

## DOCUMENTACIÓN - BASE DE DATOS

NOMBRE: STEFANO FALVO

CÓDIGO: 6872

### ARQUITECTURA DE LA APLICACIÓN

#### 1. Arquitectura General

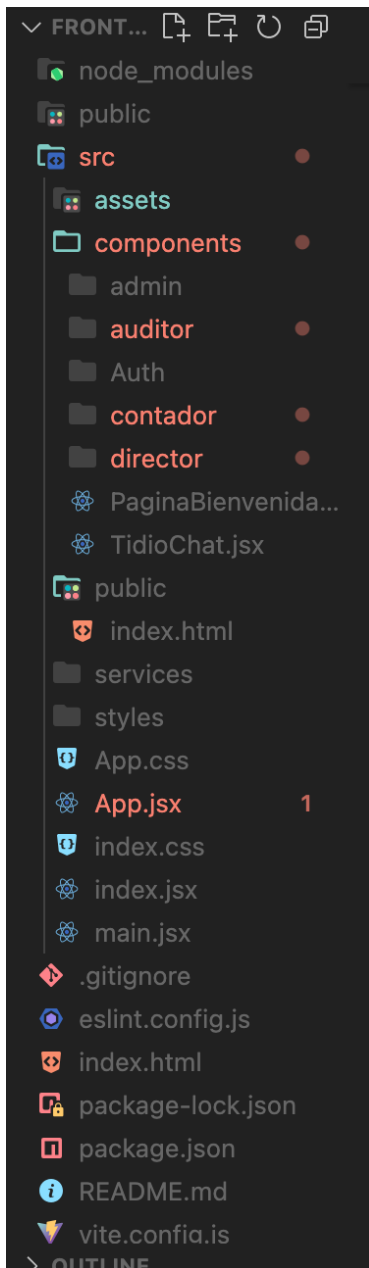
La arquitectura de la aplicación sigue un patrón de **arquitectura en capas**, donde se separan las responsabilidades para mantener un código organizado y escalable. Está dividida en tres capas principales:

- **Frontend:** Interfaz de usuario.
- **Backend:** Lógica de negocio, controladores y servicios.
- **Base de datos:** Almacenamiento persistente de datos.

#### 2. Frontend (React + Vite)

La capa de frontend se encarga de la presentación y la interacción con el usuario. Utilizas **React** para crear componentes y **Vite** como bundler para mejorar la experiencia de desarrollo.

