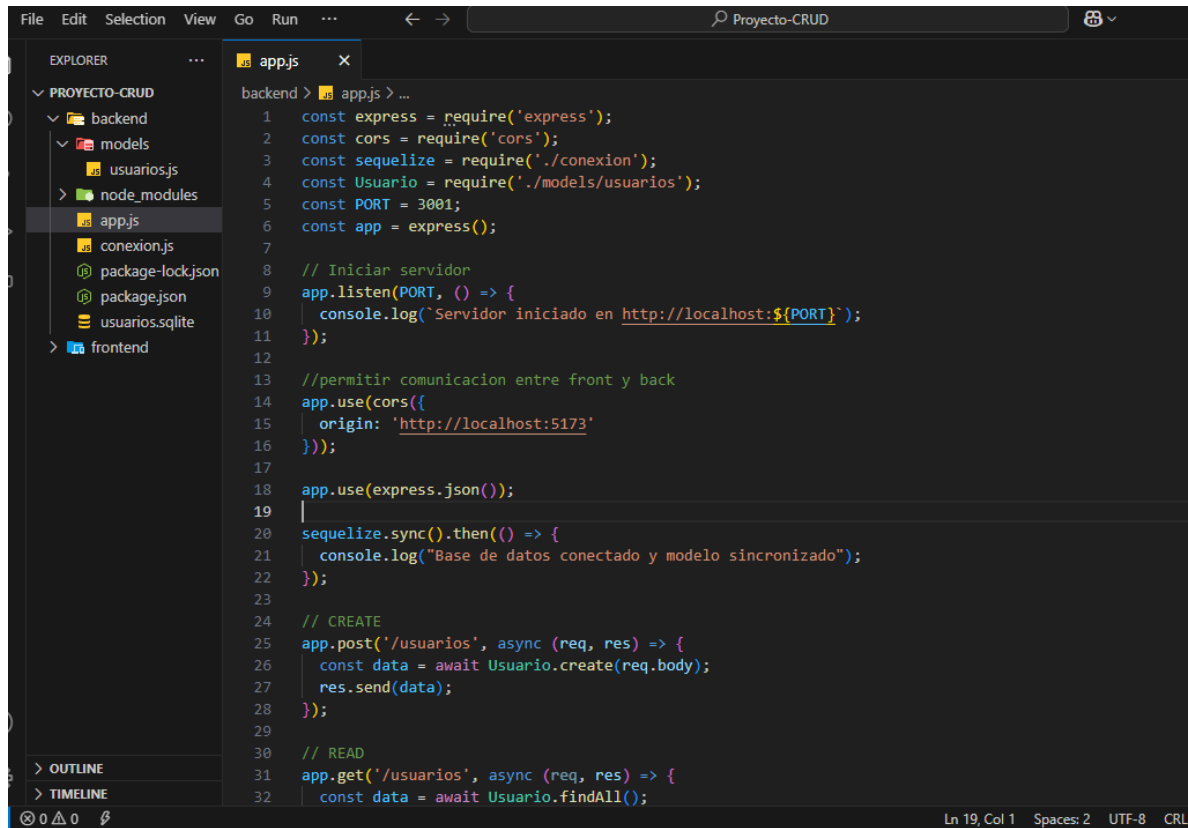
```
1 const { DataTypes } = require('sequelize');
2 const sequelize = require('../conexion');
3
4 const Usuario = sequelize.define('usuarios', {
5   id: {
6     type: DataTypes.INTEGER,
7     primaryKey: true,
8     autoIncrement: true
9   },
10  nombre: {
11    type: DataTypes.STRING
12  },
13  apellido: {
14    type: DataTypes.STRING
15  },
16  sexo: {
17    type: DataTypes.STRING
18  },
19  telefono: {
20    type: DataTypes.STRING
21  },
22  correo: {
23    type: DataTypes.STRING
24  },
25 }, {
26   timestamps: false
27 });
28
29 module.exports = Usuario;
```



```
1 const { Sequelize } = require('sequelize');
2
3 const sequelize = new Sequelize({
4   dialect: 'sqlite',
5   storage: './usuarios.sqlite'
6 });
7
8 module.exports = sequelize;
```

En estas partes se implementó un modelo de datos llamado usuarios, el cual define la estructura de los registros que serán almacenados en la base de datos. La conexión a la base de datos y su sincronización se realizó mediante Sequelize, lo que facilita la gestión y el uso de una base de datos SQLite.

Backend - App.js



```
File Edit Selection View Go Run ... < -> Proyecto-CRUD

EXPLORER
  PROJECTO-CRUD
    backend
    models
      usuarios.js
    node_modules
    app.js
    conexion.js
    package-lock.json
    package.json
    usuarios.sqlite
    frontend

  OUTLINE
  TIMELINE

backend > app.js > ...
1  const express = require('express');
2  const cors = require('cors');
3  const sequelize = require('./conexion');
4  const Usuario = require('./models/usuarios');
5  const PORT = 3001;
6  const app = express();
7
8  // Iniciar servidor
9  app.listen(PORT, () => {
10   console.log(`Servidor iniciado en http://localhost:${PORT}`);
11 });
12
13 //permitir comunicacion entre front y back
14 app.use(cors({
15   origin: 'http://localhost:5173'
16 }));
17
18 app.use(express.json());
19
20 sequelize.sync().then(() => {
21   console.log("Base de datos conectado y modelo sincronizado");
22 });
23
24 // CREATE
25 app.post('/usuarios', async (req, res) => {
26   const data = await Usuario.create(req.body);
27   res.send(data);
28 });
29
30 // READ
31 app.get('/usuarios', async (req, res) => {
32   const data = await Usuario.findAll();
```

El archivo app.js del Backend contiene el código principal del servidor, este fue desarrollado durante clases así que se tomó de ejemplo, y fue necesario para manejar todas las operaciones del CRUD. A través de este archivo se configuró la conexión con la base de datos mediante Sequelize, se definieron las rutas del API y se habilitó CORS para aceptar peticiones desde el navegador.

Base de Datos implementada “usuarios. sqlite”

Edit table definition

Table

usuarios

Advanced

FieldsIndex ConstraintsForeign KeysCheck Constraints

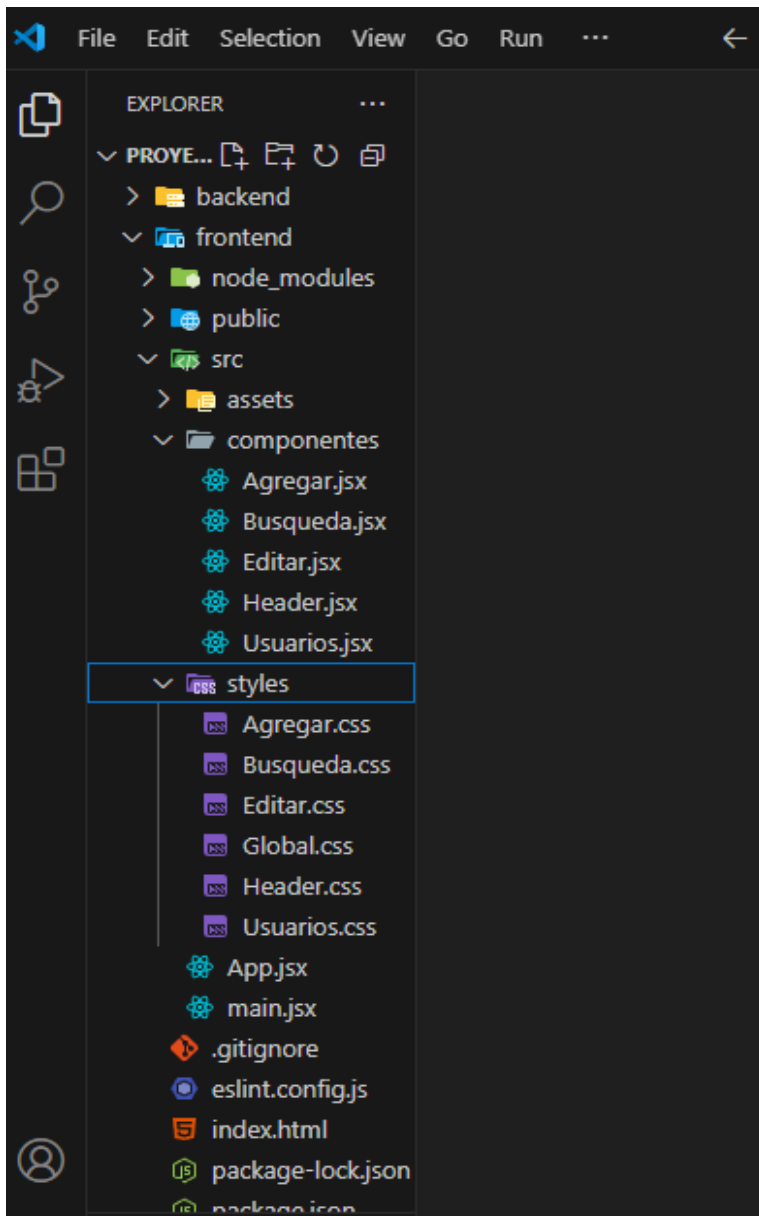
AddRemoveMove to topMove upMove downMove to bottom

Name	Type	NN	PK	AI	U	Default	Check	Collat
id	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
nombre	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
apellido	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
sexo	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
telefono	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
correo	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			

```
1 CREATE TABLE "usuarios" (  
2     "id"    INTEGER,  
3     "nombre" TEXT NOT NULL,  
4     "apellido" TEXT NOT NULL,  
5     "sexo" TEXT NOT NULL,  
6     "telefono" TEXT NOT NULL,  
7     "correo" TEXT NOT NULL,  
8     PRIMARY KEY("id" AUTOINCREMENT)  
9 );
```

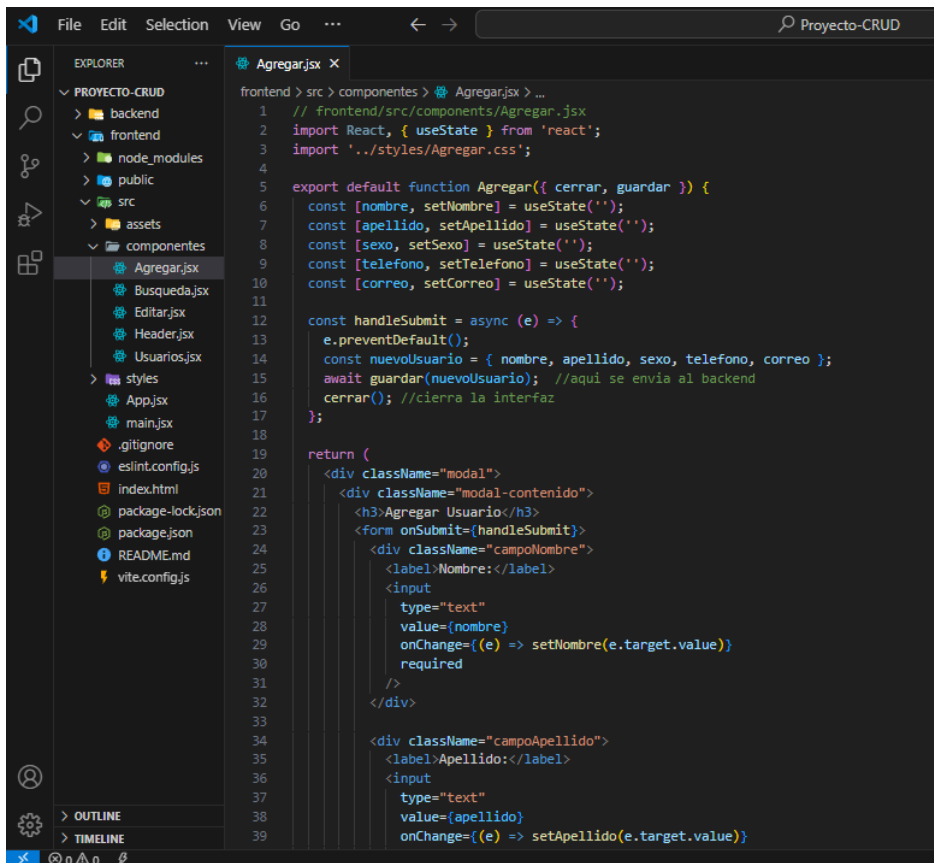
Su importancia radica en que almacena todos los usuarios registrados por el FRONTEND, permitiendo guardar datos persistentes sin depender de un sistema de base de datos externo como MySQL o PostgreSQL. Esto hace que el proyecto sea fácil de portar y ejecutar en otros entornos sin configuraciones complicadas.

