



PORTADA

Evidencia

1. Diseño y Desarrollo de servicios web - caso GA7-220501096-AA5-EV01
2. API GA7-220501096-AA5-EV02
3. Diseño y Desarrollo de servicios web – proyecto GA7-220501096-AA5-EV03
4. API del Proyecto GA7-220501096-AA5-EV04

Instructor: Esteban Cadena

Aprendiz: Julián Mosquera

Repositorio GitHub

Enlace de la carpeta del backend: <https://github.com/julianfox2016/project-root.git>

Enlace del frontend: https://github.com/julianfox2016/proyecto_react.git

Institución: Sena

Fecha: 04-05-2025

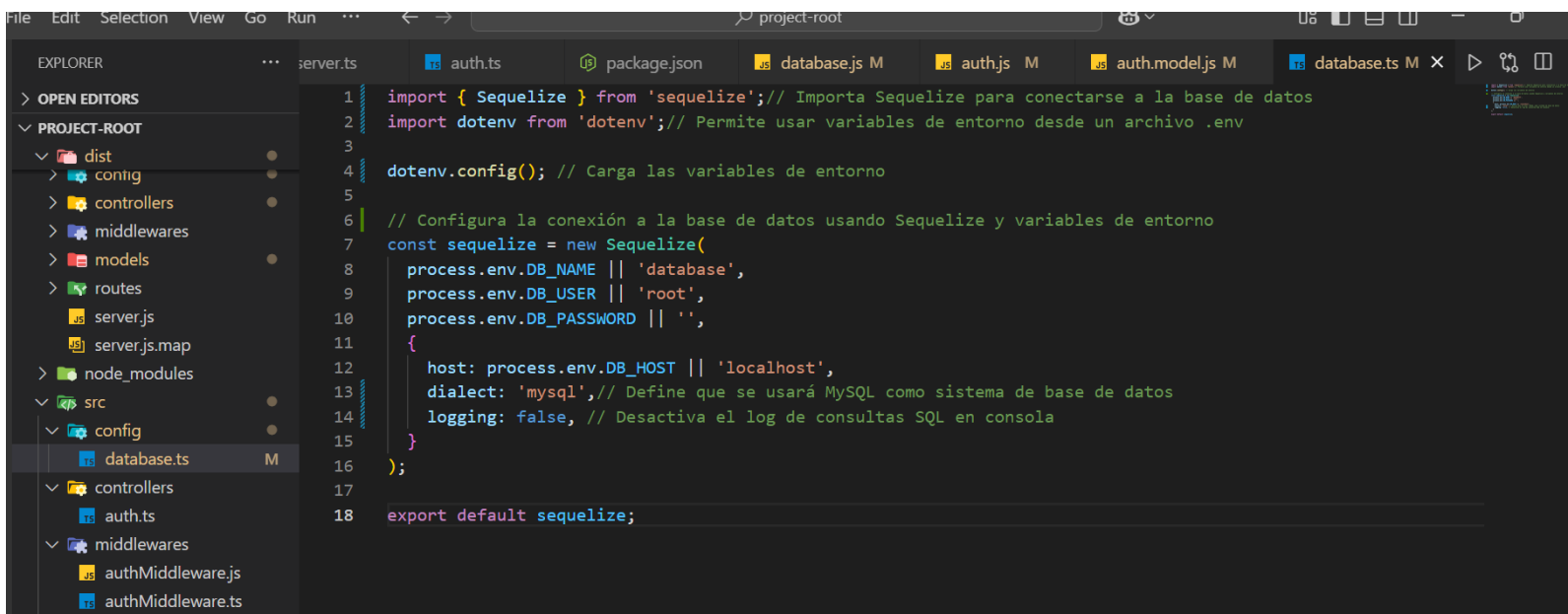
INTRODUCION

En el marco del componente formativo se plantea una serie de actividades que nos permitirán Aplicar los conocimientos adquiridos en el componente estas evidencias tiene como propósito guiarnos en la construcción de desarrollo web funcionales mediante servicios estructurados, seguros y aprobados abordando desde los aspectos básicos de autenticación hasta la implementación completas de APIs en nuestro proyecto formativo, que consiste en el diseño y codificación de un usuario web enfocado en el registro e inicio de sesión de usuario, integrando validaciones de autenticación y utilizando herramientas de control de versiones. Además, se hará pruebas funcionales utilizando la herramienta de Postman, en esta etapa se valida el comportamiento de los servicios creados.

Explicación del Código para Conexión a Base de Datos con Sequelize

Este código configura una conexión de la base de datos MySQL usando sequelize, con parámetros configurables mediante variables de entorno (almacenada en un archivo .env) proporciona valores por defecto para desarrollo local y permite la fácil configuración para diferentes entornos

Config/Database.ts



```
1 import { Sequelize } from 'sequelize'; // Importa Sequelize para conectarse a la base de datos
2 import dotenv from 'dotenv'; // Permite usar variables de entorno desde un archivo .env
3
4 dotenv.config(); // Carga las variables de entorno
5
6 // Configura la conexión a la base de datos usando Sequelize y variables de entorno
7 const sequelize = new Sequelize(
8   process.env.DB_NAME || 'database',
9   process.env.DB_USER || 'root',
10  process.env.DB_PASSWORD || '',
11  {
12    host: process.env.DB_HOST || 'localhost',
13    dialect: 'mysql', // Define que se usará MySQL como sistema de base de datos
14    logging: false, // Desactiva el log de consultas SQL en consola
15  }
16 );
17
18 export default sequelize;
```

Explicación del Código Sistema de Autenticación

Este código implementa un sistema básico de autenticación utilizando un modelo de usuario para validar credenciales, manejando tanto el registro como el inicio de sesión. Este sistema está diseñado para integrarse a una API, recibiendo y procesando datos mediante solicitudes HTTP, proporcionando respuesta adecuadas según el resultado de la autenticación.