**Estructura de carpetas:**

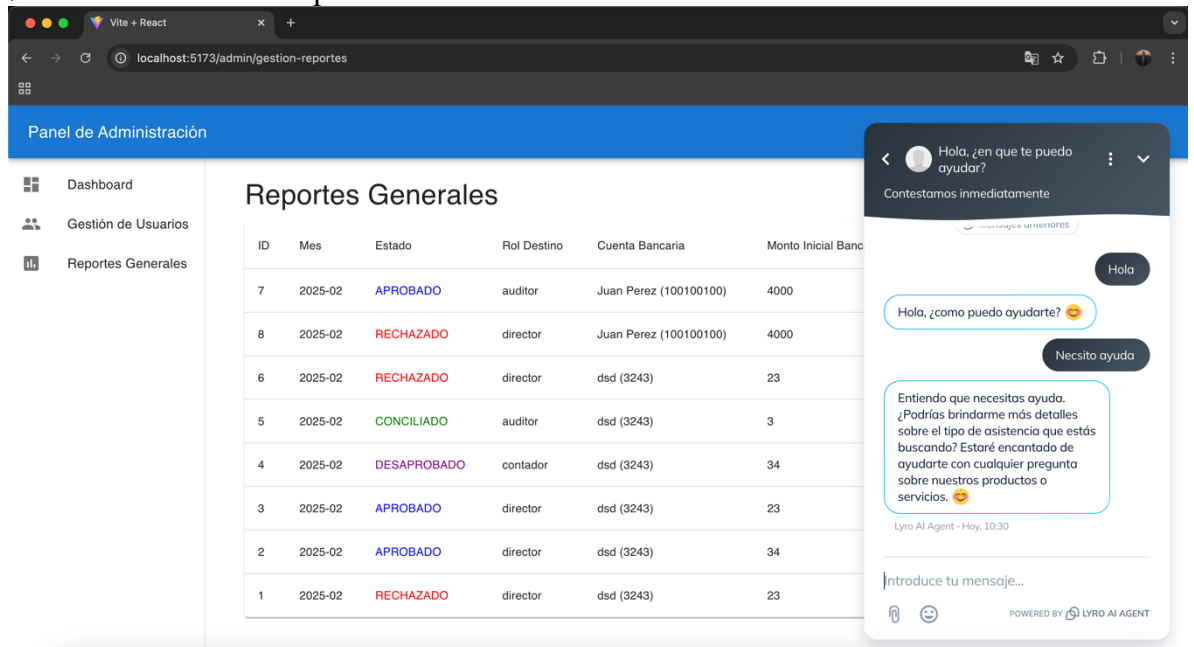


### Rutas esenciales:

- /PaginaBienvenida.jsx → Pantalla principal de la aplicación
- /components → Creación de las vistas de los diferentes roles(admin,auditor,contador,director)
- /app.jsx → Donde se declaran todas las vistas

```
src > App.jsx > ...
1 import React, { useEffect } from "react";
2 import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
3 import LoginForm from "../components/Auth/LoginForm";
4 import AdminDashboard from "../components/admin/AdminDashboard";
5 import AdminLayout from "../components/admin/AdminLayout";
6 import GestionUsuarios from "../components/admin/gestion/GestionUsuarios";
7 import ContadorDashboard from "../components/contador/ContadorDashboard";
8 import ContadorLayout from "../components/contador/ContadorLayout";
9 import CrearConciliacion from "../components/contador/CrearConciliacion";
10 import CargarCuentaBancaria from "../components/contador/gestion/CuentaBancaria";
11 import IngresoValores from "../components/contador/IngresoValores";
12 import PaginaBienvenida from "../components/PaginaBienvenida";
13 import HistorialEnvios from "../components/contador/HistorialEnvios";
14 import ReportesGenerales from "../components/admin/ReportesGenerales";
15
16 // Importamos los componentes del Director
17 import DirectorDashboard from "../components/director/DirectorDashboard";
18 import DirectorLayout from "../components/director/DirectorLayout";
19 import ConciliacionesPendientes from "../components/director/ConciliacionesPendientes";
20 import HistorialConciliaciones from "../components/director/HistorialConciliaciones";
21
22 // Importamos los componentes del Auditor
23 import AuditorDashboard from "../components/auditor/AuditorDashboard";
24 import AuditorLayout from "../components/auditor/AuditorLayout";
25 import AprobarConciliaciones from "../components/auditor/AprobarConciliaciones";
26
27 // Componente para el Chatbot de Tidio
28 const TidioChat = () => {
29   useEffect(() => {
30     const script = document.createElement("script");
31     script.src = "//code.tidio.co/z9ayukhomvojds16aaajwipqdqlrnpq.js";
32     script.async = true;
33     document.body.appendChild(script);
34   }, []);
35
36   return null; // No renderiza nada en la UI
37 }
```