Código del Frontend



Aquí se encuentra todo el código y recursos que se usó para la interfaz o página del Front este fue desarrollado utilizando React como biblioteca principal para hacer la interfaz de usuario. Y pues este Front proporciona una vista interactiva para lo que es registrar un usuario, hacer la búsqueda de un usuario, editar y eliminar usuarios de forma sencilla. Y través de componentes que se usaron pues se gestiona todas las interacciones con el Backend mediante solicitudes.

Componente “Agregar.jsx”

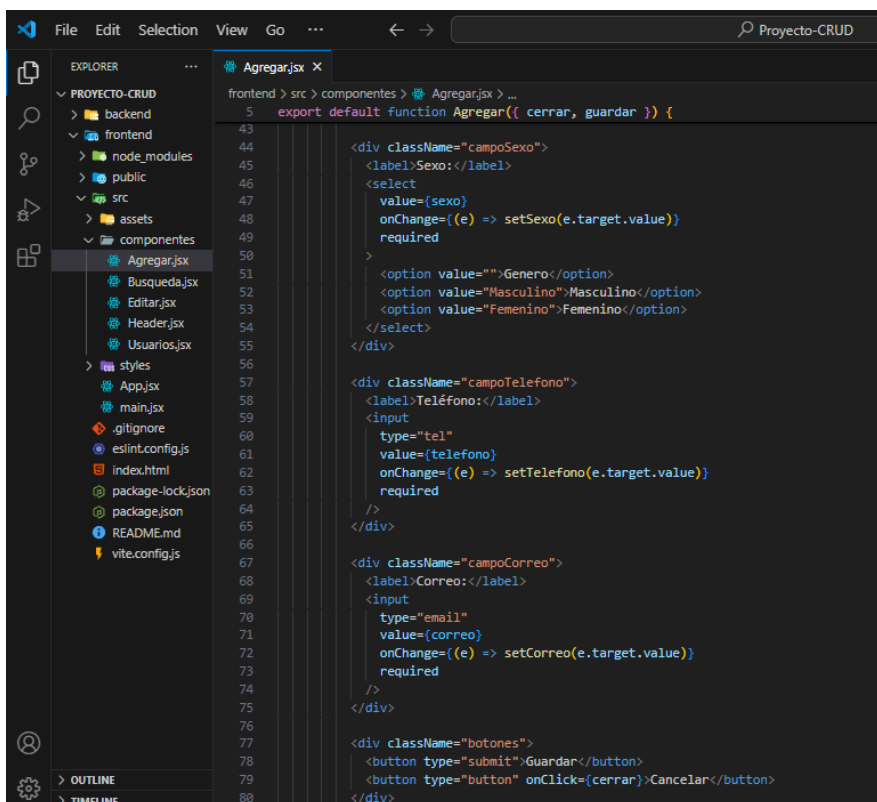


```
File Edit Selection View Go ... Proyecto-CRUD

EXPLORER
  PROJECTO-CRUD
    backend
    frontend
    node_modules
    public
    src
      assets
      componentes
        Agregar.jsx
        Busqueda.jsx
        Editar.jsx
        Header.jsx
        Usuarios.jsx
      styles
        App.jsx
        main.jsx
      .gitignore
      eslint.config.js
      index.html
      package-lock.json
      package.json
      README.md
      vite.config.js

  OUTLINE
  TIMELINE

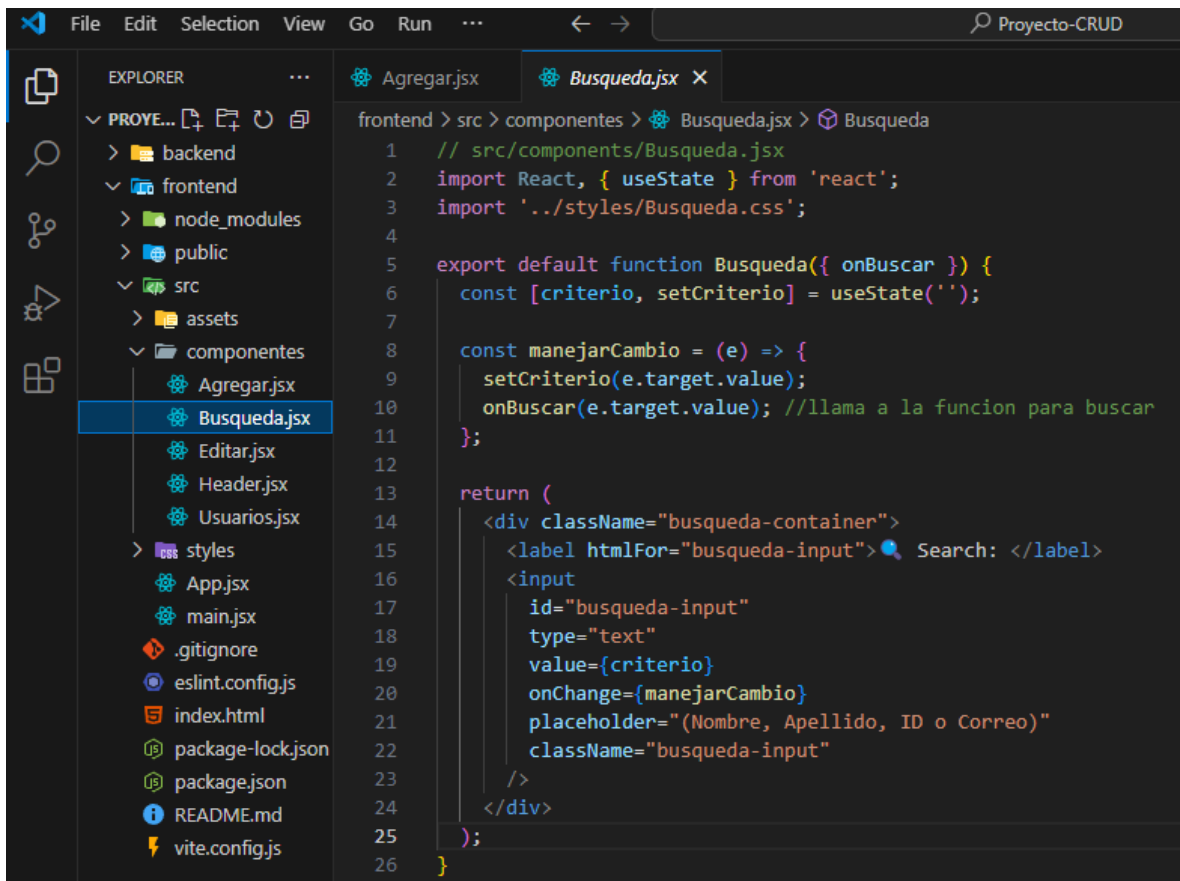
frontend > src > componentes > Agregar.jsx > ...
1 // frontend/src/componentes/Agregar.jsx
2 import React, { useState } from 'react';
3 import '../styles/Agregar.css';
4
5 export default function Agregar({ cerrar, guardar }) {
6   const [nombre, setNombre] = useState('');
7   const [apellido, setApellido] = useState('');
8   const [sexo, setSexo] = useState('');
9   const [telefono, setTelefono] = useState('');
10  const [correo, setCorreo] = useState('');
11
12  const handleSubmit = async (e) => {
13    e.preventDefault();
14    const nuevoUsuario = { nombre, apellido, sexo, telefono, correo };
15    await guardar(nuevoUsuario); //aquí se envia al backend
16    cerrar(); //cierra la interfaz
17  };
18
19  return (
20    <div className="modal">
21      <div className="modal-contenido">
22        <h3>Agregar Usuario</h3>
23        <form onSubmit={handleSubmit}>
24          <div className="campoNombre">
25            <label>Nombre:</label>
26            <input
27              type="text"
28              value={nombre}
29              onChange={(e) => setNombre(e.target.value)}
30              required
31            />
32          </div>
33
34          <div className="campoApellido">
35            <label>Apellido:</label>
36            <input
37              type="text"
38              value={apellido}
39              onChange={(e) => setApellido(e.target.value)}
40            />
41          </div>
42
43          <div className="campoSexo">
44            <label>Sexo:</label>
45            <select
46              value={sexo}
47              onChange={(e) => setSexo(e.target.value)}
48              required
49            >
50              <option value="">Genero</option>
51              <option value="Masculino">Masculino</option>
52              <option value="Femenino">Femenino</option>
53            </select>
54          </div>
55
56          <div className="campoTelefono">
57            <label>Teléfono:</label>
58            <input
59              type="tel"
60              value={telefono}
61              onChange={(e) => setTelefono(e.target.value)}
62              required
63            />
64          </div>
65
66          <div className="campoCorreo">
67            <label>Correo:</label>
68            <input
69              type="email"
70              value={correo}
71              onChange={(e) => setCorreo(e.target.value)}
72              required
73            />
74          </div>
75
76          <div className="botones">
77            <button type="submit">Guardar</button>
78            <button type="button" onClick={cerrar}>Cancelar</button>
79          </div>
80        </form>
81      </div>
82    </div>
83  );
84}
```



```
5 export default function Agregar({ cerrar, guardar }) {
43
44   <div className="campoSexo">
45     <label>Sexo:</label>
46     <select
47       value={sexo}
48       onChange={(e) => setSexo(e.target.value)}
49       required
50     >
51       <option value="">Genero</option>
52       <option value="Masculino">Masculino</option>
53       <option value="Femenino">Femenino</option>
54     </select>
55   </div>
56
57   <div className="campoTelefono">
58     <label>Teléfono:</label>
59     <input
60       type="tel"
61       value={telefono}
62       onChange={(e) => setTelefono(e.target.value)}
63       required
64     />
65   </div>
66
67   <div className="campoCorreo">
68     <label>Correo:</label>
69     <input
70       type="email"
71       value={correo}
72       onChange={(e) => setCorreo(e.target.value)}
73       required
74     />
75   </div>
76
77   <div className="botones">
78     <button type="submit">Guardar</button>
79     <button type="button" onClick={cerrar}>Cancelar</button>
80   </div>
81 </form>
82 </div>
83 </div>
84 }
```

Este componente se encarga de mostrar un formulario emergente o una interfaz desplegable que permite registrar nuevos datos personales del usuario para registrarlo. Utiliza `useState` para manejar el estado de cada campo del formulario y, al enviarlo, llama a una función guardar que comunica estos datos al Backend mediante una solicitud. Además, permite cerrar la interfaz mediante la función `cerrar`. Su propósito es la creación de nuevos usuarios, asegurando que la información sea enviada correctamente a la base de datos.