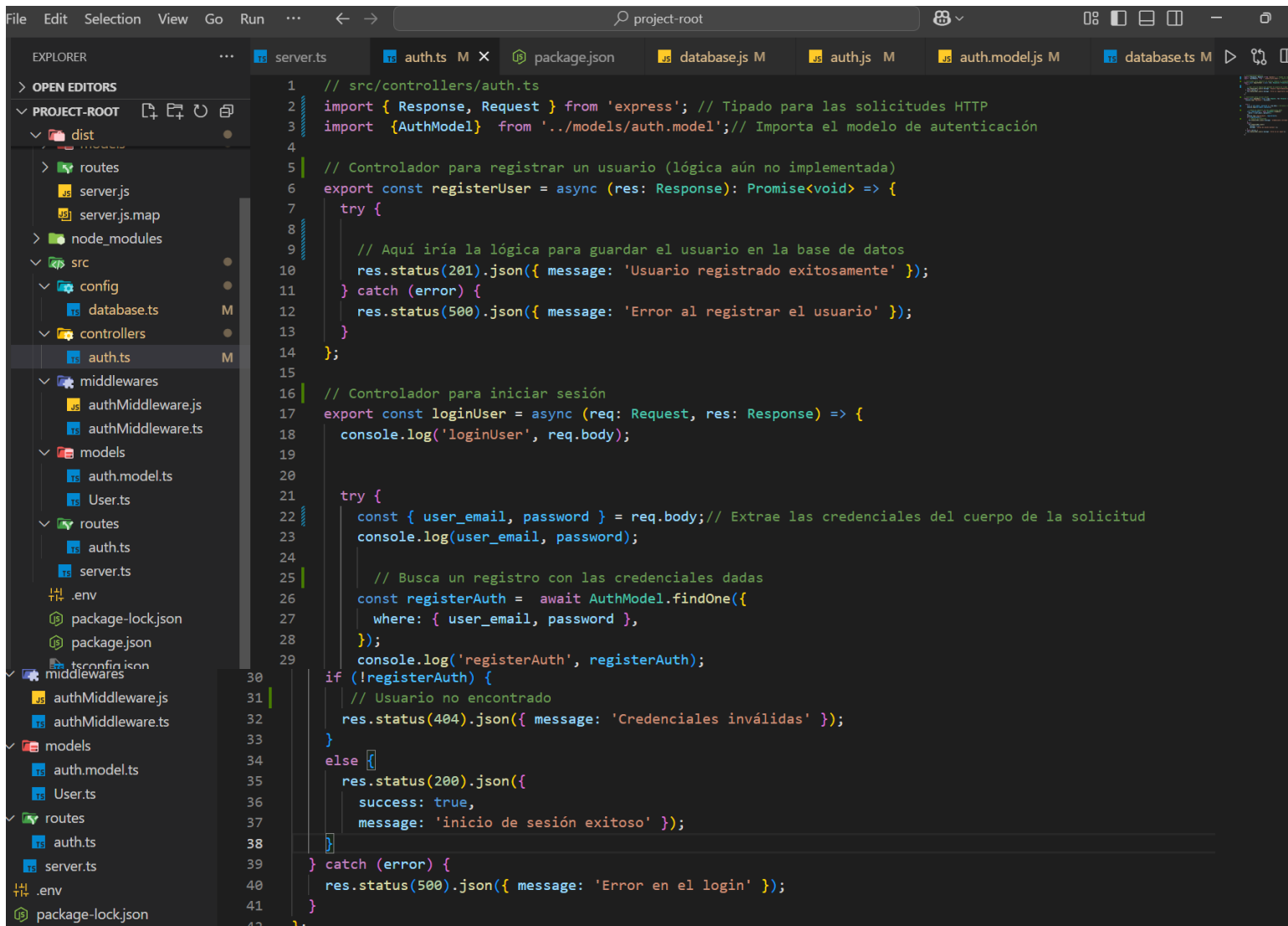
Aspecto clave

Conexión con la base de datos

Manejo de errores

Escalabilidad

Controllers/auth.ts

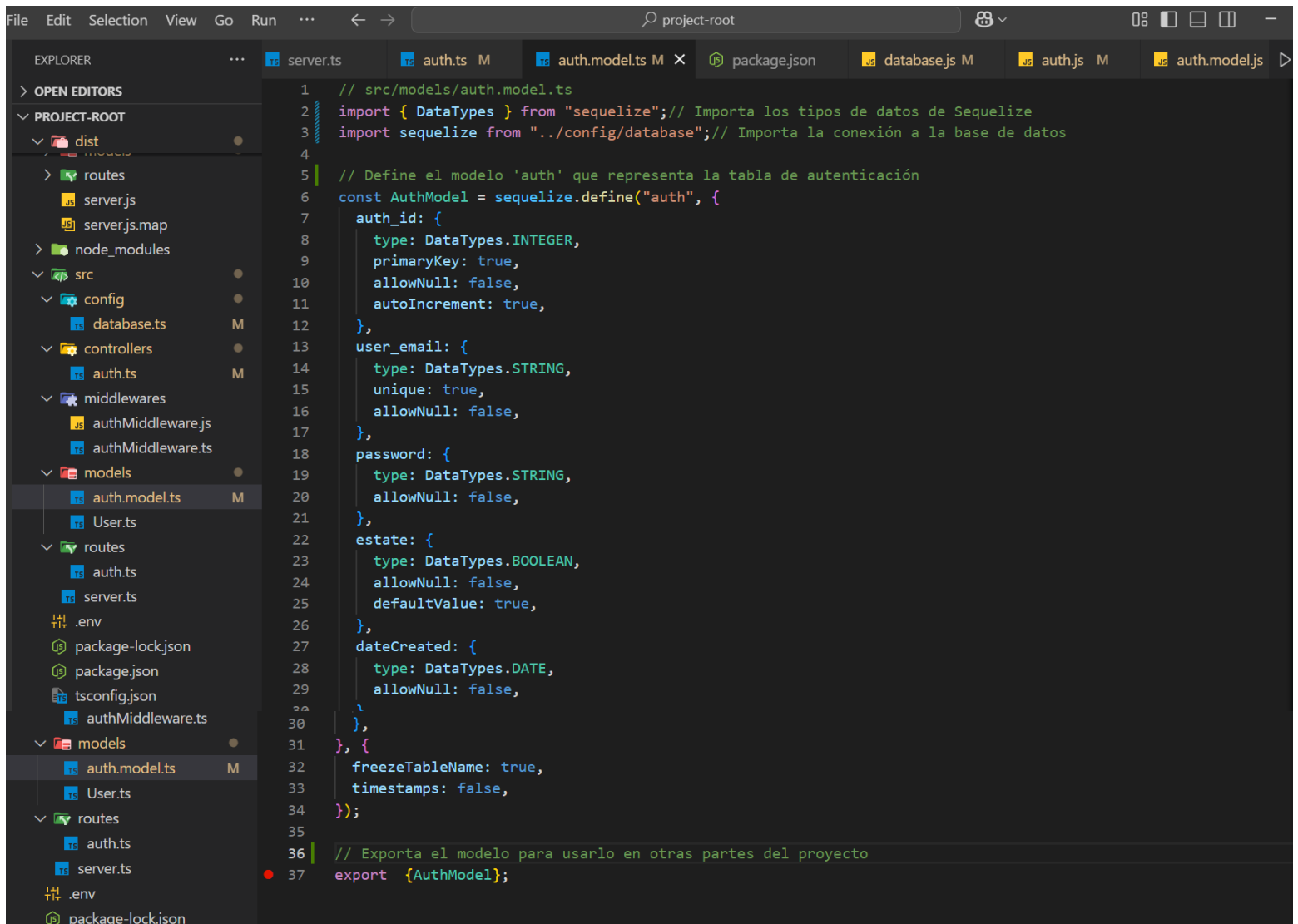
The image shows a code editor with a sidebar on the left displaying a file explorer. The main area shows the content of the file 'auth.ts' under the 'controllers' directory. The code is written in TypeScript and implements two functions: 'registerUser' and 'loginUser'. The 'registerUser' function is currently a placeholder, while 'loginUser' is fully implemented, including database lookups and response handling. The editor has a dark theme and multiple tabs are open at the top.

```
1 // src/controllers/auth.ts
2 import { Response, Request } from 'express'; // Tipado para las solicitudes HTTP
3 import { AuthModel } from '../models/auth.model'; // Importa el modelo de autenticación
4
5 // Controlador para registrar un usuario (lógica aún no implementada)
6 export const registerUser = async (res: Response): Promise<void> => {
7   try {
8     // Aquí iría la lógica para guardar el usuario en la base de datos
9     res.status(201).json({ message: 'Usuario registrado exitosamente' });
10    } catch (error) {
11      res.status(500).json({ message: 'Error al registrar el usuario' });
12    }
13  };
14
15 // Controlador para iniciar sesión
16 export const loginUser = async (req: Request, res: Response) => {
17   console.log('loginUser', req.body);
18
19   try {
20     const { user_email, password } = req.body; // Extrae las credenciales del cuerpo de la solicitud
21     console.log(user_email, password);
22
23     // Busca un registro con las credenciales dadas
24     const registerAuth = await AuthModel.findOne({
25       where: { user_email, password },
26     });
27     console.log('registerAuth', registerAuth);
28     if (!registerAuth) {
29       // Usuario no encontrado
30       res.status(404).json({ message: 'Credenciales inválidas' });
31     } else {
32       res.status(200).json({
33         success: true,
34         message: 'inicio de sesión exitoso' });
35     }
36   } catch (error) {
37     res.status(500).json({ message: 'Error en el login' });
38   }
39 };
```

Modelo de Autenticación con Sequelize

Este archivo define un modelo llamado AuthModel, usando sequelize, una biblioteca de node.js que facilita la interacción con bases de datos como MySQL, que representa una tabla de autenticación de usuario con los campos básico: ID, Correo, Contraseña, Estado y Fecha de creación

Models/authmodels.ts

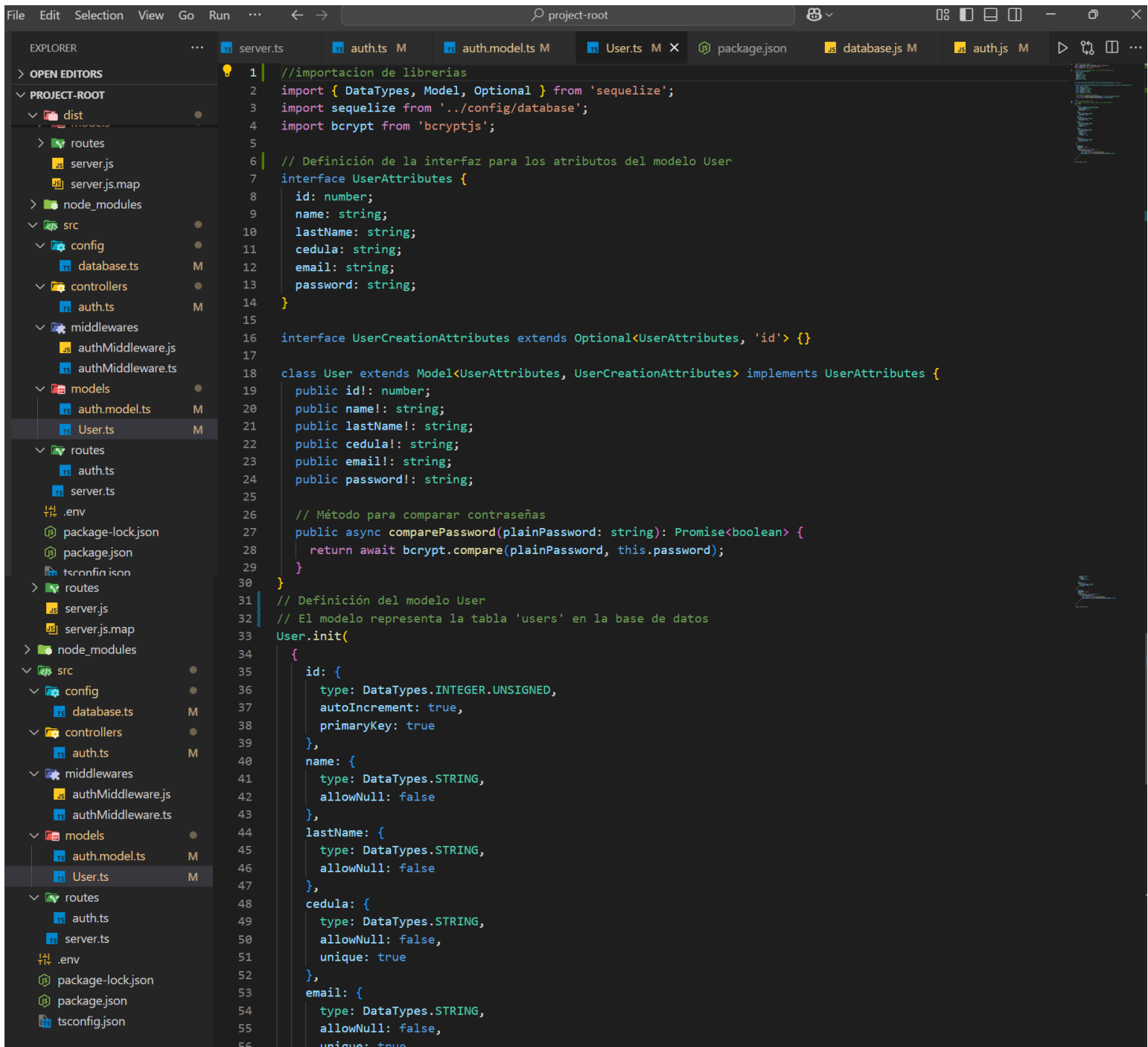


```
1 // src/models/auth.model.ts
2 import { DataTypes } from "sequelize"; // Importa los tipos de datos de Sequelize
3 import sequelize from "../config/database"; // Importa la conexión a la base de datos
4
5 // Define el modelo 'auth' que representa la tabla de autenticación
6 const AuthModel = sequelize.define("auth", {
7   auth_id: {
8     type: DataTypes.INTEGER,
9     primaryKey: true,
10    allowNull: false,
11    autoIncrement: true,
12  },
13  user_email: {
14    type: DataTypes.STRING,
15    unique: true,
16    allowNull: false,
17  },
18  password: {
19    type: DataTypes.STRING,
20    allowNull: false,
21  },
22  estate: {
23    type: DataTypes.BOOLEAN,
24    allowNull: false,
25    defaultValue: true,
26  },
27  dateCreated: {
28    type: DataTypes.DATE,
29    allowNull: false,
30  },
31 }, {
32   freezeTableName: true,
33   timestamps: false,
34 });
35
36 // Exporta el modelo para usarlo en otras partes del proyecto
37 export {AuthModel};
```

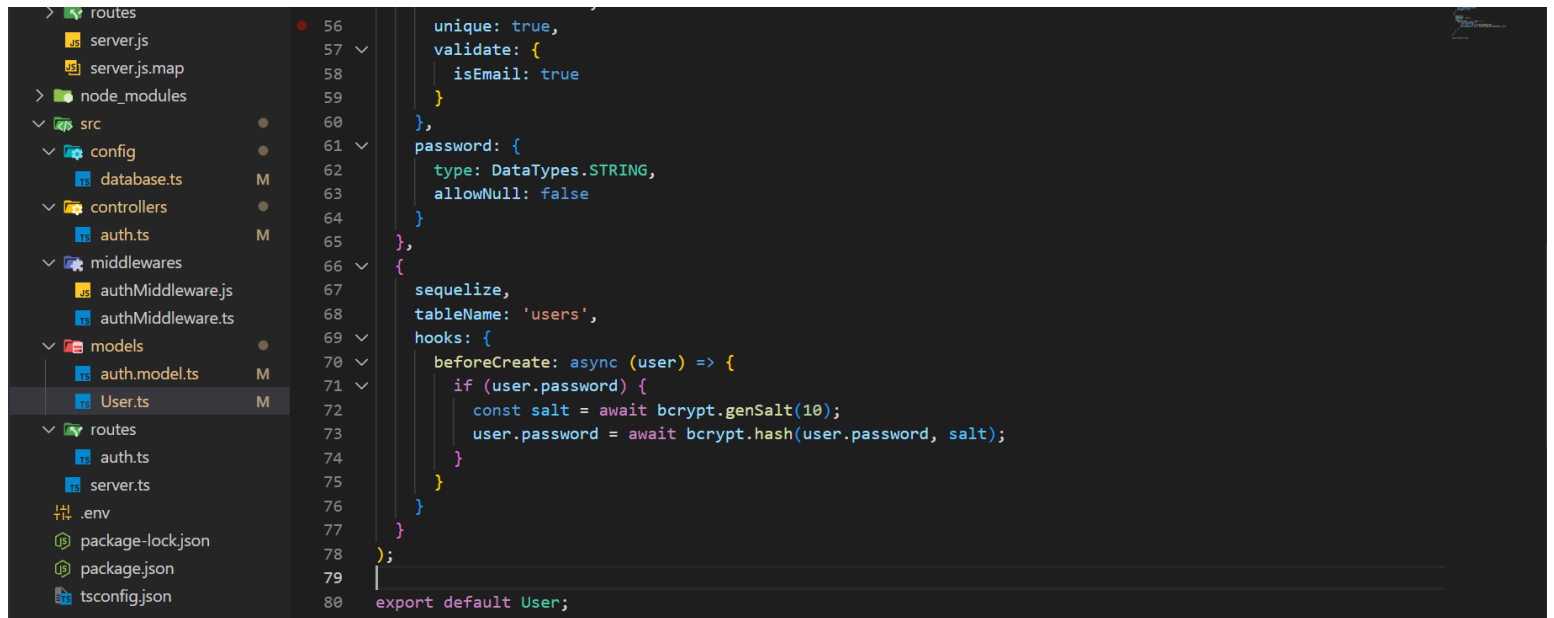
Modelo de usuario

Este código define un modelo de usuario 'User' usando sequelize, para gestionar usuarios en una base de datos MySQL utilizando TypeScript para tipar correctamente los atributos del modelo. Además, valida los campos importantes como el correo electrónico, cifra automáticamente la contraseña antes de guardar un usuario. Incluye un método para comparar contraseñas usando bcrypt útil durante el login.

Models/User.ts



```
1 //importacion de librerias
2 import { DataTypes, Model, Optional } from 'sequelize';
3 import sequelize from '../config/database';
4 import bcrypt from 'bcryptjs';
5
6 // Definición de la interfaz para los atributos del modelo User
7 interface UserAttributes {
8   id: number;
9   name: string;
10  lastName: string;
11  cedula: string;
12  email: string;
13  password: string;
14 }
15
16 interface UserCreationAttributes extends Optional<UserAttributes, 'id'> {}
17
18 class User extends Model<UserAttributes, UserCreationAttributes> implements UserAttributes {
19   public id!: number;
20   public name!: string;
21   public lastName!: string;
22   public cedula!: string;
23   public email!: string;
24   public password!: string;
25
26   // Método para comparar contraseñas
27   public async comparePassword(plainPassword: string): Promise<boolean> {
28     return await bcrypt.compare(plainPassword, this.password);
29   }
30 }
31
32 // Definición del modelo User
33 // El modelo representa la tabla 'users' en la base de datos
34 User.init(
35   {
36     id: {
37       type: DataTypes.INTEGER.UNSIGNED,
38       autoIncrement: true,
39       primaryKey: true
40     },
41     name: {
42       type: DataTypes.STRING,
43       allowNull: false
44     },
45     lastName: {
46       type: DataTypes.STRING,
47       allowNull: false
48     },
49     cedula: {
50       type: DataTypes.STRING,
51       allowNull: false,
52       unique: true
53     },
54     email: {
55       type: DataTypes.STRING,
56       allowNull: false,
57       unique: true
```



Explicación Detallada del Código del Servidor Backend

Este código configura un servidor backend completo usando Node.js Con Express y Sequelize con autenticación realiza lo siguiente

Cargar variable de entorno usando dotenv permitiendo una configuración flexible del entorno.

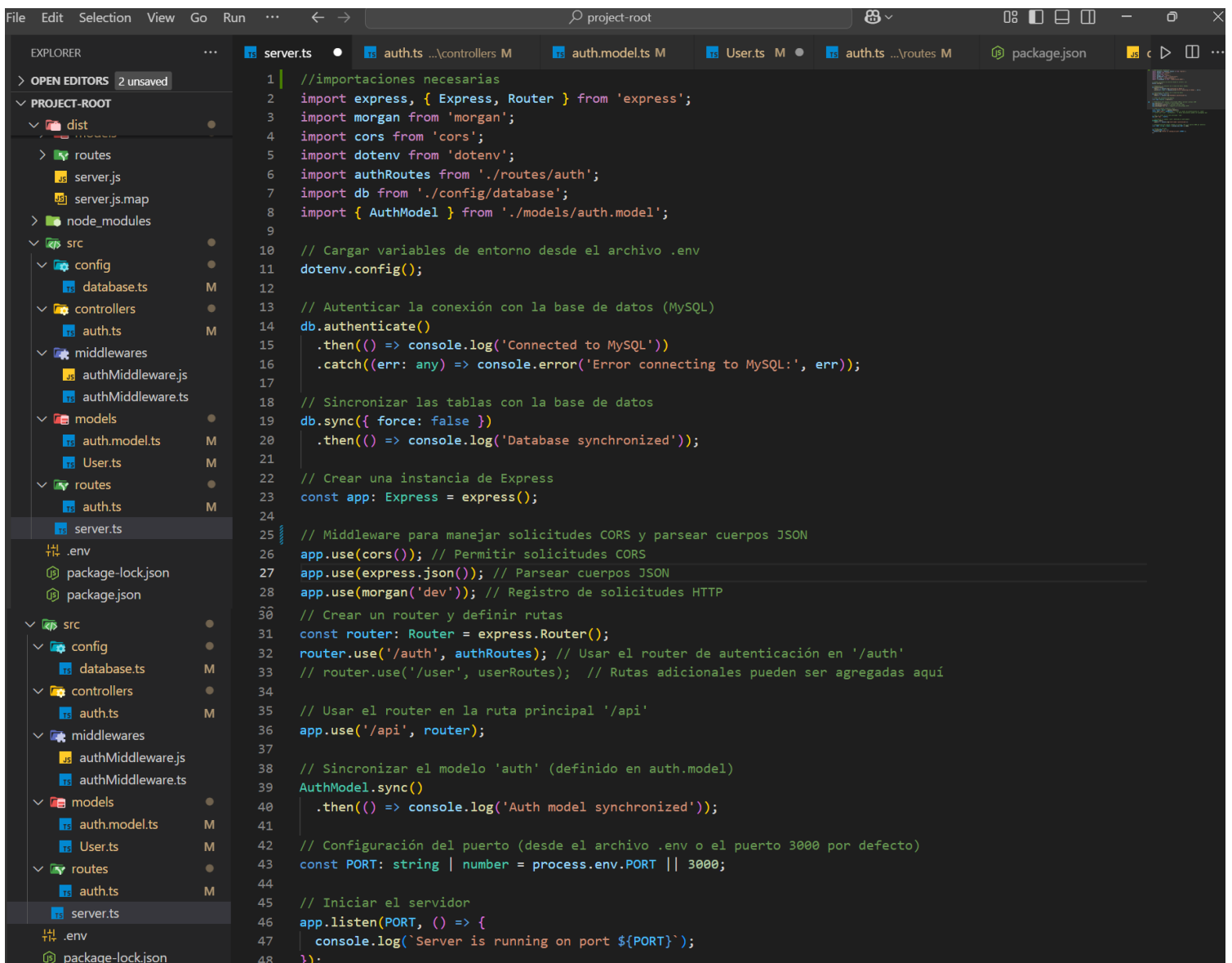
Establece conexión con MySQL a través de Sequelize y sincroniza los modelos.

Crea una aplicación Express y configuración middlewares clave como cors. express.json() y morgan para manejar solicitudes, seguridad y registros.

Define rutas de autenticación bajo el prefijo /api/auth, conectando el Router authRoutes con la aplicación.

Inicia el servidor escuchando el puerto configurado, mostrando un mensaje de confirmación en consola.

Routes/server.ts



```
1 //importaciones necesarias
2 import express, { Express, Router } from 'express';
3 import morgan from 'morgan';
4 import cors from 'cors';
5 import dotenv from 'dotenv';
6 import authRoutes from './routes/auth';
7 import db from './config/database';
8 import { AuthModel } from './models/auth.model';
9
10 // Cargar variables de entorno desde el archivo .env
11 dotenv.config();
12
13 // Autenticar la conexión con la base de datos (MySQL)
14 db.authenticate()
15   .then(() => console.log('Connected to MySQL'))
16   .catch((err: any) => console.error('Error connecting to MySQL:', err));
17
18 // Sincronizar las tablas con la base de datos
19 db.sync({ force: false })
20   .then(() => console.log('Database synchronized'));
21
22 // Crear una instancia de Express
23 const app: Express = express();
24
25 // Middleware para manejar solicitudes CORS y parsear cuerpos JSON
26 app.use(cors()); // Permitir solicitudes CORS
27 app.use(express.json()); // Parsear cuerpos JSON
28 app.use(morgan('dev')); // Registro de solicitudes HTTP
29
30 // Crear un router y definir rutas
31 const router: Router = express.Router();
32 router.use('/auth', authRoutes); // Usar el router de autenticación en '/auth'
33 // router.use('/user', userRoutes); // Rutas adicionales pueden ser agregadas aquí
34
35 // Usar el router en la ruta principal '/api'
36 app.use('/api', router);
37
38 // Sincronizar el modelo 'auth' (definido en auth.model)
39 AuthModel.sync()
40   .then(() => console.log('Auth model synchronized'));
41
42 // Configuración del puerto (desde el archivo .env o el puerto 3000 por defecto)
43 const PORT: string | number = process.env.PORT || 3000;
44
45 // Iniciar el servidor
46 app.listen(PORT, () => {
47   console.log(`Server is running on port ${PORT}`);
48 });
```

Explicación de la interfaz de usuario

Este componente implementa un sistema completo de autenticación en el frontend que:

Gestiona el flujo del login mediante:

Validación de formato de credenciales

Comunicación con el backend

Manejo de respuesta exitosas/fallidas

Integra con ecosistema de la aplicación

Utiliza un contexto global para mantener un estado de autenticación

Se conecta con el sistema de rutas para redirecciones

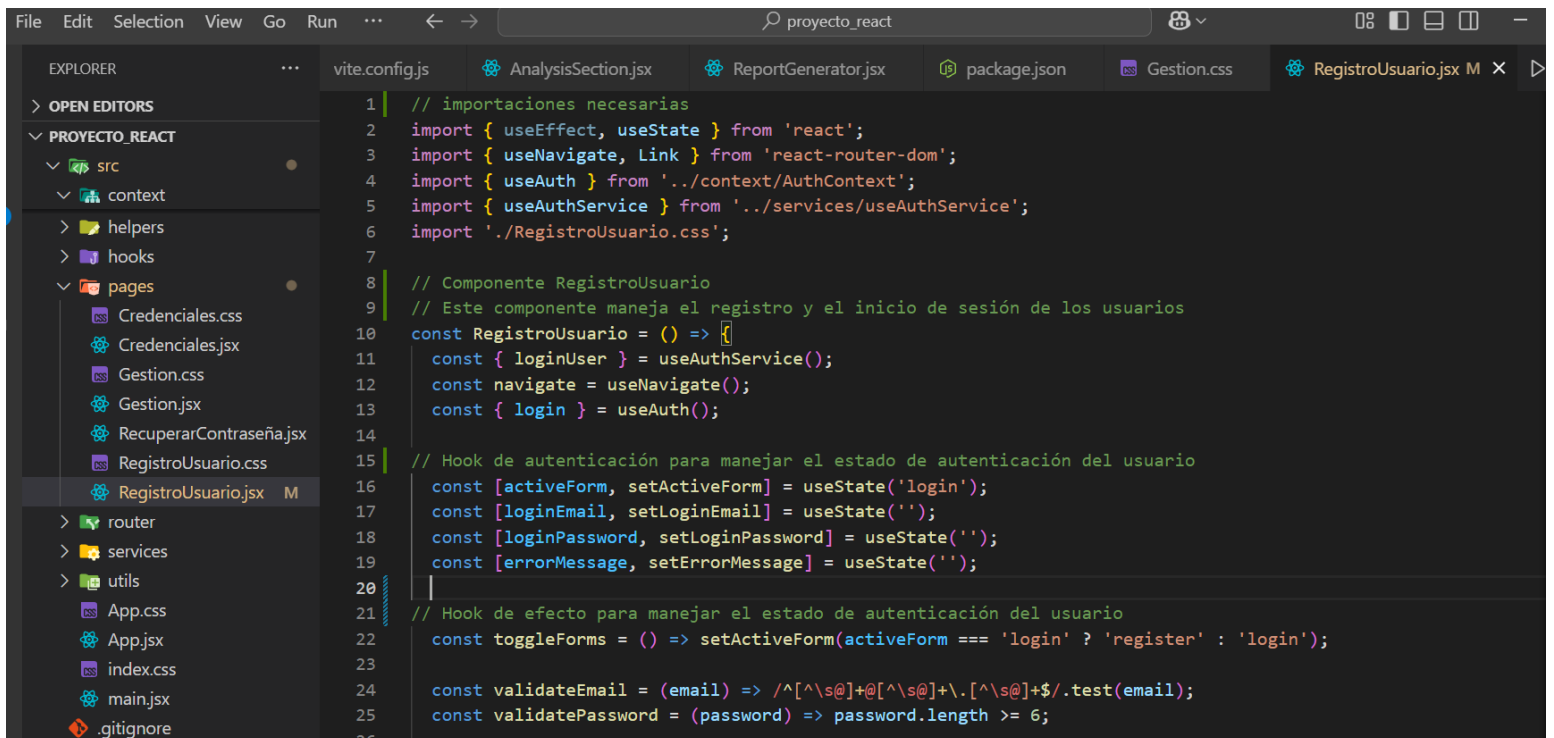
Implementa medidas de seguridad básicas

Validación

Protección contra errores mediante try/catch

No almacena credenciales localmente

pages/RegistroUsuarios.jsx



```
1 // importaciones necesarias
2 import { useEffect, useState } from 'react';
3 import { useNavigate, Link } from 'react-router-dom';
4 import { useAuth } from '../context/AuthContext';
5 import { useAuthService } from '../services/useAuthService';
6 import './RegistroUsuario.css';
7
8 // Componente RegistroUsuario
9 // Este componente maneja el registro y el inicio de sesión de los usuarios
10 const RegistroUsuario = () => {
11   const { loginUser } = useAuthService();
12   const navigate = useNavigate();
13   const { login } = useAuth();
14
15   // Hook de autenticación para manejar el estado de autenticación del usuario
16   const [activeForm, setActiveForm] = useState('login');
17   const [loginEmail, setLoginEmail] = useState('');
18   const [loginPassword, setLoginPassword] = useState('');
19   const [errorMessage, setErrorMessage] = useState('');
20
21   // Hook de efecto para manejar el estado de autenticación del usuario
22   const toggleForms = () => setActiveForm(activeForm === 'login' ? 'register' : 'login');
23
24   const validateEmail = (email) => /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);
25   const validatePassword = (password) => password.length >= 6;
26 }
```



```
26
27 // Validación de la contraseña para asegurarse de que tenga al menos 6 caracteres
28 const handleLoginSubmit = async (e) => {
29   e.preventDefault();
30
31   if (!validateEmail(loginEmail) || !validatePassword(loginPassword)) {
32     setErrorMessage('Correo o contraseña inválidos.');
```

- pages
 - Credenciales.css
 - Credenciales.jsx
 - Gestion.css
 - Gestion.jsx
 - RecuperarContraseña.jsx
 - RegistroUsuario.css
 - RegistroUsuario.jsx M
- router
- services
- utils
 - App.css
 - App.jsx
 - index.css
 - main.jsx
- .gitignore
- eslint.config.js
- index.html
- package-lock.json
- package.json
- README.md
- vite.config.js

- pages
 - Credenciales.css
 - Credenciales.jsx
 - Gestion.css
 - Gestion.jsx
 - RecuperarContraseña.jsx
 - RegistroUsuario.css
 - RegistroUsuario.jsx M
- router
- services
- utils
 - App.css
 - App.jsx
 - index.css
 - main.jsx
- .gitignore
- eslint.config.js
- index.html
- package-lock.json
- package.json
- README.md
- vite.config.js

- pages
 - Credenciales.css
 - Credenciales.jsx
 - Gestion.css
 - Gestion.jsx
 - RecuperarContraseña.jsx
 - RegistroUsuario.css
 - RegistroUsuario.jsx M
- router
- services
- utils
 - App.css
 - App.jsx
 - index.css
 - main.jsx
- .gitignore
- eslint.config.js
- index.html

```

33   return;
34 }
35
36 // Llamada al servicio de autenticación para iniciar sesión
37 // y manejar la respuesta
38 try {
39   const response = await loginUser(loginEmail, loginPassword);
40
41   if (response.success) { // pregunta si el success es true
42     login(); // SIN parámetros
43     navigate('/gestion');
44   } else {
45     setErrorMessage('Correo o contraseña incorrectos.');
```

- pages
 - Credenciales.css
 - Credenciales.jsx
 - Gestion.css
 - Gestion.jsx
 - RecuperarContraseña.jsx
 - RegistroUsuario.css
 - RegistroUsuario.jsx M
- router
- services
- utils
 - App.css
 - App.jsx
 - index.css
 - main.jsx
- .gitignore
- eslint.config.js
- index.html
- package-lock.json
- package.json
- README.md
- vite.config.js

- pages
 - Credenciales.css
 - Credenciales.jsx
 - Gestion.css
 - Gestion.jsx
 - RecuperarContraseña.jsx
 - RegistroUsuario.css
 - RegistroUsuario.jsx M
- router
- services
- utils
 - App.css
 - App.jsx
 - index.css
 - main.jsx
- .gitignore
- eslint.config.js
- index.html

```

46   }
47 } catch (error) {
48   console.error('Error during login:', error);
49   setErrorMessage('Error en el servidor. Intenta de nuevo.');
```

- pages
 - Credenciales.css
 - Credenciales.jsx
 - Gestion.css
 - Gestion.jsx
 - RecuperarContraseña.jsx
 - RegistroUsuario.css
 - RegistroUsuario.jsx M
- router
- services
- utils
 - App.css
 - App.jsx
 - index.css
 - main.jsx
- .gitignore
- eslint.config.js
- index.html
- package-lock.json
- package.json
- README.md
- vite.config.js

- pages
 - Credenciales.css
 - Credenciales.jsx
 - Gestion.css
 - Gestion.jsx
 - RecuperarContraseña.jsx
 - RegistroUsuario.css
 - RegistroUsuario.jsx M
- router
- services
- utils
 - App.css
 - App.jsx
 - index.css
 - main.jsx
- .gitignore
- eslint.config.js
- index.html

```

50 }
51
52 // Hook de efecto para manejar el estado de autenticación del usuario
53 return (
54   <div className="form-box">
55     <h2 className="title">Bienvenido</h2>
56     <p className="subtitle">Por favor, inicia sesión o regístrate</p>
57
58     <div id="loginContainer" className={`form-container ${activeForm === 'login' ? 'active' : ''}`>
59       <form id="loginForm" className="form" onSubmit={handleLoginSubmit}>
60         <input
61           type="email"
62           id="loginEmail"
63           className="input"
64           placeholder="Email"
65           required
66           value={loginEmail}
67           onChange={e => setLoginEmail(e.target.value)}
68         />
69         <input
70           type="password"
71           id="loginPassword"
72           className="input"
73           placeholder="Contraseña"
74           required
75           value={loginPassword}
76           onChange={e => setLoginPassword(e.target.value)}
77         />
78
79         {errorMessage && (
80           <div className="error-message">
81             {errorMessage}
82           </div>
83         )}
84         <button type="submit">Iniciar Sesión</button>
85         <div className="forgot-password">
86           <Link to="/recuperar-contrasena">¿Olvidaste tu contraseña?</Link>
87         </div>
88       </form>
89       <div className="toggle-section">
90         <p>¿No tienes una cuenta? <Link to="#" onClick={toggleForms}>Regístrate</Link></p>
91       </div>
92     </div>
93   )
94 }
95
96 export default RegistroUsuario;
```

Descripción Servicio de Autenticación con Axios

Este módulo crea un servicio de autenticación que:

Gestiona login/registro mediante Axios.