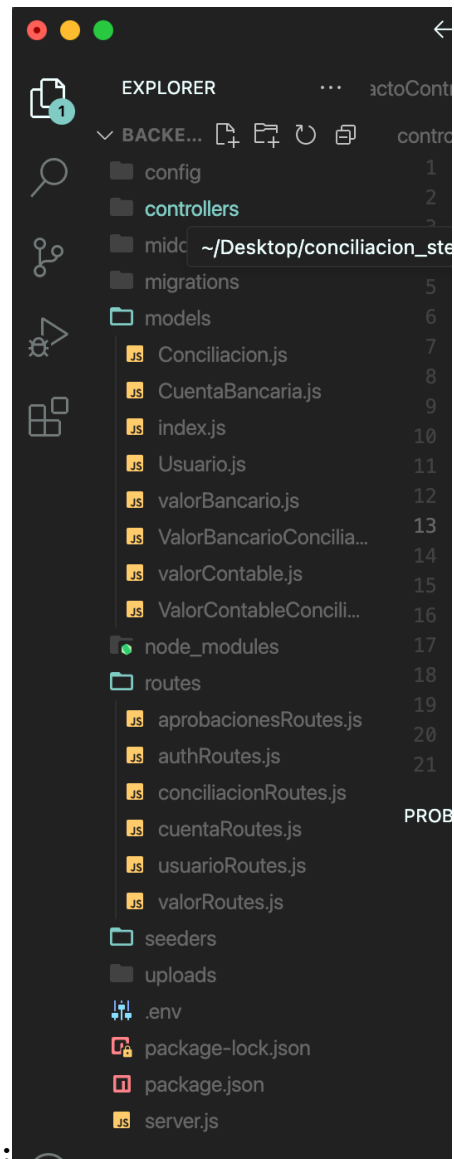
- /TidioChat→ Es la implementación del chatbot



### 3. Backend (Express + Sequelize)

La capa de backend se encarga de manejar las peticiones HTTP, procesar la lógica de negocio y hacer las interacciones con la base de datos. Usas **Express** como framework web y **Sequelize** como ORM para interactuar con la base de datos PostgreSQL.

**Estructura de carpetas:**



## Rutas:

- **Autenticación:**

- POST /login: Iniciar sesión (generar JWT).
- POST /register: Crear nuevos usuarios (solo admin).

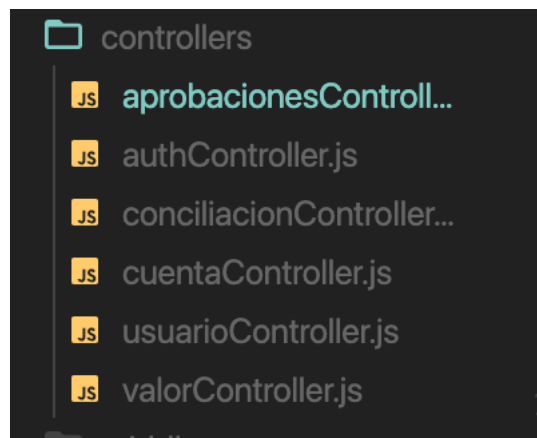
```
• const bcrypt = require("bcrypt");
• const jwt = require("jsonwebtoken");
• const {Usuario} = require("../models");
• require("dotenv").config(); // Carga las variables del archivo .env
•
• const SECRET_KEY = process.env.JWT_SECRET; // Usa la variable correcta
•
• // Función para manejar el login
• // Función para manejar el login
• const login = async (req, res) => {
•   const { email, password } = req.body;
•
•   console.log("Iniciando solicitud de login...");
•   console.log("Email:", email); // Log para ver el correo recibido
•   console.log("Contraseña:", password); // Log para ver la contraseña
recibida
•
•   try {
•     // Buscar el usuario en la base de datos
•     console.log("Buscando el usuario en la base de datos...");
•     const usuario = await Usuario.findOne({ where: { email } });
•
•     if (!usuario) {
•       console.log("Usuario no encontrado");
•       return res.status(404).json({ message: "Usuario no encontrado" });
•     }
•
•     // Comparar la contraseña con la guardada en la base de datos
•     console.log("Comparando las contraseñas...");
•     const isValidPassword = await bcrypt.compare(password,
usuario.contraseña);
•
•     if (!isValidPassword) {
•       console.log("Contraseña incorrecta");
•       return res.status(401).json({ message: "Contraseña incorrecta" });
•     }
•
•     // Generar un token JWT con una expiración de 1 hora para mejorar la
seguridad
•     console.log("Generando el token JWT...");
•     const token = jwt.sign({ id: usuario.id, rol: usuario.rol },
SECRET_KEY, { expiresIn: '1h' }); // Cambié "role" a "rol"
```

```

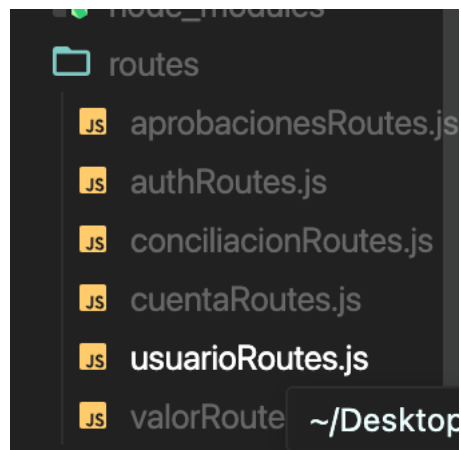
•
• // Devolver la respuesta con el rol, el id y el token
• console.log("Login exitoso. Token generado.");
• res.json({ rol: usuario.rol, id: usuario.id, token }); // Añadí el id
aquí
•
• } catch (error) {
• console.error("Error en el login:", error);
• res.status(500).json({ message: "Error en el servidor", error:
error.message });
• }
• };
•
• module.exports = { login };
•