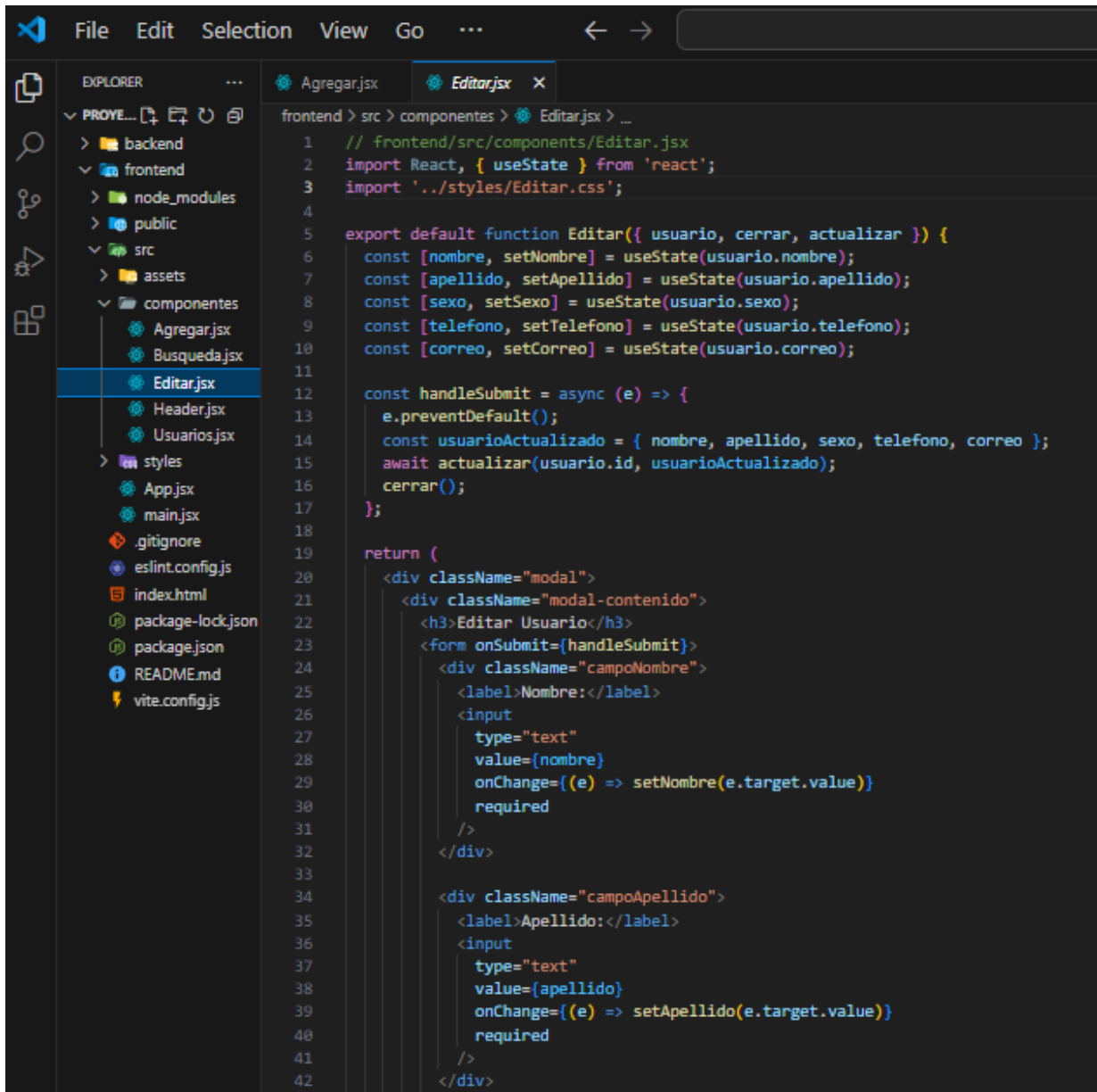
Componente “Busqueda.jsx”



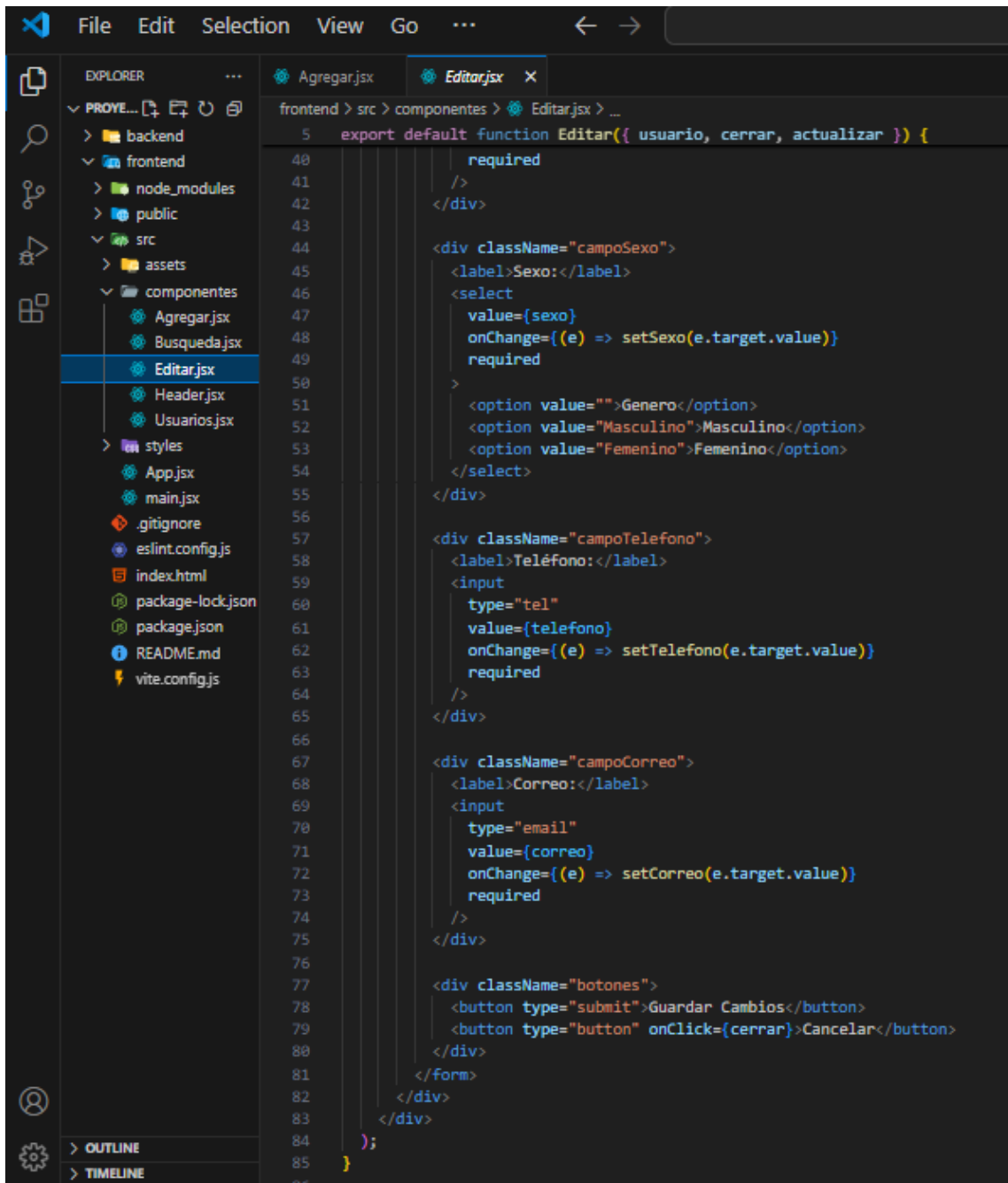
```
1 // src/components/Busqueda.jsx
2 import React, { useState } from 'react';
3 import '../styles/Busqueda.css';
4
5 export default function Busqueda({ onBuscar }) {
6   const [criterio, setCriterio] = useState('');
7
8   const manejarCambio = (e) => {
9     setCriterio(e.target.value);
10    onBuscar(e.target.value); //llama a la funcion para buscar
11  };
12
13  return (
14    <div className="busqueda-container">
15      <label htmlFor="busqueda-input"> Search: </label>
16      <input
17        id="busqueda-input"
18        type="text"
19        value={criterio}
20        onChange={manejarCambio}
21        placeholder="(Nombre, Apellido, ID o Correo)"
22        className="busqueda-input"
23      />
24    </div>
25  );
26 }
```

El componente `Busqueda.jsx` hace la función de permitir filtrar y buscar registros dentro de la tabla de usuarios del front mediante la BD. Utiliza el estado `criterio` para capturar el texto ingresado por el usuario y llama a la función designada cada vez que el valor del input cambia. Y pues esto hace una búsqueda en tiempo real que facilita la localización del usuario ya sea por nombre, apellido, ID o correo.

Componente “Editar.jsx”



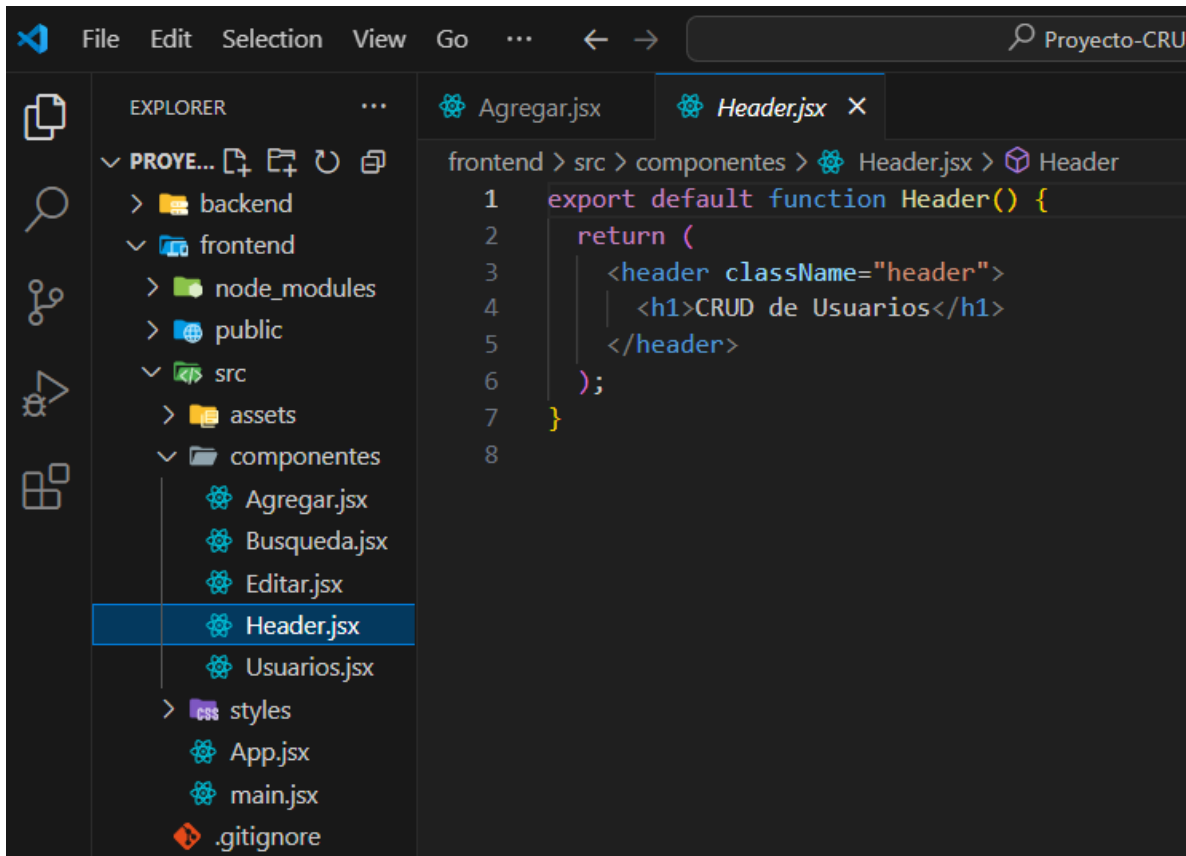
```
1 // frontend/src/components/Editar.jsx
2 import React, { useState } from 'react';
3 import '../styles/Editar.css';
4
5 export default function Editar({ usuario, cerrar, actualizar }) {
6   const [nombre, setNombre] = useState(usuario.nombre);
7   const [apellido, setApellido] = useState(usuario.apellido);
8   const [sexo, setSexo] = useState(usuario.sexo);
9   const [telefono, setTelefono] = useState(usuario.telefono);
10  const [correo, setCorreo] = useState(usuario.correo);
11
12  const handleSubmit = async (e) => {
13    e.preventDefault();
14    const usuarioActualizado = { nombre, apellido, sexo, telefono, correo };
15    await actualizar(usuario.id, usuarioActualizado);
16    cerrar();
17  };
18
19  return (
20    <div className="modal">
21      <div className="modal-contenido">
22        <h3>Editar Usuario</h3>
23        <form onSubmit={handleSubmit}>
24          <div className="campoNombre">
25            <label>Nombre:</label>
26            <input
27              type="text"
28              value={nombre}
29              onChange={(e) => setNombre(e.target.value)}
30              required
31            />
32          </div>
33
34          <div className="campoApellido">
35            <label>Apellido:</label>
36            <input
37              type="text"
38              value={apellido}
39              onChange={(e) => setApellido(e.target.value)}
40              required
41            />
42          </div>
```



```
5 export default function Editar({ usuario, cerrar, actualizar }) {
40   required
41   />
42 </div>
43
44   <div className="campoSexo">
45     <label>Sexo:</label>
46     <select
47       value={sexo}
48       onChange={(e) => setSexo(e.target.value)}
49       required
50     >
51       <option value="">Genero</option>
52       <option value="Masculino">Masculino</option>
53       <option value="Femenino">Femenino</option>
54     </select>
55   </div>
56
57   <div className="campoTelefono">
58     <label>Teléfono:</label>
59     <input
60       type="tel"
61       value={telefono}
62       onChange={(e) => setTelefono(e.target.value)}
63       required
64     />
65   </div>
66
67   <div className="campoCorreo">
68     <label>Correo:</label>
69     <input
70       type="email"
71       value={correo}
72       onChange={(e) => setCorreo(e.target.value)}
73       required
74     />
75   </div>
76
77   <div className="botones">
78     <button type="submit">Guardar Cambios</button>
79     <button type="button" onClick={cerrar}>Cancelar</button>
80   </div>
81 </form>
82 </div>
83 </div>
84 );
85 }
```

En Editar.jsx se buscó para que fuera la opción de poder modificar los datos de un usuario ya existente dentro del sistema. Ósea este componente se encarga de mostrar una interfaz emergente con los campos prellenados del usuario seleccionado, y al cambiar y guardar los cambios, se envía una solicitud de actualización al Backend, lo que resulta esencial para la gestión eficiente de los datos en el CRUD. Su diseño fue o se inspiró en el de Agregar.jsx.

Componente “Header.jsx”

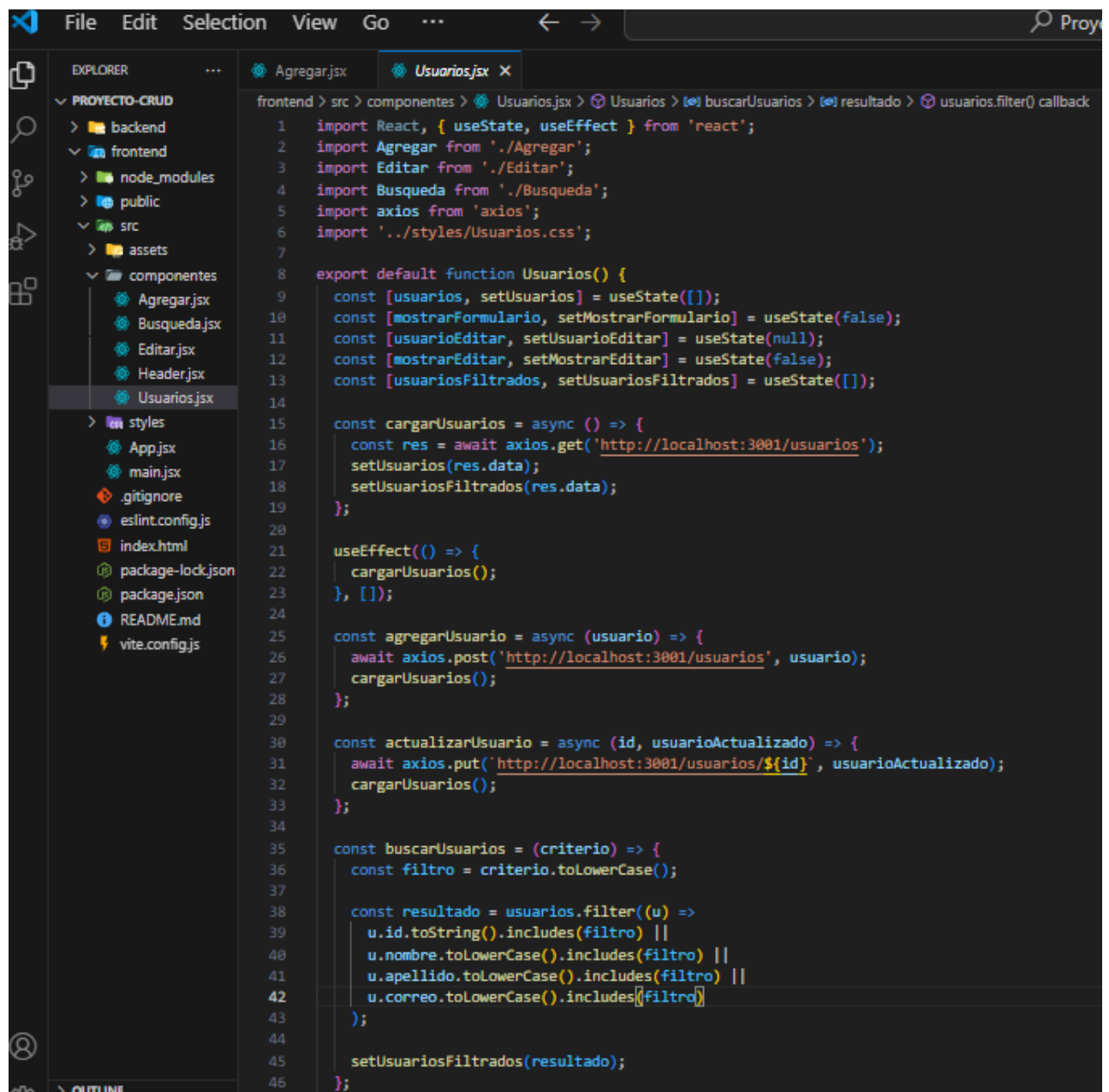


The screenshot shows the Visual Studio Code editor with the Explorer sidebar on the left and the Editor pane on the right. The Explorer sidebar shows the project structure with the 'Header.jsx' file selected under the 'componentes' folder. The Editor pane displays the code for 'Header.jsx'.

```
1 export default function Header() {  
2   return (  
3     <header className="header">  
4       <h1>CRUD de Usuarios</h1>  
5     </header>  
6   );  
7 }  
8
```

Este es más como un componente para diseño, ya que no aporta mucho, pero es para mejorar la presentación del diseño de la pag web.

Componente “Usuarios.jsx”



```
1  import React, { useState, useEffect } from 'react';
2  import Agregar from './Agregar';
3  import Editar from './Editar';
4  import Busqueda from './Busqueda';
5  import axios from 'axios';
6  import '../styles/Usuarios.css';
7
8  export default function Usuarios() {
9    const [usuarios, setUsuarios] = useState([]);
10   const [mostrarFormulario, setMostrarFormulario] = useState(false);
11   const [usuarioEditar, setUsuarioEditar] = useState(null);
12   const [mostrarEditar, setMostrarEditar] = useState(false);
13   const [usuariosFiltrados, setUsuariosFiltrados] = useState([]);
14
15   const cargarUsuarios = async () => {
16     const res = await axios.get('http://localhost:3001/usuarios');
17     setUsuarios(res.data);
18     setUsuariosFiltrados(res.data);
19   };
20
21   useEffect(() => {
22     cargarUsuarios();
23   }, []);
24
25   const agregarUsuario = async (usuario) => {
26     await axios.post('http://localhost:3001/usuarios', usuario);
27     cargarUsuarios();
28   };
29
30   const actualizarUsuario = async (id, usuarioActualizado) => {
31     await axios.put(`http://localhost:3001/usuarios/${id}`, usuarioActualizado);
32     cargarUsuarios();
33   };
34
35   const buscarUsuarios = (criterio) => {
36     const filtro = criterio.toLowerCase();
37
38     const resultado = usuarios.filter(u =>
39       u.id.toString().includes(filtro) ||
40       u.nombre.toLowerCase().includes(filtro) ||
41       u.apellido.toLowerCase().includes(filtro) ||
42       u.correo.toLowerCase().includes(filtro)
43     );
44
45     setUsuariosFiltrados(resultado);
46   };
47 }
```

File Edit Selection View Go ... Proyecto-CR

EXPLORER

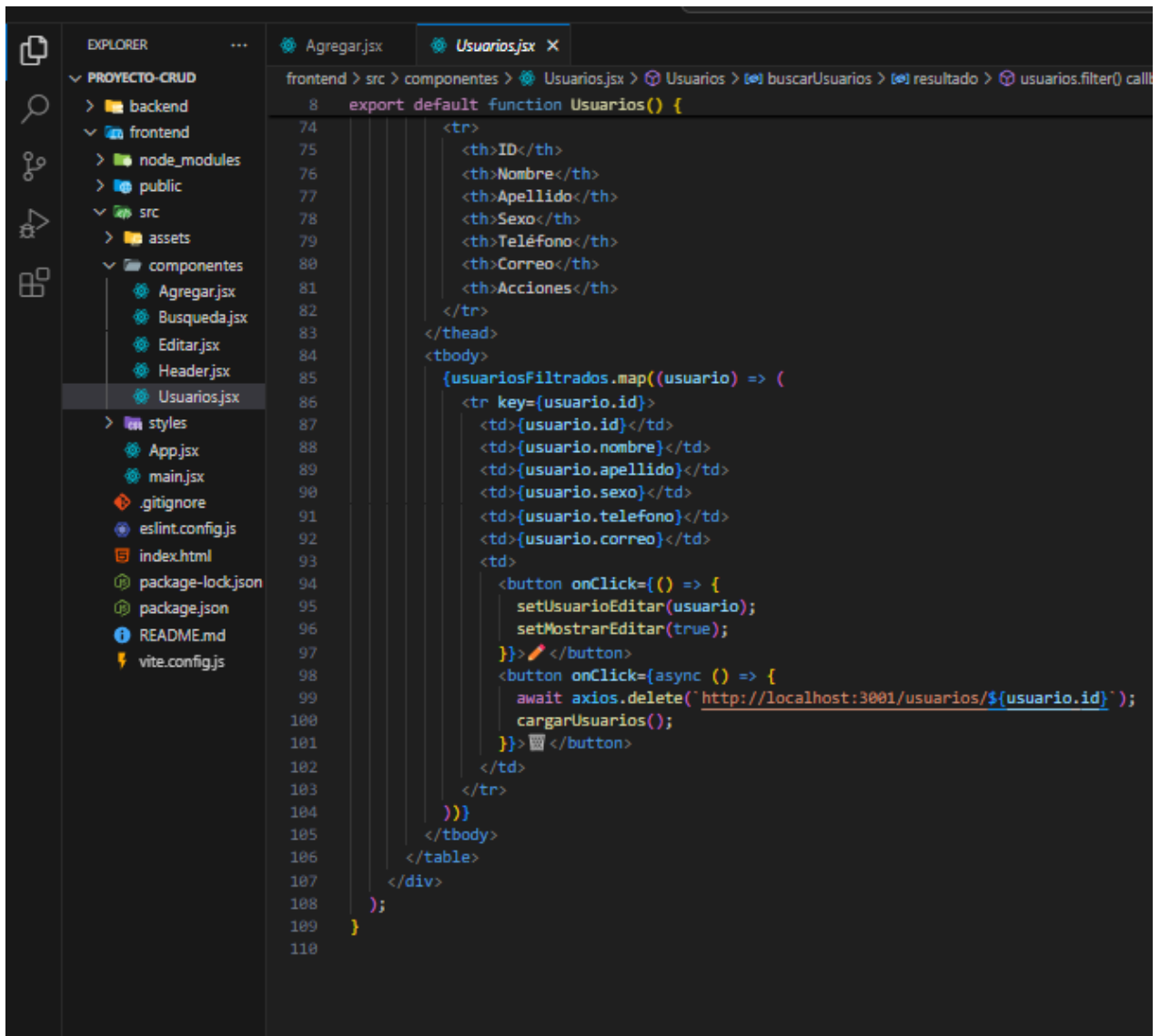
- PROYECTO-CRUD
 - backend
 - frontend
 - node_modules
 - public
 - src
 - assets
 - componentes
 - Agregar.jsx
 - Busqueda.jsx
 - Editar.jsx
 - Header.jsx
 - Usuarios.jsx
 - styles
 - App.jsx
 - main.jsx
 - .gitignore
 - eslint.config.js
 - index.html
 - package-lock.json
 - package.json
 - README.md
 - vite.config.js

Agregar.jsx

Usuarios.jsx

frontend > src > componentes > Usuarios.jsx > Usuarios > buscarUsuarios > resultado > usuarios.filter() callback

```
8 export default function Usuarios() {
48
49   return (
50     <div className="contenedor">
51       <h2 className="titulo">Gestión de Usuarios</h2>
52       <button className="btn-agregar" onClick={() => setMostrarFormulario(true)}>
53         Agregar
54       </button>
55
56       {mostrarFormulario && (
57         <Agregar cerrar={() => setMostrarFormulario(false)} guardar={agregarUsuario} />
58       )}
59
60       {mostrarEditar && usuarioEditar && (
61         <Editar
62           usuario={usuarioEditar}
63           cerrar={() => {
64             setUsuarioEditar(null);
65             setMostrarEditar(false);
66           }}
67           actualizar={actualizarUsuario}
68         />
69       )}
70
71       <Busqueda onBuscar={buscarUsuarios} />
72
73       <table className="tabla-usuarios">
74         <thead>
75           <tr>
76             <th>ID</th>
77             <th>Nombre</th>
78             <th>Apellido</th>
79             <th>Sexo</th>
80             <th>Teléfono</th>
81             <th>Correo</th>
82             <th>Acciones</th>
83           </tr>
84         </thead>
85         <tbody>
86           {usuariosFiltrados.map((usuario) => (
87             <tr key={usuario.id}>
88               <td>{usuario.id}</td>
89               <td>{usuario.nombre}</td>
90               <td>{usuario.apellido}</td>
91               <td>{usuario.sexo}</td>
92               <td>{usuario.telefono}</td>
93               <td>{usuario.correo}</td>
```



The image shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure for 'PROYECTO-CRUD' with folders 'backend', 'frontend', 'node_modules', 'public', 'src', and 'assets'. The 'src' folder is expanded, showing 'components' and 'styles'. The 'components' folder is expanded, showing 'Agregar.jsx', 'Busqueda.jsx', 'Editar.jsx', 'Header.jsx', and 'Usuarios.jsx'. The 'Usuarios.jsx' file is selected. The code editor shows the code for 'Usuarios.jsx'.

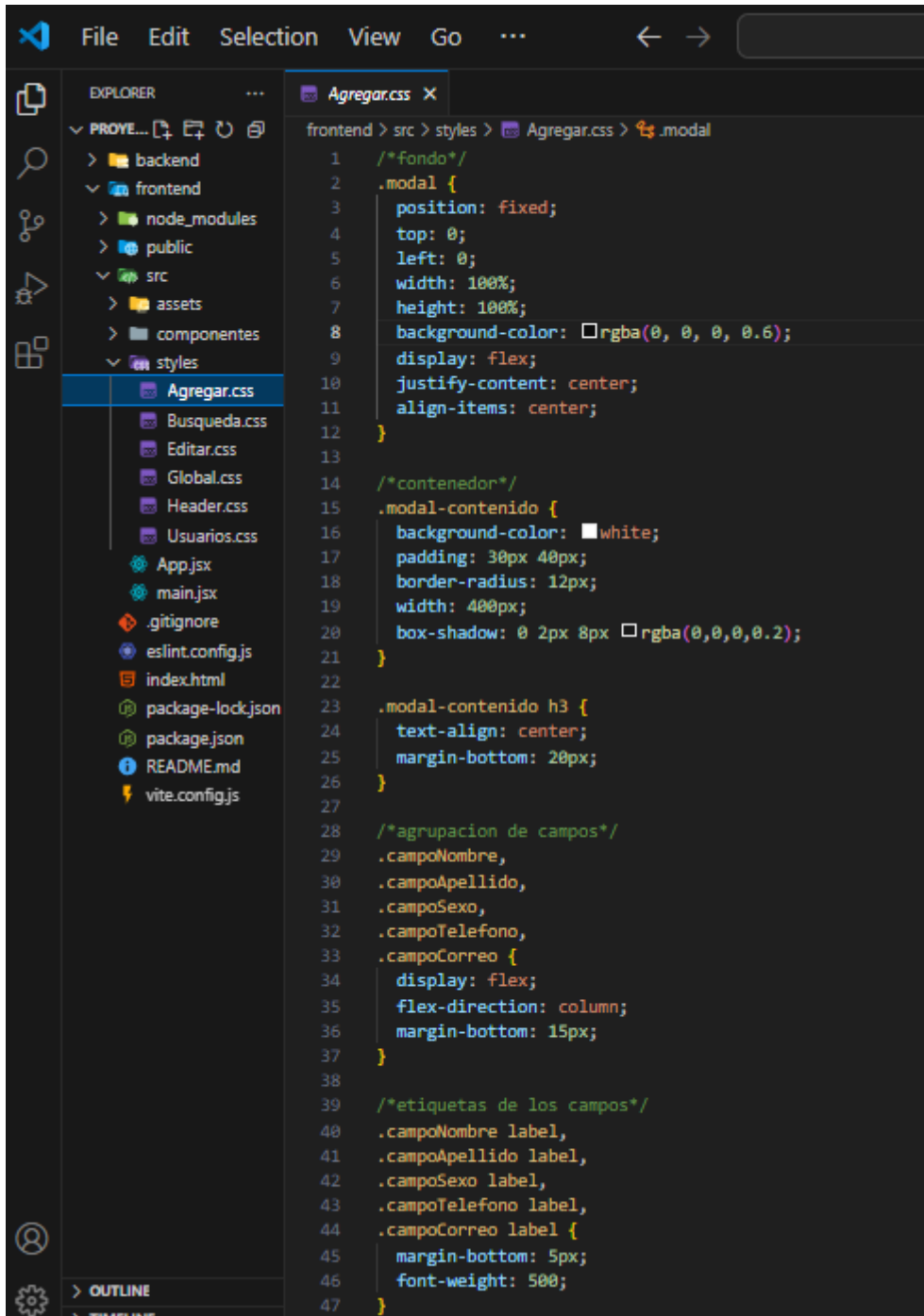
```
8 export default function Usuarios() {
74
75     <th>ID</th>
76     <th>Nombre</th>
77     <th>Apellido</th>
78     <th>Sexo</th>
79     <th>Teléfono</th>
80     <th>Correo</th>
81     <th>Acciones</th>
82   </tr>
83 </thead>
84 <tbody>
85   {usuariosFiltrados.map((usuario) => (
86     <tr key={usuario.id}>
87       <td>{usuario.id}</td>
88       <td>{usuario.nombre}</td>
89       <td>{usuario.apellido}</td>
90       <td>{usuario.sexo}</td>
91       <td>{usuario.telefono}</td>
92       <td>{usuario.correo}</td>
93       <td>
94         <button onClick={() => {
95           setUsuarioEditar(usuario);
96           setMostrarEditar(true);
97         }}> ✎ </button>
98         <button onClick={async () => {
99           await axios.delete(`http://localhost:3001/usuarios/${usuario.id}`);
100           cargarUsuarios();
101         }}> 🗑 </button>
102       </td>
103     </tr>
104   )})
105 </tbody>
106 </table>
107 </div>
108 );
109 }
110
```

Este componente centraliza y organiza las operaciones de gestión de usuarios y también controla el acceso a los demás componentes fundamentales. Este mantiene un estado general de los usuarios mediante `useState`, y `Axios` para comunicarse con el Backend a través de la API REST, lo que le permite realizar operaciones de lectura, inserción, actualización y eliminación.

También maneja la lógica de visualización de formularios mediante condicionales que activan o cierran las interfaces de agregar o editar según sea la acción seleccionada. En resumen, este componente actúa como una interfaz general para interactuar con el sistema y para garantizar la sincronización de los datos entre la base de datos y la vista del usuario.

Estilos

Agregar.css



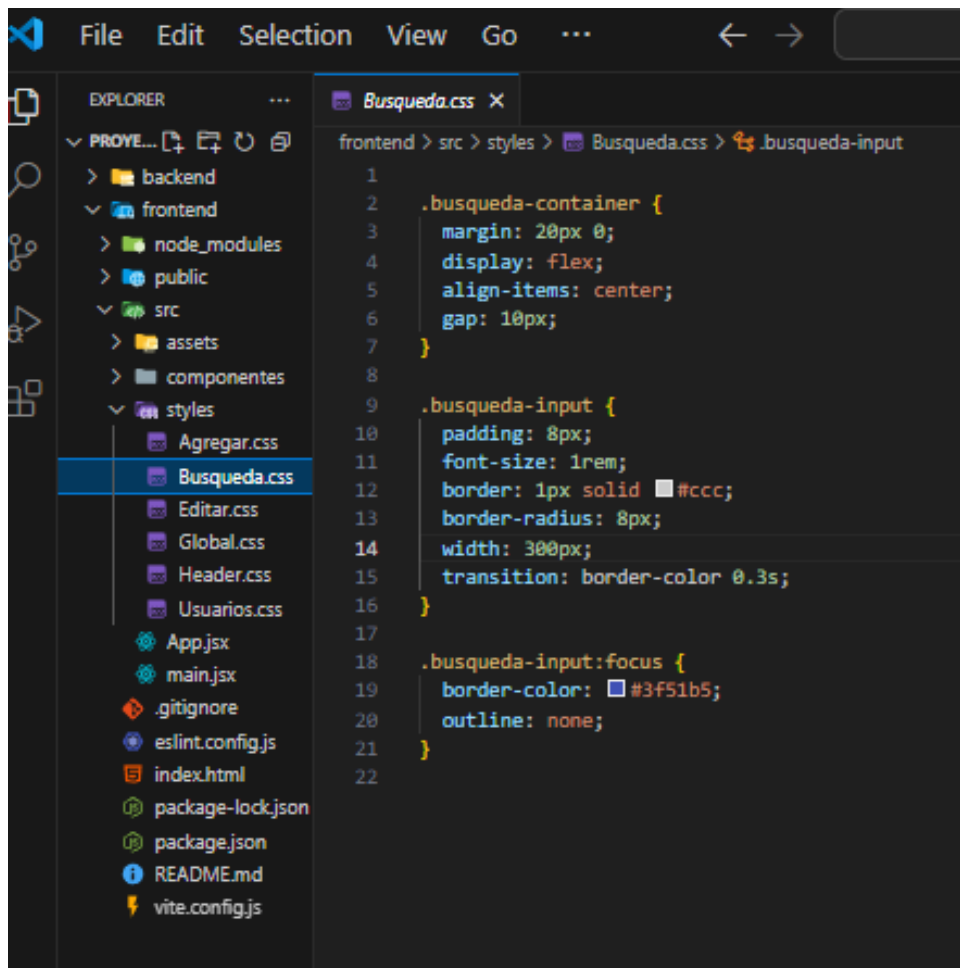
The image shows a screenshot of the Visual Studio Code (VS Code) interface. On the left, the Explorer sidebar is open, showing a file tree with the following structure:

- PROYECTO
- backend
- frontend
 - node_modules
 - public
 - src
 - assets
 - componentes
 - styles
 - Agregar.css** (selected)
 - Busqueda.css
 - Editar.css
 - Global.css
 - Header.css
 - Usuarios.css
- .gitignore
- eslint.config.js
- index.html
- package-lock.json
- package.json
- README.md
- vite.config.js

On the right, the Editor view shows the content of the `Agregar.css` file. The code is as follows:

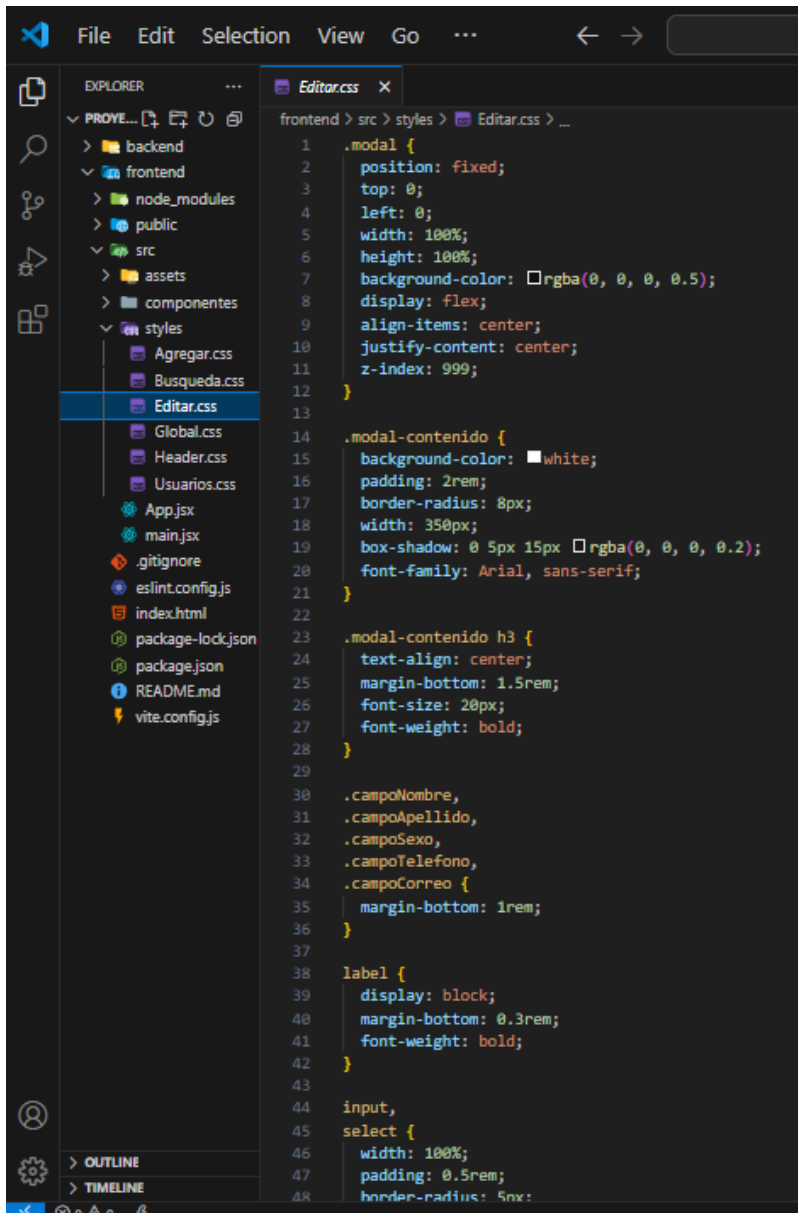
```
1 /*fondo*/
2 .modal {
3   position: fixed;
4   top: 0;
5   left: 0;
6   width: 100%;
7   height: 100%;
8   background-color: rgba(0, 0, 0, 0.6);
9   display: flex;
10  justify-content: center;
11  align-items: center;
12 }
13
14 /*contenedor*/
15 .modal-contenido {
16   background-color: white;
17   padding: 30px 40px;
18   border-radius: 12px;
19   width: 400px;
20   box-shadow: 0 2px 8px rgba(0,0,0,0.2);
21 }
22
23 .modal-contenido h3 {
24   text-align: center;
25   margin-bottom: 20px;
26 }
27
28 /*agrupacion de campos*/
29 .campoNombre,
30 .campoApellido,
31 .campoSexo,
32 .campoTelefono,
33 .campoCorreo {
34   display: flex;
35   flex-direction: column;
36   margin-bottom: 15px;
37 }
38
39 /*etiquetas de los campos*/
40 .campoNombre label,
41 .campoApellido label,
42 .campoSexo label,
43 .campoTelefono label,
44 .campoCorreo label {
45   margin-bottom: 5px;
46   font-weight: 500;
47 }
```

Busqueda.css

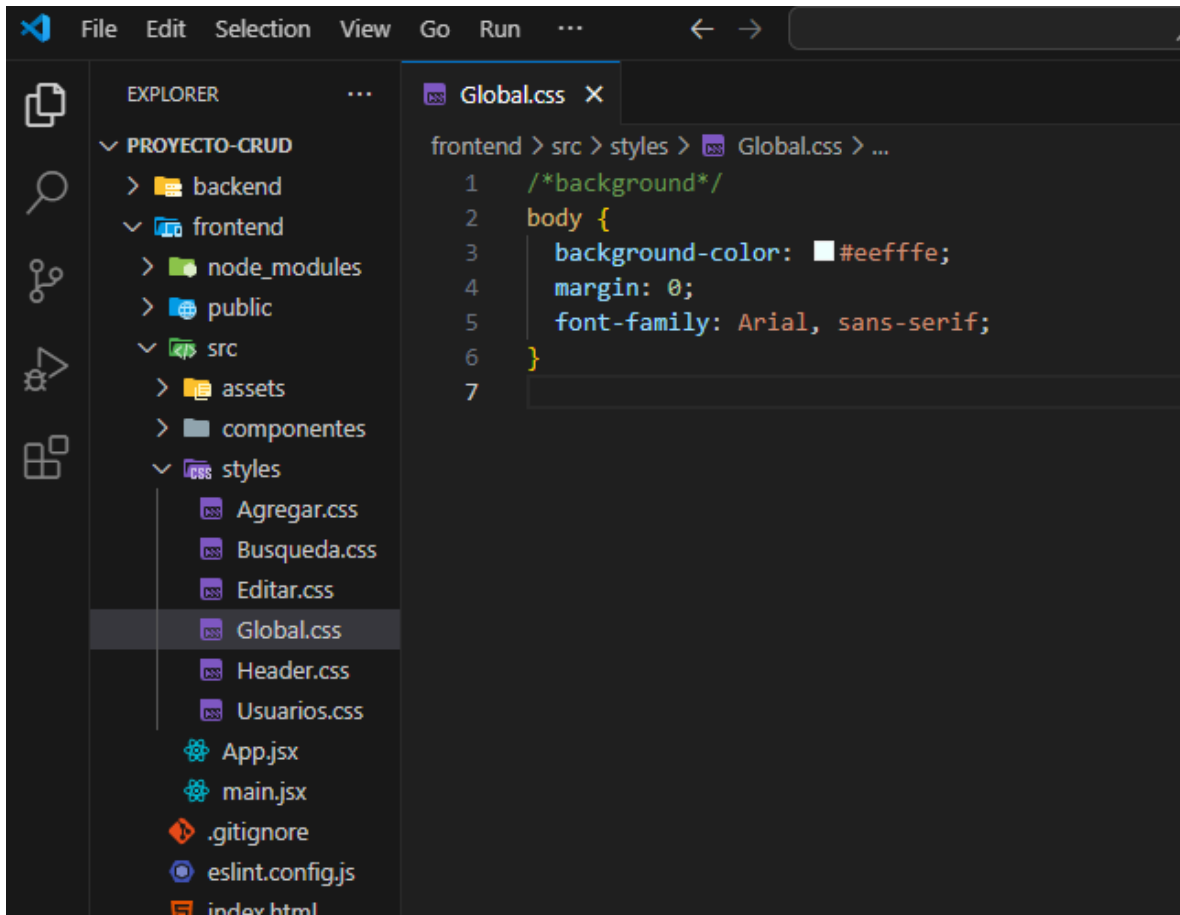


```
1
2 .busqueda-container {
3   margin: 20px 0;
4   display: flex;
5   align-items: center;
6   gap: 10px;
7 }
8
9 .busqueda-input {
10  padding: 8px;
11  font-size: 1rem;
12  border: 1px solid #ccc;
13  border-radius: 8px;
14  width: 300px;
15  transition: border-color 0.3s;
16 }
17
18 .busqueda-input:focus {
19  border-color: #3f51b5;
20  outline: none;
21 }
22
```


Editor.css

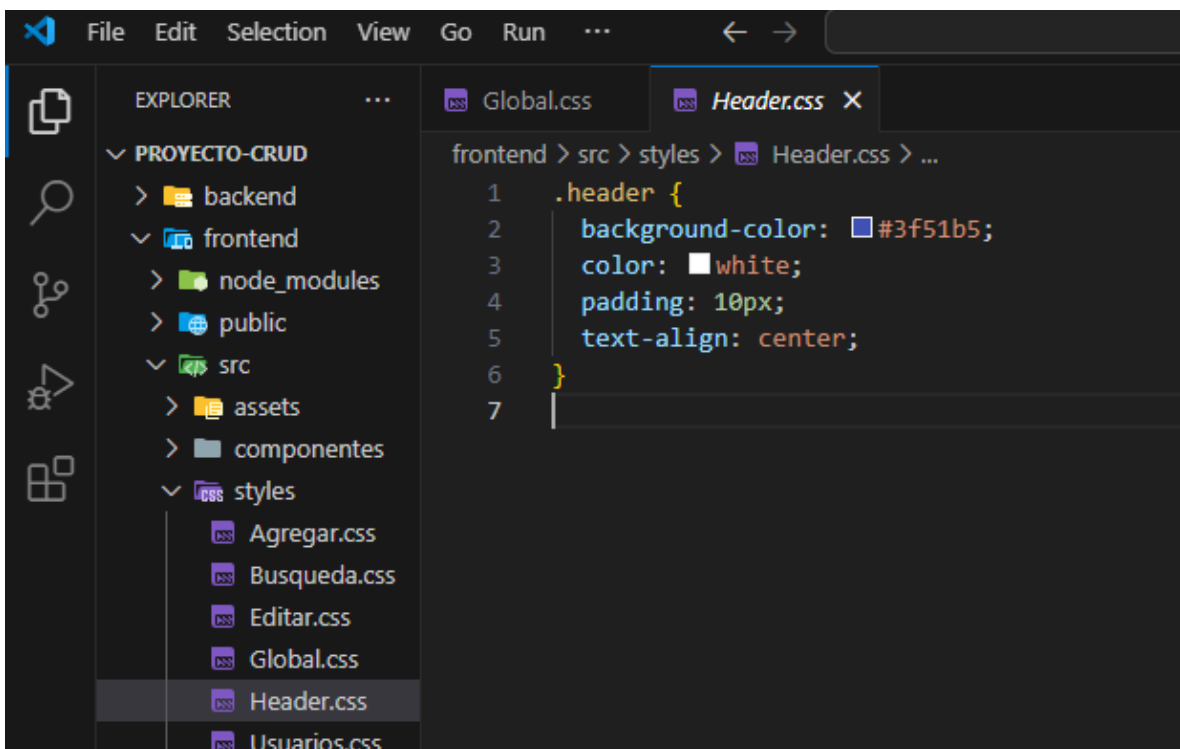


Global.css y Header.css



This screenshot shows the Visual Studio Code editor with the Explorer sidebar on the left and the Global.css file open in the editor. The Explorer sidebar shows the project structure for 'PROYECTO-CRUD', with the 'styles' folder expanded. The 'Global.css' file is selected. The editor displays the following CSS code:

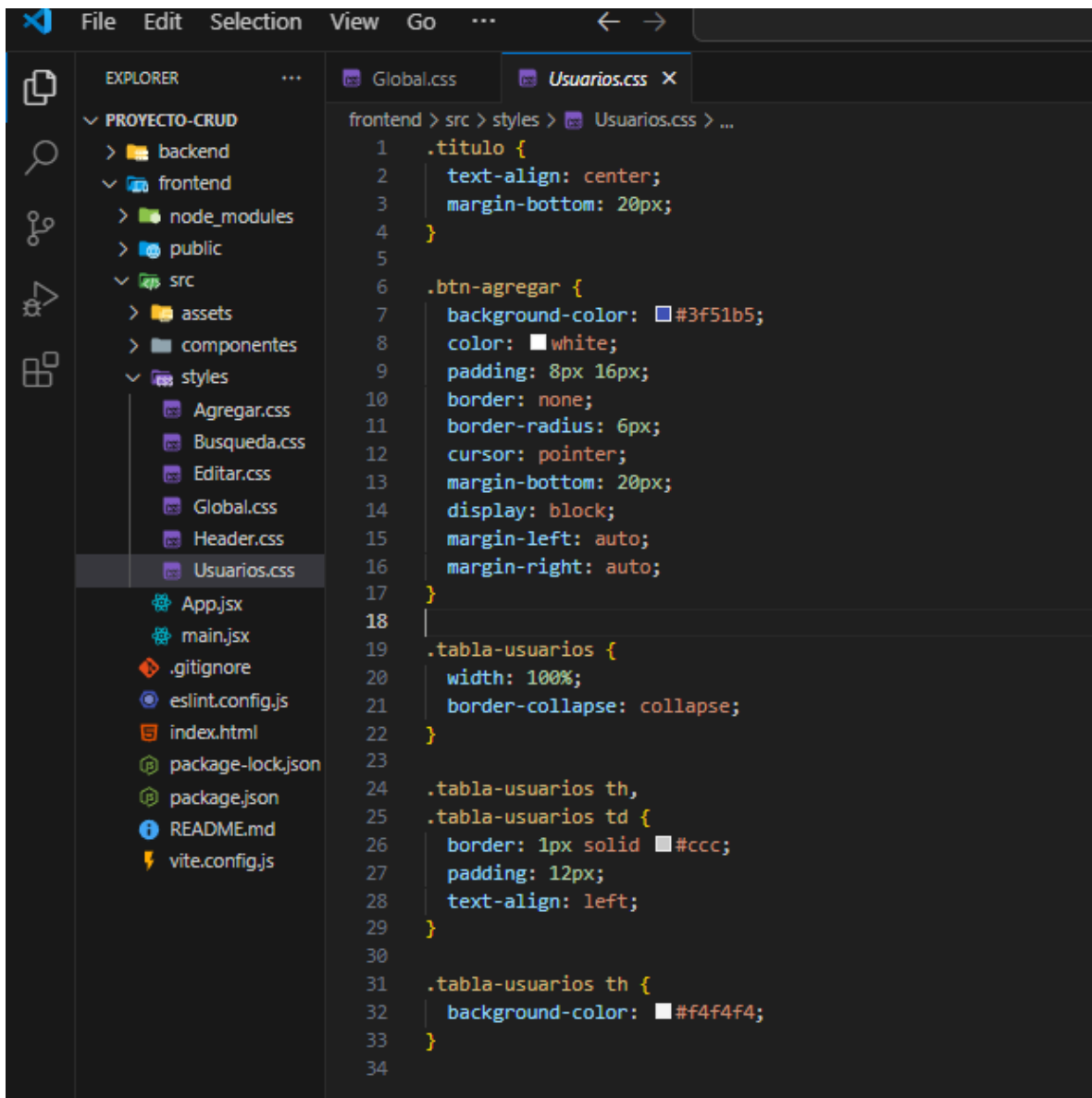
```
1  /*background*/
2  body {
3      background-color: #eefffe;
4      margin: 0;
5      font-family: Arial, sans-serif;
6  }
```



This screenshot shows the Visual Studio Code editor with the Explorer sidebar on the left and the Header.css file open in the editor. The Explorer sidebar shows the project structure for 'PROYECTO-CRUD', with the 'styles' folder expanded. The 'Header.css' file is selected. The editor displays the following CSS code:

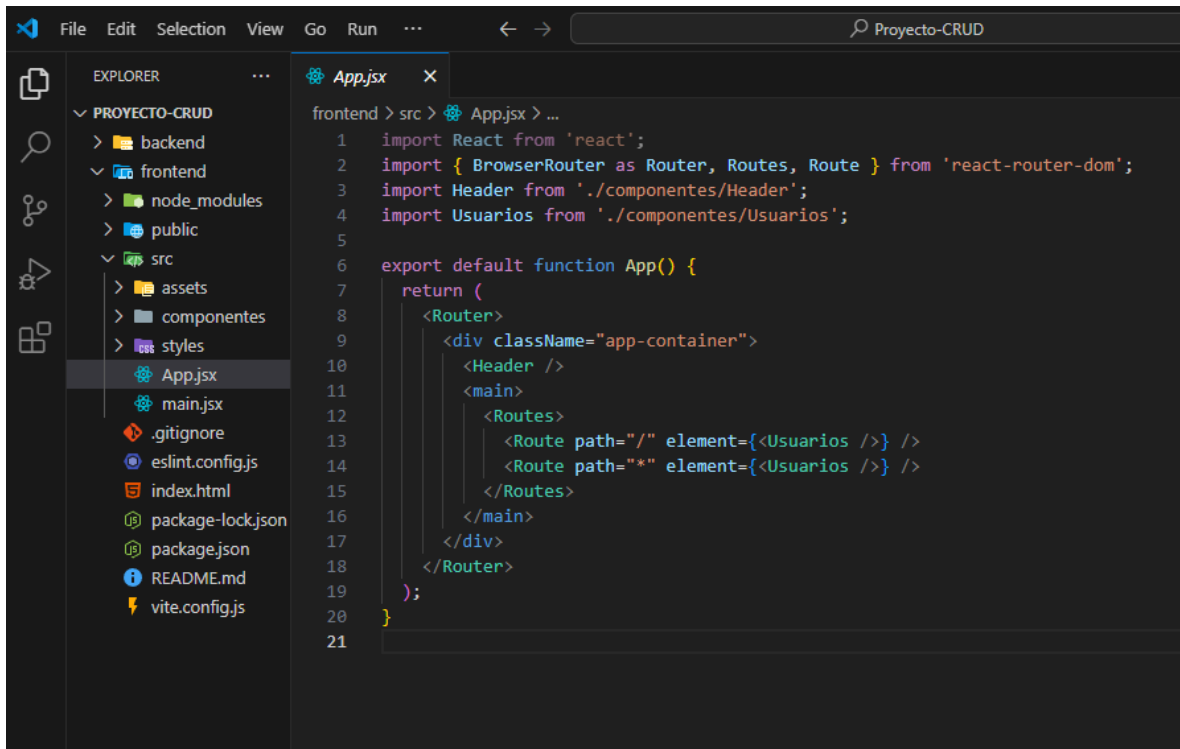
```
1  .header {
2      background-color: #3f51b5;
3      color: white;
4      padding: 10px;
5      text-align: center;
6  }
```

Usuarios.css



```
File Edit Selection View Go ...  
EXPLORER  
PROYECTO-CRUD  
  backend  
  frontend  
  node_modules  
  public  
  src  
    assets  
    componentes  
    styles  
      Agregars.css  
      Busqueda.css  
      Editar.css  
      Global.css  
      Header.css  
      Usuarios.css  
  App.jsx  
  main.jsx  
  .gitignore  
  eslint.config.js  
  index.html  
  package-lock.json  
  package.json  
  README.md  
  vite.config.js  
Global.css  
Usuarios.css X  
frontend > src > styles > Usuarios.css > ...  
1  .titulo {  
2    text-align: center;  
3    margin-bottom: 20px;  
4  }  
5  
6  .btn-agregar {  
7    background-color: #3f51b5;  
8    color: white;  
9    padding: 8px 16px;  
10   border: none;  
11   border-radius: 6px;  
12   cursor: pointer;  
13   margin-bottom: 20px;  
14   display: block;  
15   margin-left: auto;  
16   margin-right: auto;  
17 }  
18  
19 .tabla-usuarios {  
20   width: 100%;  
21   border-collapse: collapse;  
22 }  
23  
24 .tabla-usuarios th,  
25 .tabla-usuarios td {  
26   border: 1px solid #ccc;  
27   padding: 12px;  
28   text-align: left;  
29 }  
30  
31 .tabla-usuarios th {  
32   background-color: #f4f4f4;  
33 }  
34
```

App.js del Front

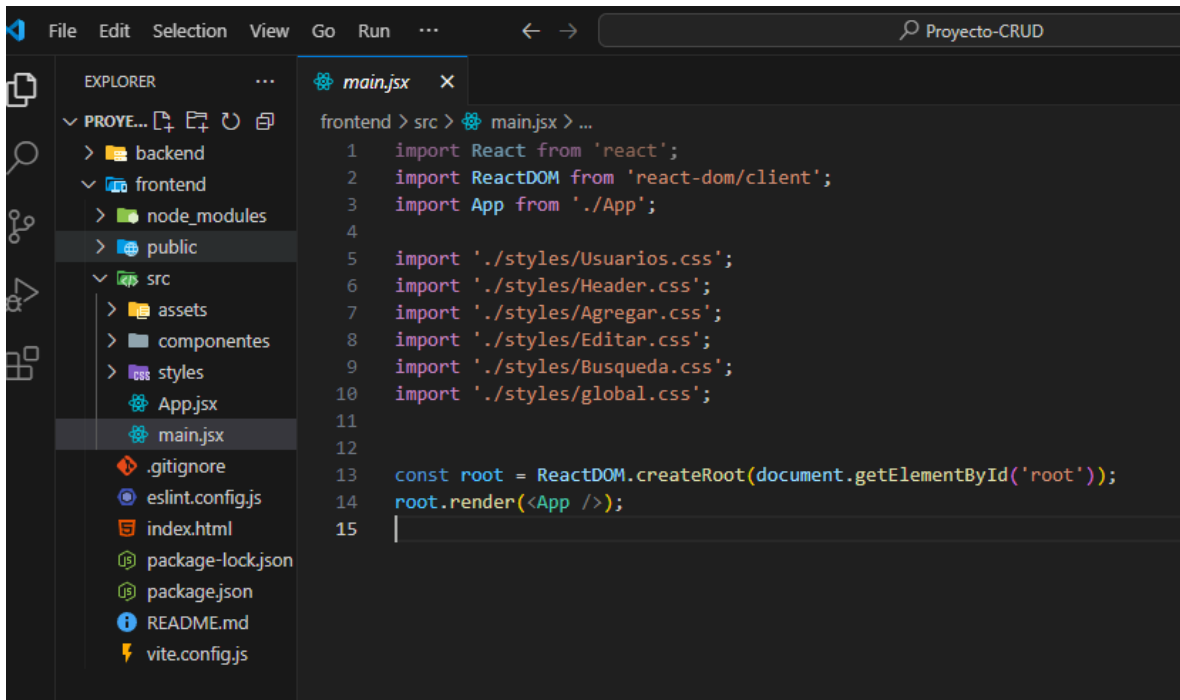


The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar shows a file tree for 'PROYECTO-CRUD' with folders 'backend', 'frontend', 'node_modules', 'public', and 'src'. The 'src' folder is expanded, showing 'assets', 'componentes', 'styles', 'App.jsx' (selected), 'main.jsx', and various configuration files. The main editor area displays the code for 'App.jsx' with line numbers 1 through 21. The code imports React, Router, Routes, Route, Header, and Usuarios, and defines an App function that returns a JSX structure with a Router, Header, and Routes.

```
1 import React from 'react';
2 import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
3 import Header from './componentes/Header';
4 import Usuarios from './componentes/Usuarios';
5
6 export default function App() {
7   return (
8     <Router>
9       <div className="app-container">
10        <Header />
11        <main>
12          <Routes>
13            <Route path="/" element={<Usuarios />} />
14            <Route path="*" element={<Usuarios />} />
15          </Routes>
16        </main>
17      </div>
18    </Router>
19  );
20 }
21
```

El archivo App.js tenemos la entrada principal de la aplicación React donde se encarga de estructurar la navegación entre componentes usando React. Aquí se definió que, al acceder a la ruta principal (/) o cualquier otra no reconocida (*), se mostrará el componente Usuarios, que contiene toda la funcionalidad del CRUD. También incluye el componente Header, que se muestra siempre como parte de la interfaz, independientemente de la ruta. Ósea aquí se hace la visualización de los componentes.

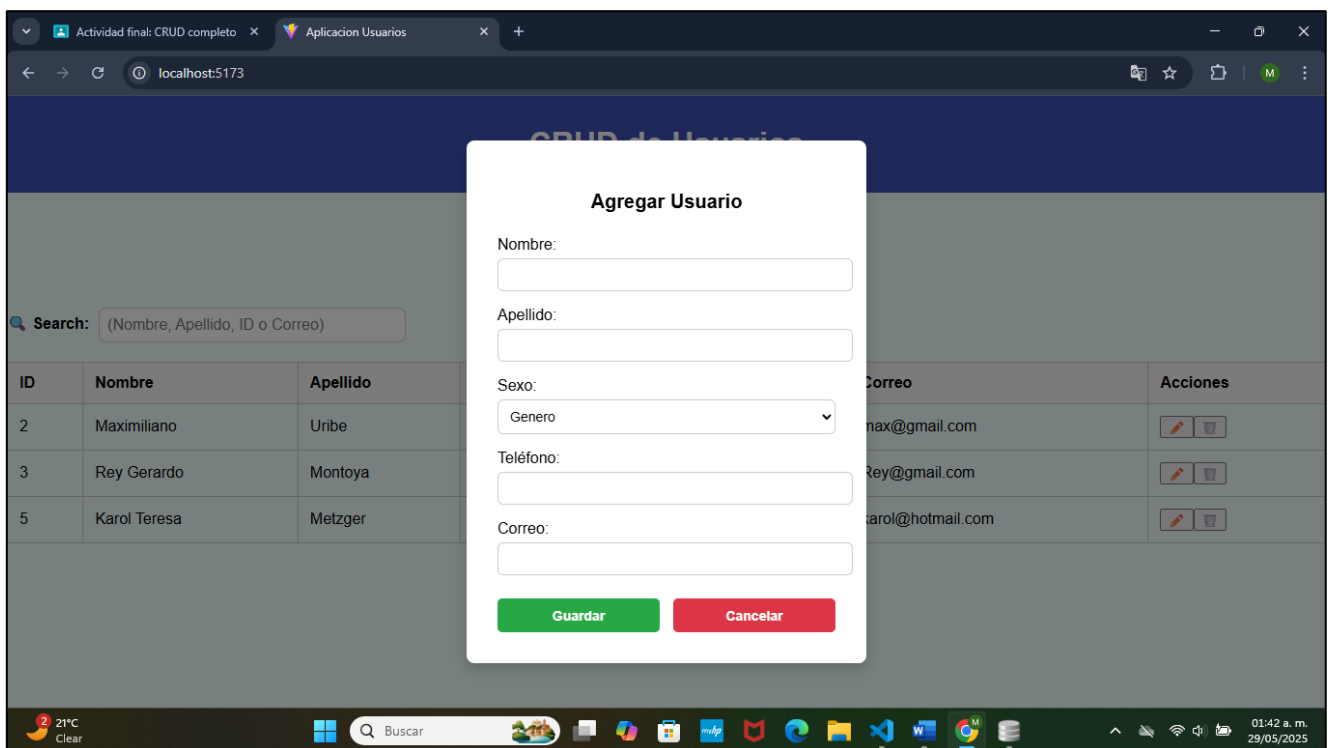
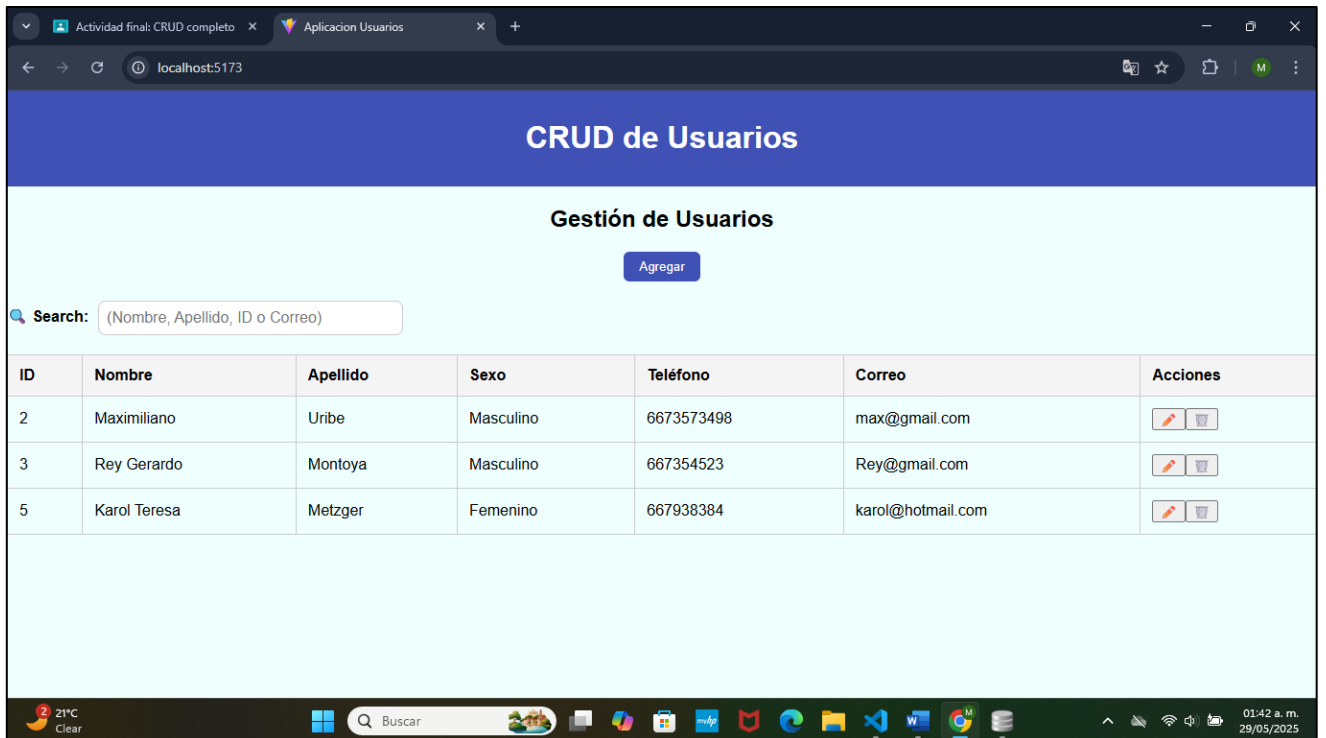
Main.jsx

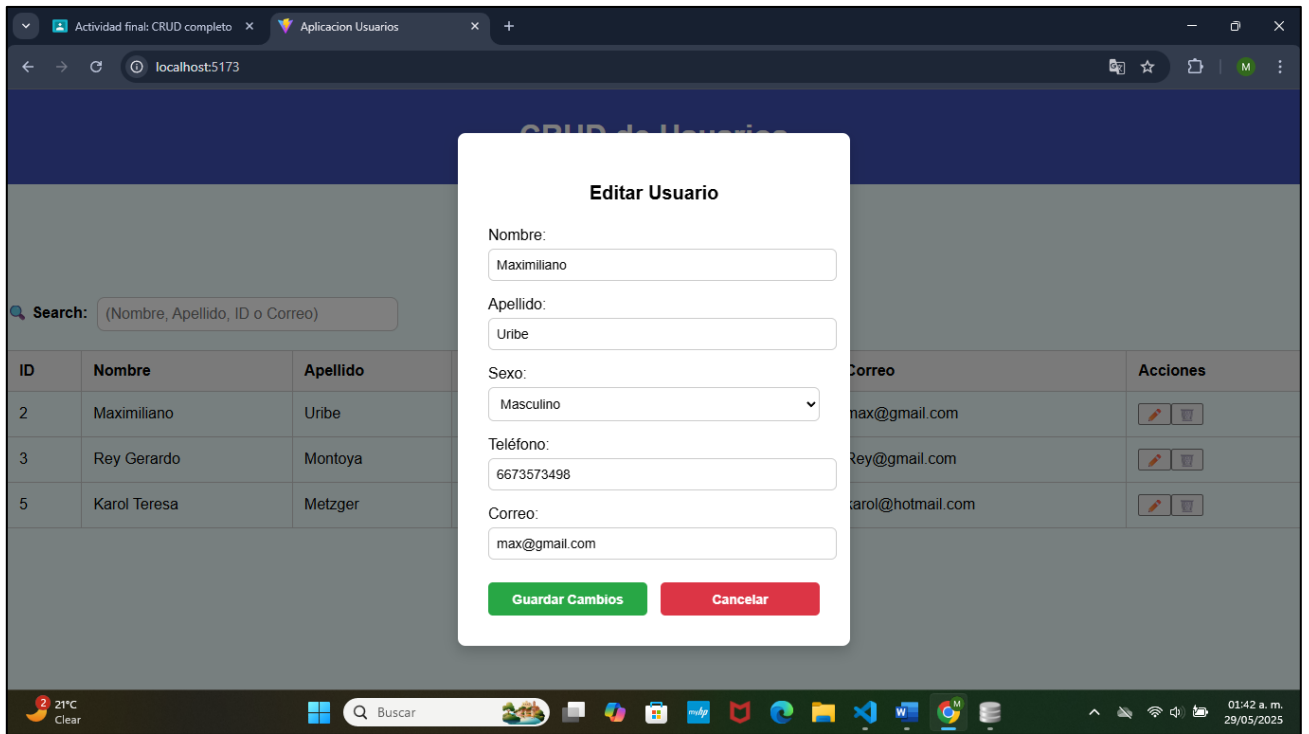


```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3  import App from './App';
4
5  import './styles/Usuarios.css';
6  import './styles/Header.css';
7  import './styles/Agregar.css';
8  import './styles/Editar.css';
9  import './styles/Busqueda.css';
10 import './styles/global.css';
11
12
13 const root = ReactDOM.createRoot(document.getElementById('root'));
14 root.render(<App />);
15
```

Este archivo donde se crea el nodo raíz en el elemento HTML con un id root, y luego renderiza el App, que contiene toda la estructura. Además, este main importa todos los archivos CSS necesarios para los estilos de los componentes.

Imágenes de la app ya terminada





DB Browser for SQLite - C:\Users\jesus\OneDrive\Escritorio\Proyecto-CRUD\backend\usuarios.sqlite

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragma Execute SQL

sqlite_sequence usuarios

Table: usuarios

id	nombre	apellido	sexo	telefono	correo
1	2 Maximiliano	Uribe	Masculino	6673573498	max@gmail.com
2	3 Rey Gerardo	Montoya	Masculino	667354523	Rey@gmail.com
3	5 Karol Teresa	Metzger	Femenino	667938384	karol@hotma...

Go to: 1

Edit Database Cell

Mode: Text

1

Editing row=1, column=0
Type: Text / Numeric; Size: 1 character(s)

Remote

Identity Select an identity to connect

DBHub.io Local Current Database

Name Last modified Size

SQL Log Plot DB Schema Remote

UTF-8

21°C Clear

Buscar

01:43 a.m. 29/05/2025