



MATERIA:

DW LADO DEL SERVIDOR

TITULO:

**Actividad final: CRUD completo
Explicación del código (Front y Back)**

MAESTRO:

José Manuel Cazarez Alderete

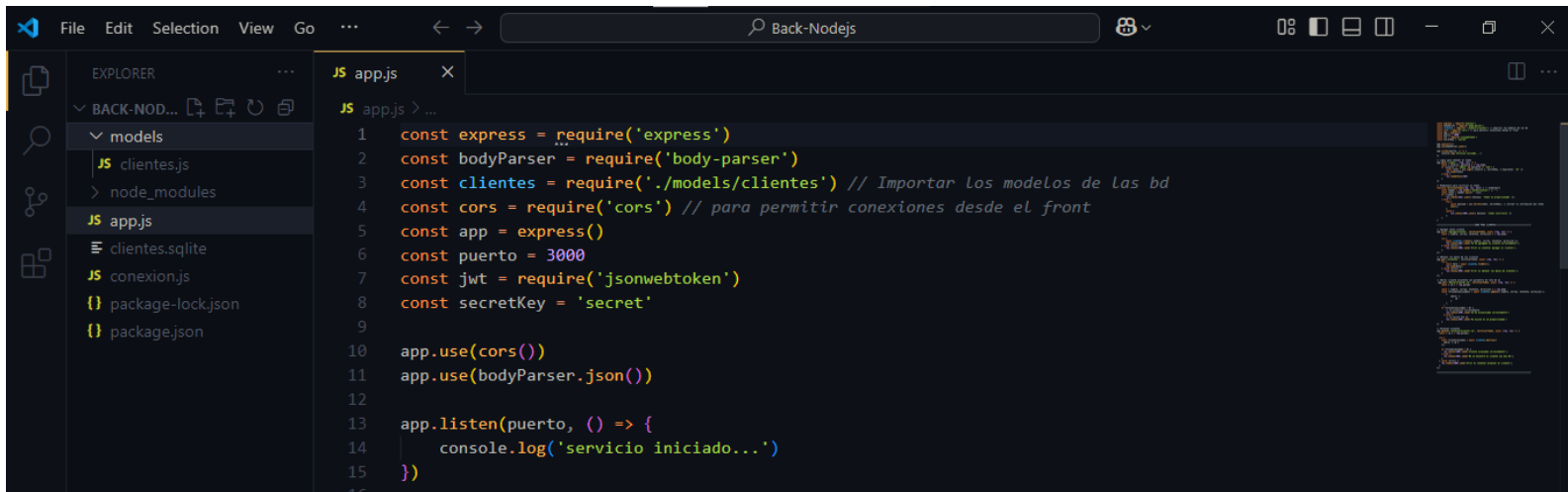
ALUMNO:

Jorge Luis Astorga Meza

Grupo: 2-3

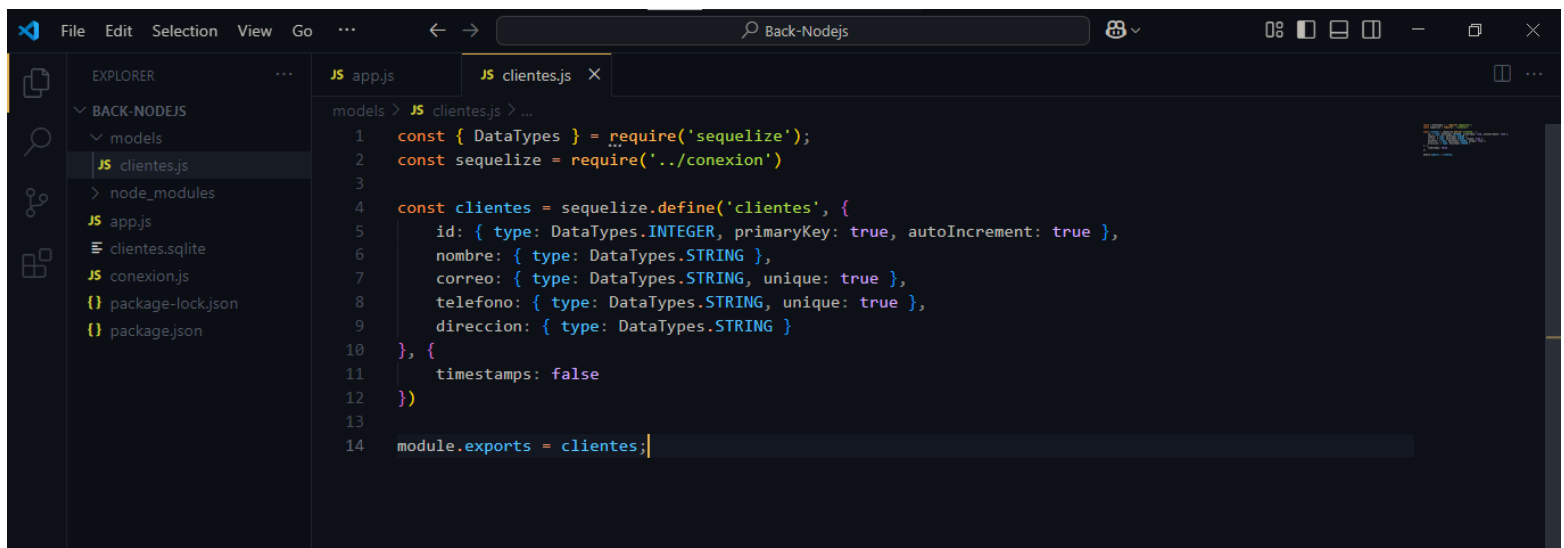
FECHA: 29/05/25

Explicación del código del Backend (Capturas)



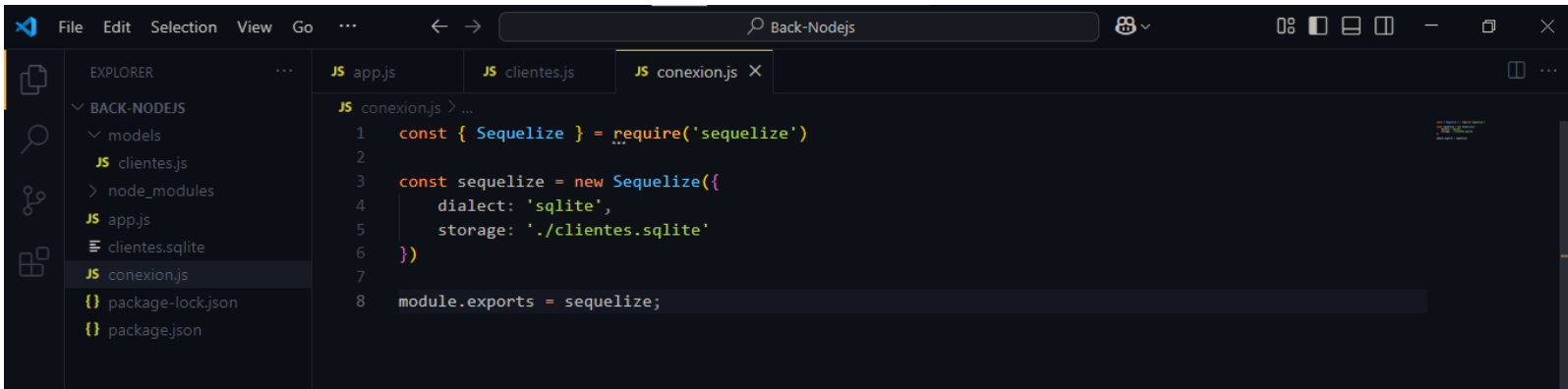
```
1  const express = require('express')
2  const bodyParser = require('body-parser')
3  const clientes = require('./models/clientes') // Importar Los modelos de Las bd
4  const cors = require('cors') // para permitir conexiones desde el front
5  const app = express()
6  const puerto = 3000
7  const jwt = require('jsonwebtoken')
8  const secretKey = 'secret'
9
10 app.use(cors())
11 app.use(bodyParser.json())
12
13 app.listen(puerto, () => {
14   console.log('servicio iniciado...')
15 })
```

Empezando con el (Backend) en el app.js básicamente tenemos los requerimientos que tenemos utilizar en si para poder hacer uso de funciones que nos permitirán más adelante como por ejemplo **express** que nos deja hacer el framework principal para crear el servidor con Node.js, el **bodyParser** que como tal que permite leer datos del cuerpo de las peticiones HTTP, como formularios o JSON. **Siguiendo pues importamos el modelo de clientes, el cual definimos con (Sequelize)**, que lo importamos de una carpeta de modelos previamente hecha tanto en **DB Browser** como la tabla (estructura) en código.



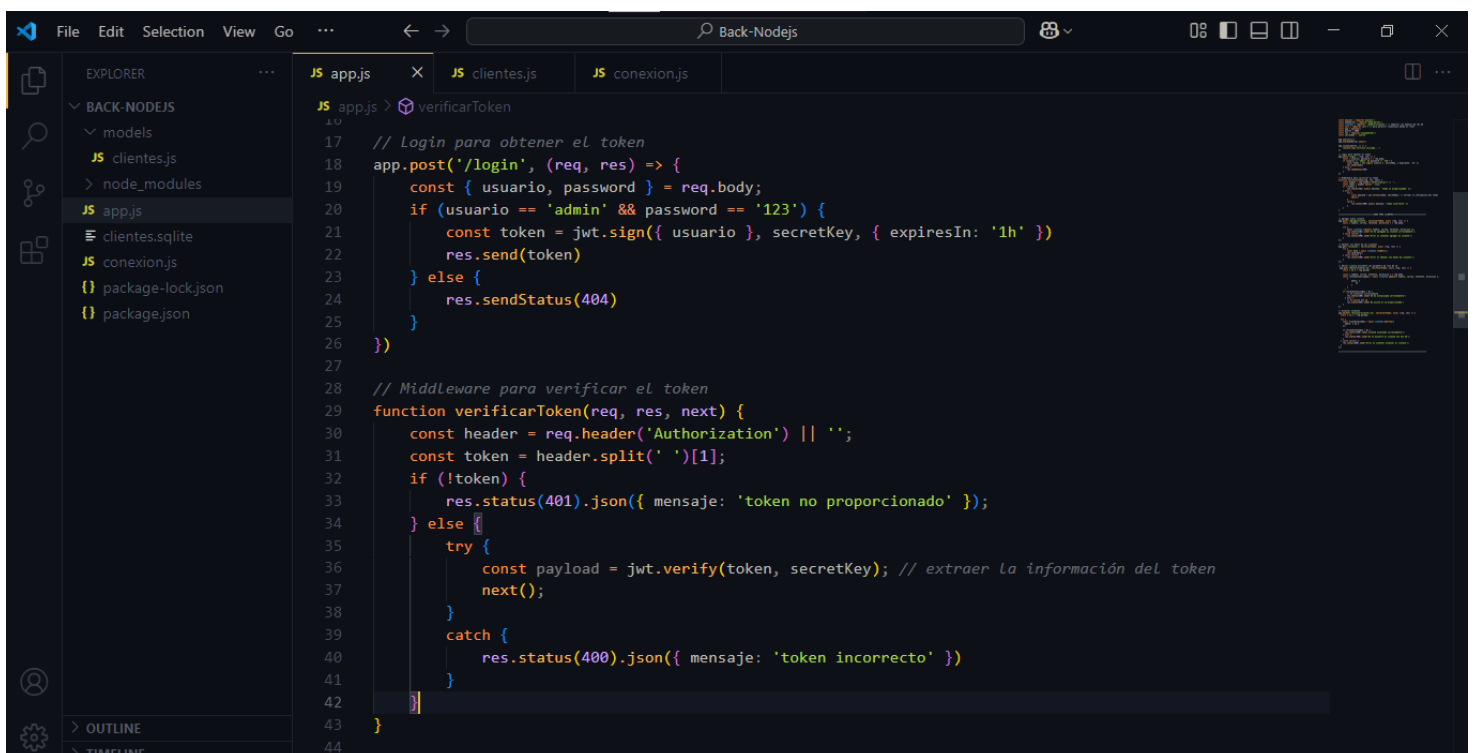
```
1  const { DataTypes } = require('sequelize');
2  const sequelize = require('../conexion')
3
4  const clientes = sequelize.define('clientes', {
5    id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
6    nombre: { type: DataTypes.STRING },
7    correo: { type: DataTypes.STRING, unique: true },
8    telefono: { type: DataTypes.STRING, unique: true },
9    direccion: { type: DataTypes.STRING }
10  }, {
11    timestamps: false
12  })
13
14  module.exports = clientes;
```

El puerto quedaría por defecto a como se manejó en clases siendo 3000 como todo los demás. Continuando con **CORS** y **JWT** básicamente el primero de ellos nos sirve para **Importar el** (Cross-Origin Resource Sharing), que nos permite que el frontend en React (en otro puerto) pueda hacer peticiones al backend (en este caso, a Express). Y el otro **Importa JWT**, que se usa para crear y verificar tokens de autenticación.



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

En esta parte **Importa** Sequelize, que es el **ORM** que permite conectar y trabajar con bases de datos desde Node.js de una manera sencilla y no tan compleja. Crea una conexión a una **base de datos SQLite** que se guarda en el archivo clientes.sqlite (Creado desde DB Browser Claramente) para de ultimo exportarla y hacer uso de ella.



```
17 // Login para obtener el token
18 app.post('/login', (req, res) => {
19   const { usuario, password } = req.body;
20   if (usuario == 'admin' && password == '123') {
21     const token = jwt.sign({ usuario }, secretKey, { expiresIn: '1h' });
22     res.send(token)
23   } else {
24     res.sendStatus(404)
25   }
26 })
27
28 // Middleware para verificar el token
29 function verificarToken(req, res, next) {
30   const header = req.header('Authorization') || '';
31   const token = header.split(' ')[1];
32   if (!token) {
33     res.status(401).json({ mensaje: 'token no proporcionado' });
34   } else {
35     try {
36       const payload = jwt.verify(token, secretKey); // extraer la información del token
37       next();
38     } catch {
39       res.status(400).json({ mensaje: 'token incorrecto' });
40     }
41   }
42 }
43
44 }
```

El *Login* más que nada nos funciona cuando alguien que pone el usuario y contraseña correctos (admin y 123), se le dará un **token** con acceso a las rutas y así pueda entrar a las demás partes del sistema que integran el **CRUD**.

El *Middleware* es como un código que revisa si el token que viene en la petición, es válido; si no lo es, **no lo dejara pasar** a las rutas protegidas.



```
File Edit Selection View Go ... Back-Nodejs
EXPLORER
  BACK-NODEJS
    models
      JS clientes.js
    node_modules
  JS app.js
    clientes.sqlite
    conexion.js
    package-lock.json
    package.json
  JS app.js
    43
    44
    45 //////////////////////////////////////////////////CRUD PARA CLIENTES////////////////////////////////////
    46
    47 // Agregar nuevo cliente
    48 app.post('/AgregarCliente', verificarToken, async (req, res) => {
    49   const { nombre, correo, telefono, direccion } = req.body;
    50
    51   try {
    52     await clientes.create({ nombre, correo, telefono, direccion });
    53     res.status(201).send('Se ha agregado un cliente correctamente');
    54   } catch (error) {
    55     res.status(500).send('Error al intentar agregar al cliente');
    56   }
    57 });
    58
    59 // Obtener los datos de los clientes
    60 app.get('/Clientes', verificarToken, async (req, res) => {
    61   try {
    62     const data = await clientes.findAll();
    63     res.send(data);
    64   } catch (error) {
    65     res.status(500).send('Error al obtener los datos de clientes');
    66   }
    67 });
    68
```

CRUD COMPLETO

```
// Editar cliente existente con parametro de ruta de id
app.put('/EditarCliente/:id', verificarToken, async (req, res) => {
  const { id } = req.params

  const { nombre, correo, telefono, direccion } = req.body
  const [filasActualizadas] = await clientes.update({ nombre, correo, telefono, direccion },
    {
      where: {
        id
      }
    }
  )
  if (filasActualizadas > 0) {
    // se actualizó correctamente
    res.status(200).send('Se ha actualizado correctamente')
  } else {
    // no existe ese id
    res.status(404).send('No existe el id proporcionado')
  }
})
})
```

```

90 // Eliminar Clientes
91 app.delete('/EliminarCliente/:id', verificarToken, async (req, res) => {
92     const { id } = req.params;
93
94     try {
95         const filasEliminadas = await clientes.destroy({
96             where: { id }
97         });
98
99         if (filasEliminadas > 0) {
100             res.status(200).send('Cliente eliminado correctamente');
101         } else {
102             res.status(404).send('No se encontró el cliente con ese ID');
103         }
104     } catch (error) {
105         res.status(500).send('Error al intentar eliminar el cliente');
106     }
107 });
108
109 //////////////////////////////////////

```

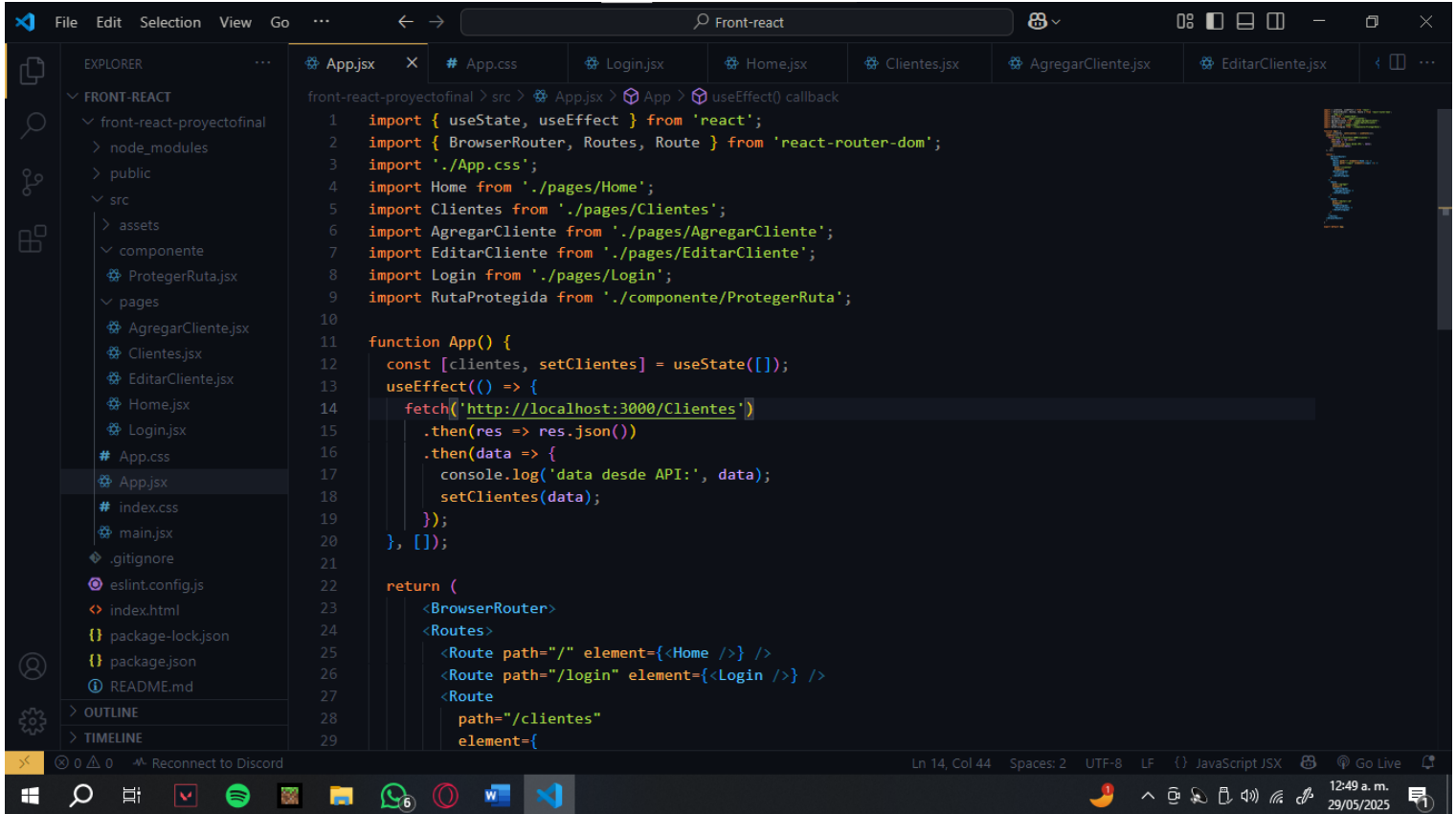
Agregar cliente: Crea un nuevo cliente con los datos enviados desde el frontend. Solo funciona si el usuario tiene un token válido (está logueado).

Obtener clientes: Devuelve la lista de todos los clientes guardados en la base de datos. También requiere que el usuario esté autenticado con un token.

Editar cliente: Busca un cliente por su ID y actualiza su información con los nuevos datos. La edición solo se permite si el token es válido.

Eliminar cliente: Elimina un cliente de la base de datos según su ID. Para poder hacerlo, el usuario debe estar autenticado con un token.

Explicación del código del Frontend (Capturas)

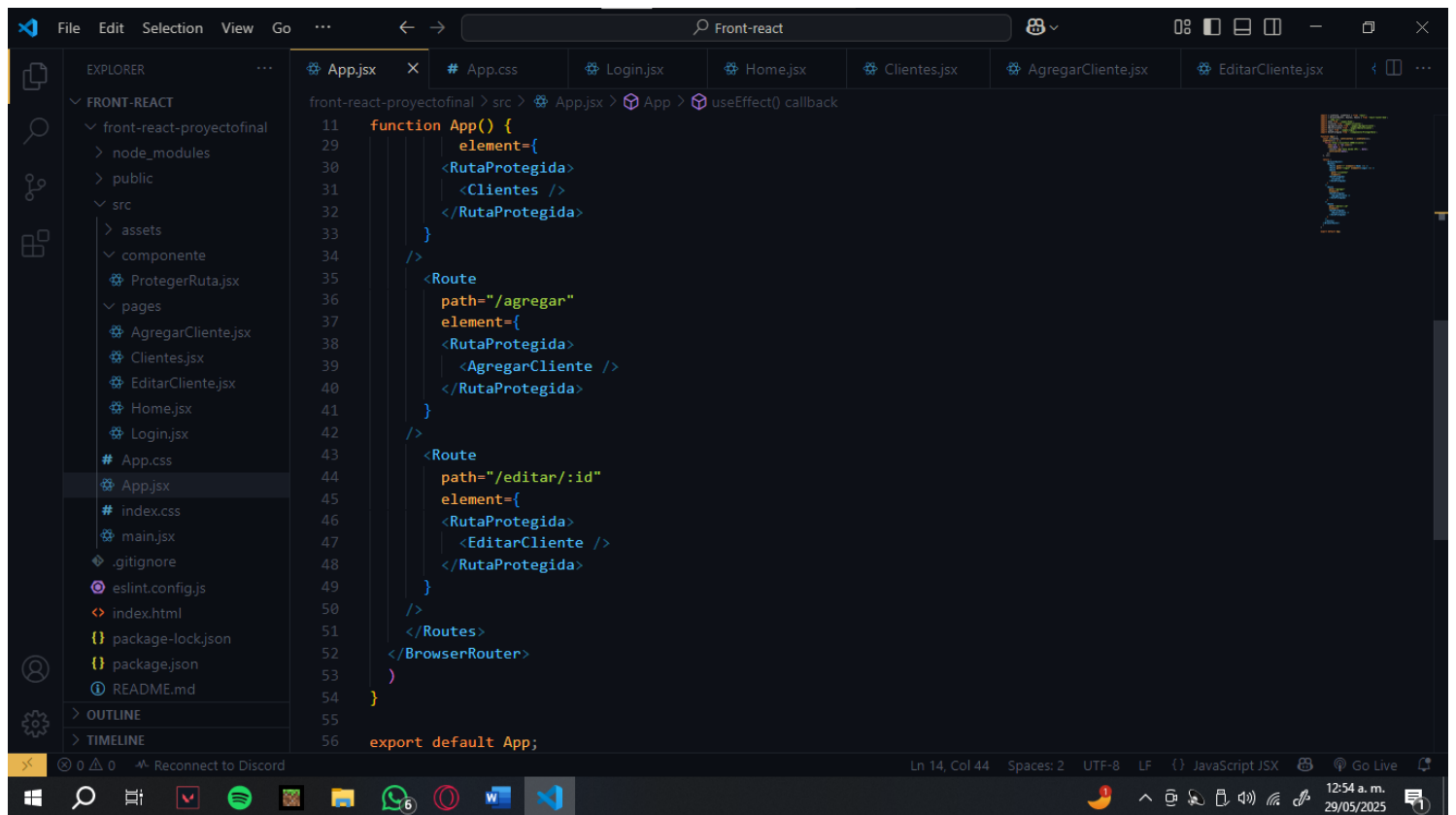


```
1 import { useState, useEffect } from 'react';
2 import { BrowserRouter, Routes, Route } from 'react-router-dom';
3 import './App.css';
4 import Home from './pages/Home';
5 import Clientes from './pages/Clientes';
6 import AgregarCliente from './pages/AgregarCliente';
7 import EditarCliente from './pages/EditarCliente';
8 import Login from './pages/Login';
9 import RutaProtegida from './componente/ProtegerRuta';
10
11 function App() {
12   const [clientes, setClientes] = useState([]);
13   useEffect(() => {
14     fetch('http://localhost:3000/Clientes')
15       .then(res => res.json())
16       .then(data => {
17         console.log('data desde API:', data);
18         setClientes(data);
19       });
20   }, []);
21
22   return (
23     <BrowserRouter>
24     <Routes>
25       <Route path="/" element={<Home />} />
26       <Route path="/login" element={<Login />} />
27       <Route
28         path="/clientes"
29         element={
```

useState guarda los datos de los clientes (API).

useEffect se usa para **cargar los datos desde el backend** al iniciar el proyecto.

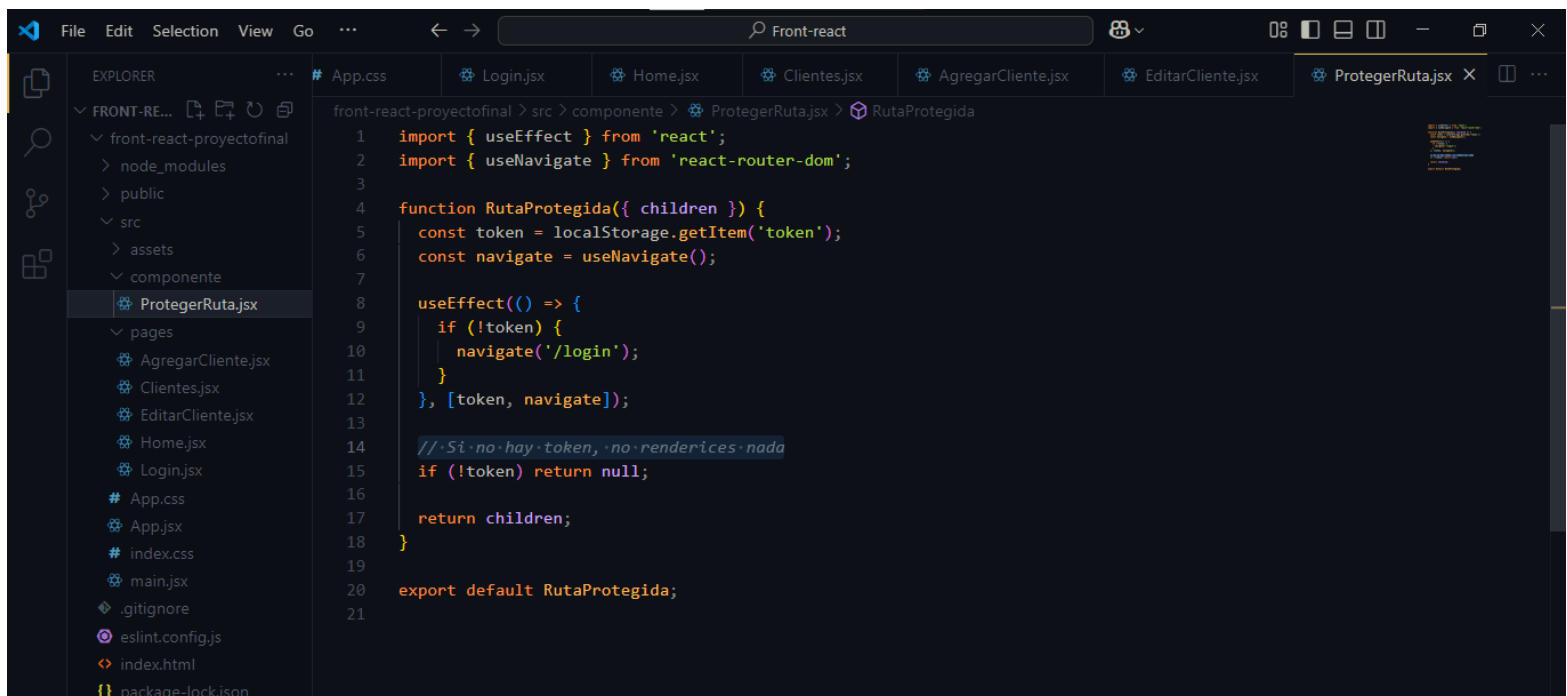
Las rutas con react (Semestre pasado): Activa el sistema de navegación de páginas y Permite usar rutas como /, /login, /clientes, etc. (import { BrowserRouter, Routes, Route } from 'react-router-dom';)



```
11 function App() {
12   element={
13     <RutaProtegida>
14     <Clientes />
15   </RutaProtegida>
16 }
17 />
18 <Route
19   path="/agregar"
20   element={
21     <RutaProtegida>
22     <AgregarCliente />
23   </RutaProtegida>
24 }
25 />
26 <Route
27   path="/editar/:id"
28   element={
29     <RutaProtegida>
30     <EditarCliente />
31   </RutaProtegida>
32 }
33 />
34 </Routes>
35 </BrowserRouter>
36 )
37 }
38 export default App;
```

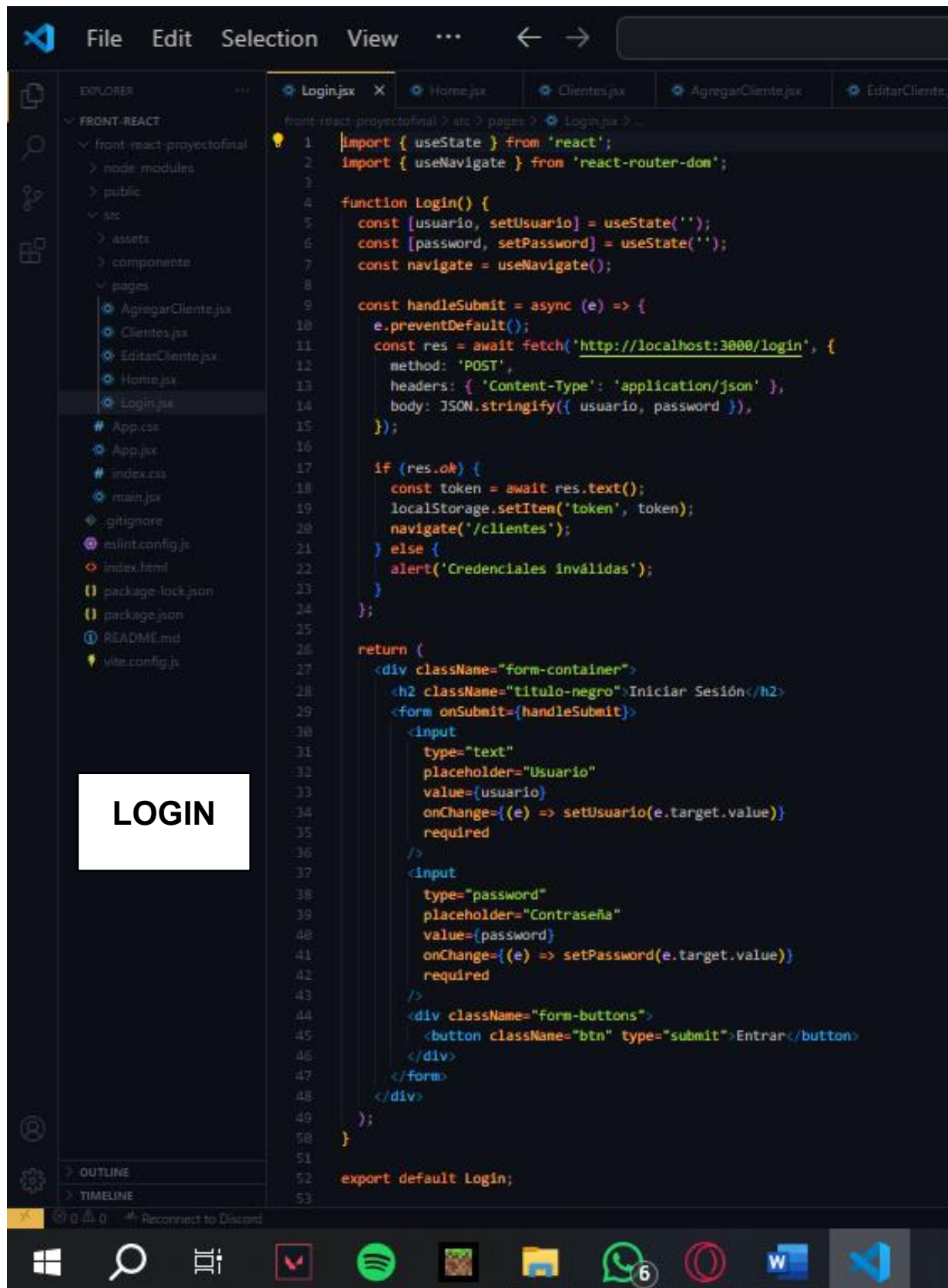
El import RutaProtegida from './componente/ProtegerRuta';

Lo agregue como componente que **bloquea rutas privadas** cuando no tienes token JWT. Evita que se acceda sin iniciar sesión.



```
1 import { useEffect } from 'react';
2 import { useNavigate } from 'react-router-dom';
3
4 function RutaProtegida({ children }) {
5   const token = localStorage.getItem('token');
6   const navigate = useNavigate();
7
8   useEffect(() => {
9     if (!token) {
10       navigate('/login');
11     }
12   }, [token, navigate]);
13
14   //Si no hay token, no renderices nada
15   if (!token) return null;
16
17   return children;
18 }
19
20 export default RutaProtegida;
```


Y pues la función de la app.jsx como tal Muestra las diferentes pantallas de la app según la URL. Protege las rutas sensibles y carga los componentes necesarios.



```
1  import { useState } from 'react';
2  import { useNavigate } from 'react-router-dom';
3
4  function Login() {
5    const [usuario, setUsuario] = useState('');
6    const [password, setPassword] = useState('');
7    const navigate = useNavigate();
8
9    const handleSubmit = async (e) => {
10     e.preventDefault();
11     const res = await fetch('http://localhost:3000/login', {
12       method: 'POST',
13       headers: { 'Content-Type': 'application/json' },
14       body: JSON.stringify({ usuario, password }),
15     });
16
17     if (res.ok) {
18       const token = await res.text();
19       localStorage.setItem('token', token);
20       navigate('/clientes');
21     } else {
22       alert('Credenciales inválidas');
23     }
24   };
25
26   return (
27     <div className="form-container">
28       <h2 className="titulo-negro">Iniciar Sesión</h2>
29       <form onSubmit={handleSubmit}>
30         <input
31           type="text"
32           placeholder="Usuario"
33           value={usuario}
34           onChange={(e) => setUsuario(e.target.value)}
35           required
36         />
37         <input
38           type="password"
39           placeholder="Contraseña"
40           value={password}
41           onChange={(e) => setPassword(e.target.value)}
42           required
43         />
44         <div className="form-buttons">
45           <button className="btn" type="submit">Entrar</button>
46         </div>
47       </form>
48     </div>
49   );
50 }
51
52 export default Login;
```

LOGIN

Ok en el useState y useNavigate:

```
const [usuario, setUsuario] = useState("");
```

```
const [password, setPassword] = useState("");
```

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

en esta parte **useState** guarda lo que el usuario escribe en los campos. Y el **useNavigate** como tal nos permite **redirigir** al usuario a otra página después de iniciar sesión.

El handleSubmit:

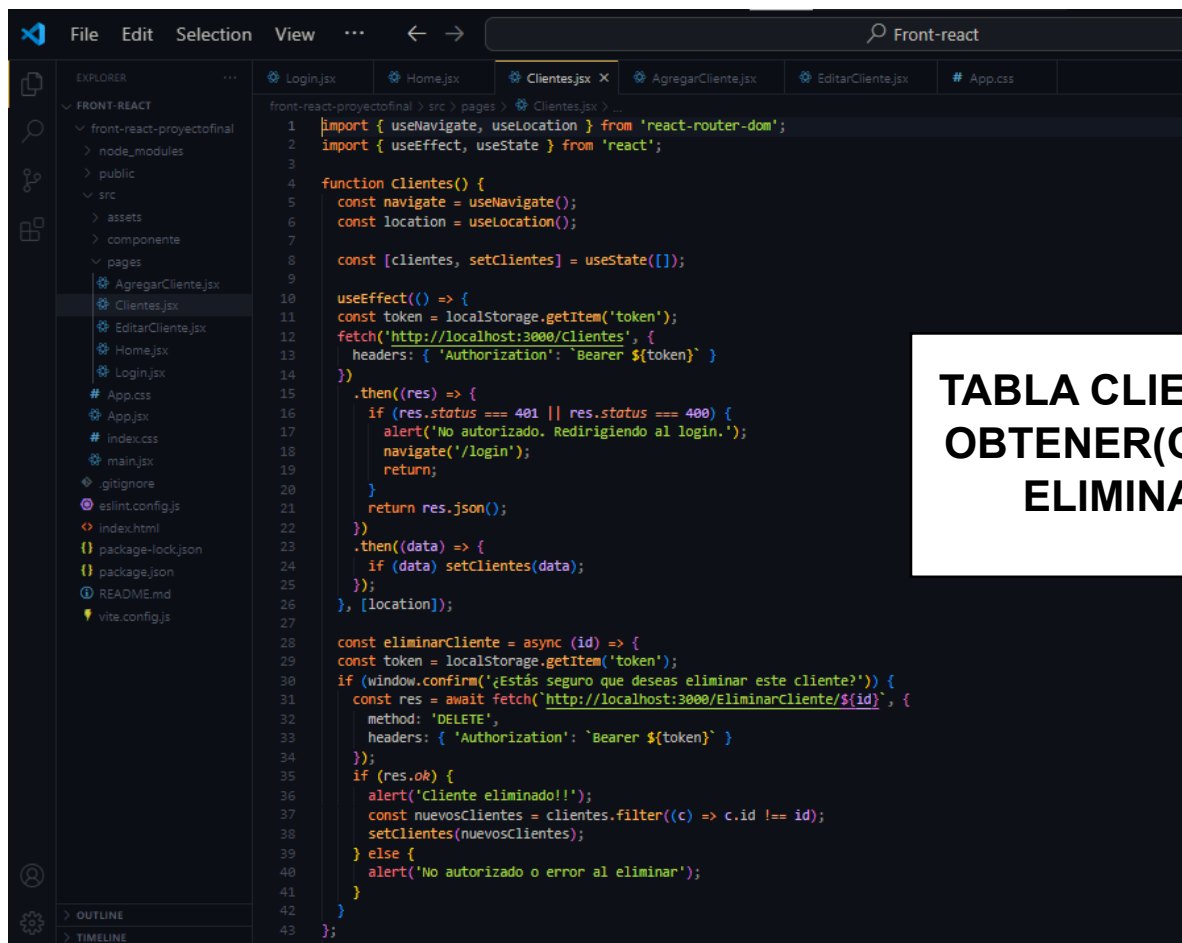
```
const res = await fetch('http://localhost:3000/login', { ... });
```

Cuando el usuario envía el formulario:

-Se hace una petición POST al backend con el usuario y contraseña.