```

- **Controladores:**



- **Rutas:**



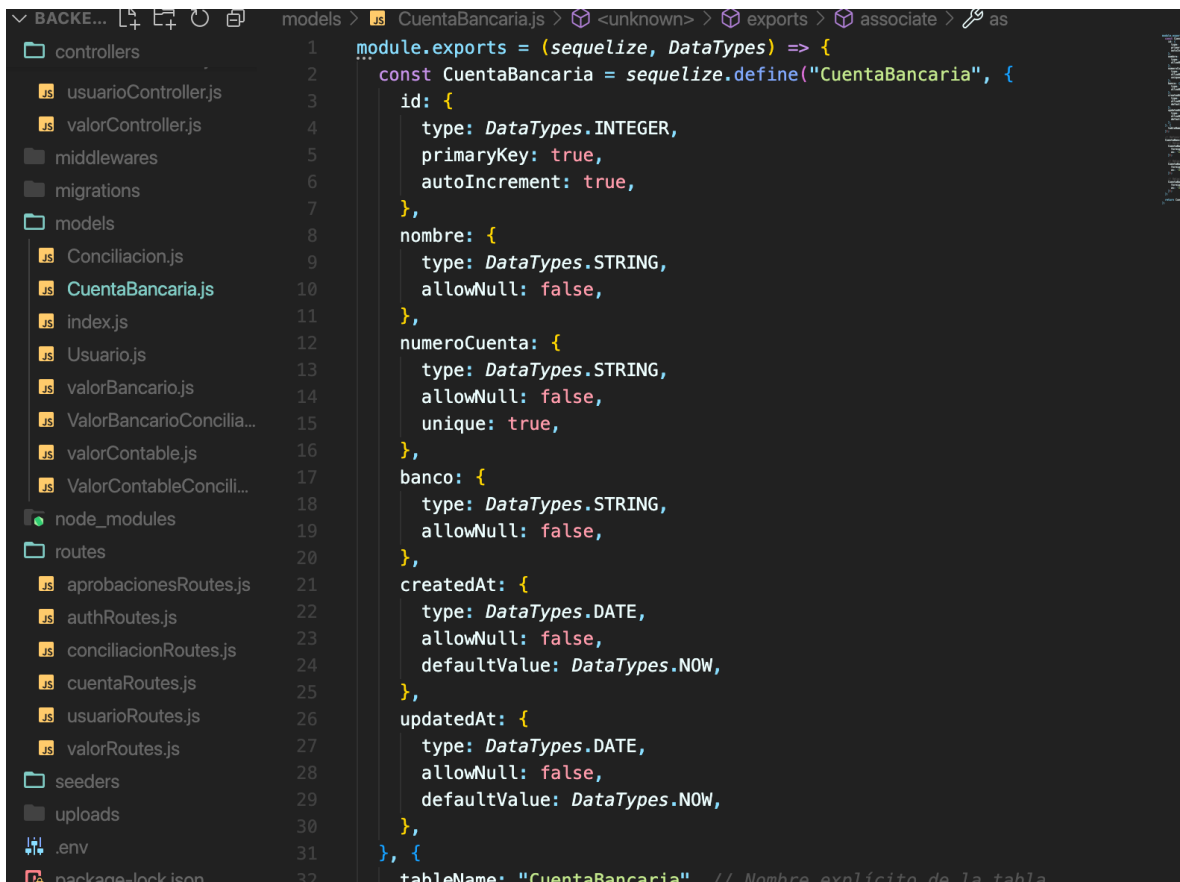
- **Server.js:** Declarar todas las rutas y configuración de la base de datos

```
server.js > ...
1 require("dotenv").config(); // Cargar variables de entorno desde .env
2 const express = require("express");
3 const cors = require("cors");
4 const { sequelize } = require("../models"); // Importar la sincronización
5 const usuarioRoutes = require("../routes/usuarioRoutes");
6 const authRoutes = require("../routes/authRoutes");
7 const cuentaBancariaRoutes = require("../routes/cuentaRoutes");
8 const valorRoutes = require("../routes/valorRoutes");
9 const conciliacionRoutes = require("../routes/conciliacionRoutes");
10 const aprobacionesRoutes = require("../routes/aprobacionesRoutes"); // Nuev.
11
12 const app = express();
13 const PORT = process.env.PORT || 5000;
14
15 app.use(cors());
16 app.use(express.json());
17
18 // Sincronización de base de datos
19 sequelize.sync({ force: false })
20   .then(() => {
21     console.log("✅ Base de datos sincronizada");
22     console.log(sequelize.models); // Verifica si todos los modelos están
23   })
24   .catch(err => console.error("❌ Error al sincronizar la DB:", err));
25
26 // Usar las rutas
27 app.use("/api/auth", authRoutes);
28 app.use("/api/usuarios", usuarioRoutes);
29 app.use("/api/cuentas", cuentaBancariaRoutes);
30 app.use("/api/valores", valorRoutes);
31 app.use("/api/conciliacion", conciliacionRoutes);
32 app.use("/api/aprobaciones", aprobacionesRoutes); // Nueva ruta para aprob.
```