Conecta con el backend <http://localhost:3000/api/auth>

Envía credenciales y devuelve respuestas

Separa claramente login de registro

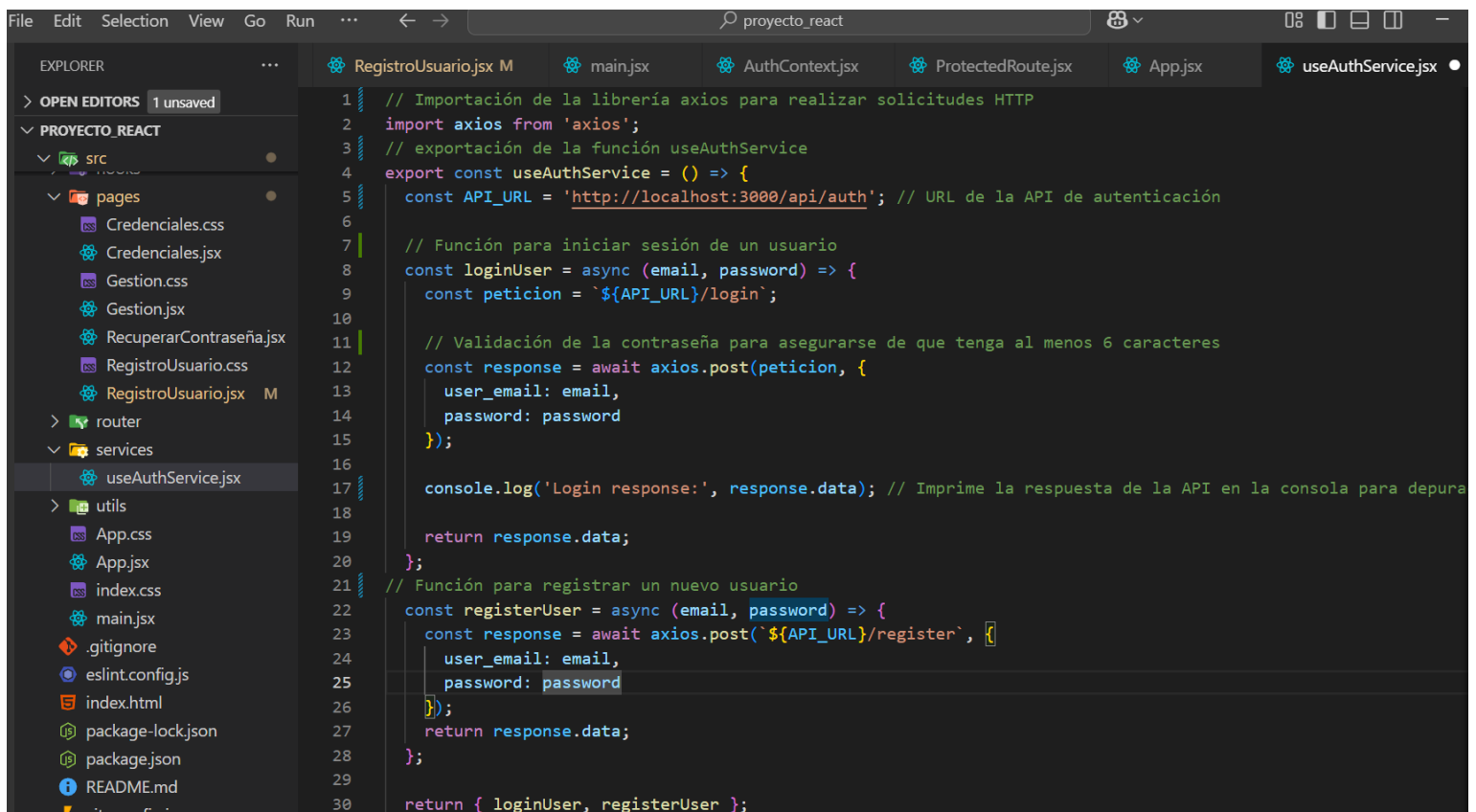
Formatea los datos para el backend

Retorna promesas con la respuesta del servidor

Realiza peticiones POST al endpoint /api/auth/login

Envía credenciales (email y contraseña) al servidor

Services/useAuthservice.jsx



```
1 // Importación de la librería axios para realizar solicitudes HTTP
2 import axios from 'axios';
3 // exportación de la función useAuthService
4 export const useAuthService = () => {
5   const API_URL = 'http://localhost:3000/api/auth'; // URL de la API de autenticación
6
7   // Función para iniciar sesión de un usuario
8   const loginUser = async (email, password) => {
9     const petición = `${API_URL}/login`;
10
11     // Validación de la contraseña para asegurarse de que tenga al menos 6 caracteres
12     const response = await axios.post(petición, {
13       user_email: email,
14       password: password
15     });
16
17     console.log('Login response:', response.data); // Imprime la respuesta de la API en la consola para depurar
18
19     return response.data;
20   };
21
22   // Función para registrar un nuevo usuario
23   const registerUser = async (email, password) => {
24     const response = await axios.post(`${API_URL}/register`, {
25       user_email: email,
26       password: password
27     });
28
29     return response.data;
30   };
31
32   return { loginUser, registerUser };
33 }
```

Comando de ejecución: npm run dev

HERRAMIENTA POSTMAN

The screenshot shows the Postman desktop application. The top bar includes navigation icons, a search bar, and an 'Upgrade' button. The left sidebar displays the 'julián's Workspace' with a tree view containing 'proyecto rock', 'rules', 'auth', 'POST login', 'usuarops', and 'reports'. The main panel shows a 'POST' request to 'http://localhost:3000/api/auth/login'. The 'Body' tab is selected, showing a JSON payload:

```
{  "user_email": "user@gmail.com",  "password": "123"}
```

. Below the request, the 'Test Results' tab shows a '200 OK' status with a response body:

```
{  "message": "inicio de sesión exitoso"}
```

. The bottom status bar includes icons for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a help icon.

The screenshot shows the Documenter web application. The top bar includes navigation icons, a search bar, and a 'Run in Postman' button. The left sidebar displays the 'PROYECTO ROCK' environment with a tree view containing 'Introduction', 'rules', 'auth', 'usuarops', and 'reports'. The main panel shows a 'POST' request to 'http://localhost:3000/api/users'. The 'Body' tab is selected, showing a JSON payload:

```
{  "user_email": "user@gmail.com",  "password": "123"}
```

. Below the request, the 'Test Results' tab shows a '200 OK' status with a response body:

```
{  "message": "inicio de sesión exitoso"}
```

. The bottom status bar includes icons for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a help icon.

Enlace: <http://localhost:3000/api/auth/login>

Enlace al repositorio GitHub

Enlace de la carpeta del backend: <https://github.com/julianfox2016/project-root.git>

Enlace del frontend: https://github.com/julianfox2016/proyecto_react.git