-Si son correctos, se guarda el token y se redirige a /clientes.

-Si no, muestra una alerta de "Credenciales inválidas".



```
1 import { useNavigate, useLocation } from 'react-router-dom';
2 import { useEffect, useState } from 'react';
3
4 function Clientes() {
5   const navigate = useNavigate();
6   const location = useLocation();
7
8   const [clientes, setClientes] = useState([]);
9
10  useEffect(() => {
11    const token = localStorage.getItem('token');
12    fetch('http://localhost:3000/Clientes', {
13      headers: { 'Authorization': 'Bearer ${token}' }
14    })
15      .then((res) => {
16        if (res.status === 401 || res.status === 400) {
17          alert('No autorizado. Redirigiendo al login.');
```

**TABLA CLIENTES,
OBTENER(GET) Y
ELIMINAR**

useState, useNavigate, useLocation:

-const [clientes, setClientes] = useState([]);

-const navigate = useNavigate();

-const location = useLocation();

- Guarda la lista de clientes.
- “Navigate” redirige si no hay token.
- “location” hace que el useEffect se actualice cuando cambias de ruta.

useEffect (cargar clientes):

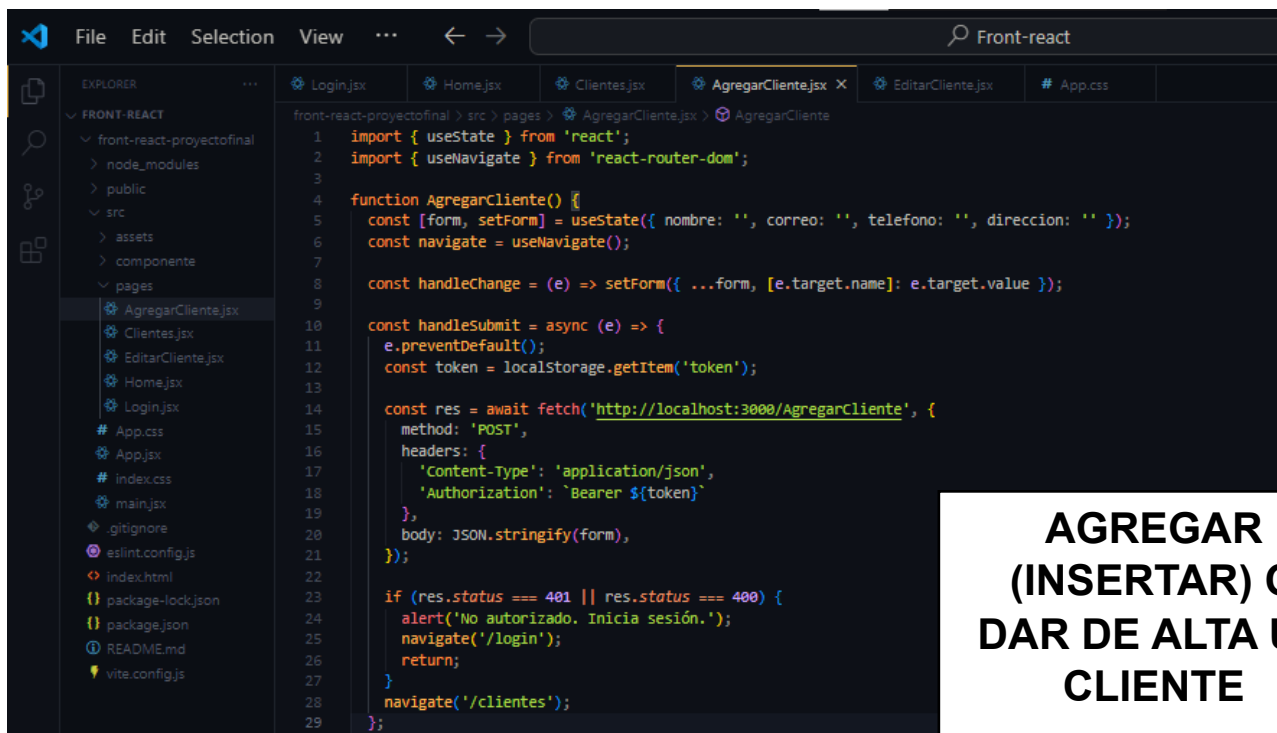
fetch('http://localhost:3000/Clientes', { headers: { Authorization } })

-Cuando el componente carga, **hace una petición al backend** para obtener todos los clientes. Si no hay token válido, **redirige al login**.

EliminarCliente

fetch('http://localhost:3000/EliminarCliente/\${id}', { method: 'DELETE' })

-Si el usuario confirma, se manda una petición para **eliminar al cliente por su ID**. Después actualiza la lista sin recargar la página.



```
1 import { useState } from 'react';
2 import { useNavigate } from 'react-router-dom';
3
4 function AgregarCliente() {
5   const [form, setForm] = useState({ nombre: '', correo: '', telefono: '', direccion: '' });
6   const navigate = useNavigate();
7
8   const handleChange = (e) => setForm({ ...form, [e.target.name]: e.target.value });
9
10  const handleSubmit = async (e) => {
11    e.preventDefault();
12    const token = localStorage.getItem('token');
13
14    const res = await fetch('http://localhost:3000/AgregarCliente', {
15      method: 'POST',
16      headers: {
17        'Content-Type': 'application/json',
18        'Authorization': `Bearer ${token}`
19      },
20      body: JSON.stringify(form),
21    });
22
23    if (res.status === 401 || res.status === 400) {
24      alert('No autorizado. Inicia sesión.');
```

**AGREGAR
(INSERTAR) O
DAR DE ALTA UN
CLIENTE**

useState, useNavigate:

```
-const [form, setForm] = useState({ nombre: "", correo: "", telefono: "", direccion: "" });
```

```
-const navigate = useNavigate();
```

- Guarda los datos del formulario para agregar un cliente (nombre, correo, teléfono, dirección).
- “Navigate” permite redirigir a otra ruta según el resultado de la acción.

handleChange:

```
-const handleChange = (e) => setForm({ ...form, [e.target.name]: e.target.value });
```

- Actualiza el estado form cada vez que el usuario escribe en algún input del formulario, manteniendo los valores anteriores.

handleSubmit (enviar datos al backend):

```
const res = await fetch('http://localhost:3000/AgregarCliente', { method: 'POST', headers: { 'Content-Type': 'application/json', 'Authorization': Bearer ${token} }, body: JSON.stringify(form), });
```

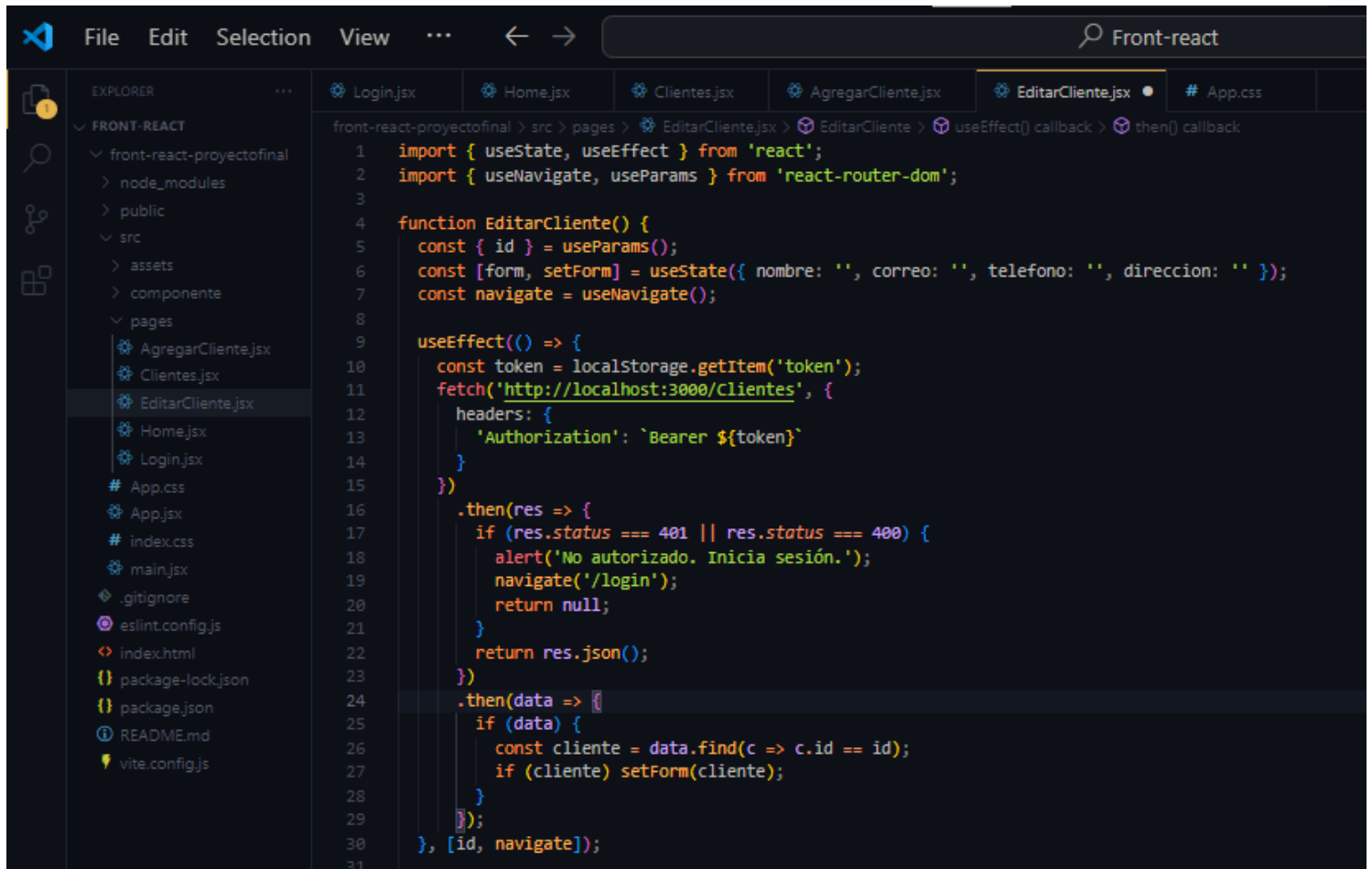
- Cuando el formulario se envía, previene la acción por defecto y hace una petición POST al backend para agregar un cliente.
- Envía los datos del formulario como JSON junto con un token JWT en la cabecera para autenticación.

Manejo de respuesta:

```
if (res.status === 401 || res.status === 400) { alert('No autorizado. Inicia sesión. '); navigate('/login'); return; }
```

```
navigate('/clientes');
```

- Si el backend responde que el token no es válido o hay error (401 o 400), muestra alerta y redirige al login.
- Si todo va bien, redirige a la lista de clientes.



```
1 import { useState, useEffect } from 'react';
2 import { useNavigate, useParams } from 'react-router-dom';
3
4 function EditarCliente() {
5   const { id } = useParams();
6   const [form, setForm] = useState({ nombre: '', correo: '', telefono: '', direccion: '' });
7   const navigate = useNavigate();
8
9   useEffect(() => {
10     const token = localStorage.getItem('token');
11     fetch('http://localhost:3000/Clientes', {
12       headers: {
13         'Authorization': `Bearer ${token}`
14       }
15     })
16     .then(res => {
17       if (res.status === 401 || res.status === 400) {
18         alert('No autorizado. Inicia sesión.');
```

```
const handleChange = (e) => setForm({ ...form, [e.target.name]: e.target.value });
const handleSubmit = async (e) => {
  e.preventDefault();
  const token = localStorage.getItem('token');
  const res = await fetch(`http://localhost:3000/EditarCliente/${id}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${token}`
    },
    body: JSON.stringify(form),
  });

  if (res.status === 401 || res.status === 400) {
    alert('No autorizado. Inicia sesión.');
```

**ACTUALIZAR(EDITAR)
A UN CLIENTE**

useParams, useState, useNavigate:

-const { id } = useParams();

-const [form, setForm] = useState({ nombre: "", correo: "", telefono: "", -direccion: " " });

-const navigate = useNavigate();

- Obtiene el id del cliente desde la URL para saber cuál editar.
- Guarda los datos del cliente en el formulario para editar.
- Permite redirigir a otras rutas según el resultado.

useEffect (cargar cliente):

- Hace una petición al backend para obtener todos los clientes con el token de autorización.
- Si el token no es válido (401 o 400), muestra alerta y redirige a login.
- Si es válido, busca el cliente con el id indicado y carga sus datos en el formulario.
- El efecto se ejecuta cuando cambia el id o el navigate.

handleChange (actualizar formulario):

- Actualiza el estado form conforme el usuario escribe en los inputs, manteniendo los datos previos.

handleSubmit (enviar edición):

- Previene el envío por defecto del formulario.
- Hace una petición PUT al backend para actualizar el cliente con el id y los datos del formulario.
- Envía el token JWT en la cabecera para autorización.
- Si la respuesta es 401 o 400, alerta y redirige a login.
- Si todo sale bien, redirige a la lista de clientes.