## 4. Base de Datos (PostgreSQL + Sequelize)

La base de datos almacena los datos persistentes de la aplicación. Se usa **PostgreSQL** como sistema de gestión de base de datos y **Sequelize** como ORM para interactuar con la base de datos.

**Sequelize(ORM):** Se van definiendo los modelos para crear las tablas en /Models



```
1 module.exports = (sequelize, DataTypes) => {
2   const CuentaBancaria = sequelize.define("CuentaBancaria", {
3     id: {
4       type: DataTypes.INTEGER,
5       primaryKey: true,
6       autoIncrement: true,
7     },
8     nombre: {
9       type: DataTypes.STRING,
10      allowNull: false,
11    },
12    numeroCuenta: {
13      type: DataTypes.STRING,
14      allowNull: false,
15      unique: true,
16    },
17    banco: {
18      type: DataTypes.STRING,
19      allowNull: false,
20    },
21    createdAt: {
22      type: DataTypes.DATE,
23      allowNull: false,
24      defaultValue: DataTypes.NOW,
25    },
26    updatedAt: {
27      type: DataTypes.DATE,
28      allowNull: false,
29      defaultValue: DataTypes.NOW,
30    },
31  }, {
32    tableName: "CuentaBancaria", // Nombre explícito de la tabla
```

### Tablas principales:

- Tables (8)
  - > Conciliaciones
  - > CuentaBancaria
  - > SequelizeMeta
  - > Usuarios
  - > ValorBancarioConciliados
  - > ValorBancarios
  - > ValorContableConciliados
  - > ValorContables



## Diagrama:

