**PROYECTO BACKEND GESTION DE TRANSFERENCIAS**

El proyecto de backend de una Api de Gestión de las transferencias asociadas a una empres

El mismo se implementa en Node.js con Express, utilizando WebSockets y como base de datos MongoDB.

**Aclaración** : La implementación incluye los tests unitarios y el swagger

1. **MODELO**
2. **Empresa :{RazonSocial ,cuit ,fechaAdhesion }**
3. **Transferencia:** {numero ,empresa\_Id ,cuenta,importe, fechaTransferencia}
4. **EmpresaTransferencia:** {empresa\_id, cuit } ===> Entidad que relacion la empresa con la transferencia.
5. **Estructura del Proyecto**

backend/  
├── controllers/  
│ ├── empresaController.js  
│ ├── transferenciaController.js  
│ └── empresaTransferenciaController.js  
├── models/  
│ ├── empresa.js  
│ ├── transferencia.js  
│ └── empresaTransferencia.js  
├── routes/  
│ ├── empresaRoutes.js  
│ ├── transferenciaRoutes.js  
│ └── empresaTransferenciaRoutes.js  
├── tests/  
│ ├── empresa.test.js  
│ ├── transferencia.test.js  
│ └── empresaTransferencia.test.js  
├── utils/  
│ ├── database.js  
│ └── socket.js  
├── app.js  
├── package.json  
└── swagger.js

**2. Instalación de Dependencias**

Bash

npm init -y

npm install express mongoose socket.io swagger-ui-express chai mocha supertest dotenv

npm install swagger-jsdoc

**3. Archivo .env (en la raíz del proyecto)**

MONGO\_URI=mongodb://localhost:27017/trnsferencias  
PORT=3000

**4. Modelos (models/)**

* **empresa.js** JavaScript

const mongoose = require('mongoose');  
  
const empresaSchema = new mongoose.Schema({  
 razonSocial: { type: String, required: true },  
 cuit: { type: String, required: true, unique: true },  
 fechaAdhesion: { type: Date, default: Date.now },  
});  
  
module.exports = mongoose.model('Empresa', empresaSchema);

* **transferencia.js** JavaScript

const mongoose = require('mongoose');  
  
const transferenciaSchema = new mongoose.Schema({  
 numero: { type: String, required: true, unique: true },  
 empresa\_id: { type: mongoose.Schema.Types.ObjectId, ref: 'Empresa', required: true },  
 cuenta: { type: String, required: true },  
 importe: { type: Number, required: true },  
 fechaTransferencia: { type: Date, default: Date.now },  
});  
  
module.exports = mongoose.model('Transferencia', transferenciaSchema);

* **empresaTransferencia.js** JavaScript

const mongoose = require('mongoose');  
  
const empresaTransferenciaSchema = new mongoose.Schema({  
 empresa\_id: { type: mongoose.Schema.Types.ObjectId, ref: 'Empresa', required: true },  
 cuit: { type: String, required: true },  
});  
  
module.exports = mongoose.model('EmpresaTransferencia', empresaTransferenciaSchema);

**5. Conexión a MongoDB (utils/database.js)** JavaScript

const mongoose = require('mongoose');  
  
const connectDB = async () => {  
 try {  
 await mongoose.connect(process.env.MONGO\_URI, {  
 useNewUrlParser: true,  
 useUnifiedTopology: true,  
 });  
 console.log('MongoDB conectado');  
 } catch (error) {  
 console.error('Error de conexión a MongoDB:', error);  
 process.exit(1);  
 }  
};  
  
module.exports = connectDB;

**6. Controladores (controllers/)**

* **empresaController.js** JavaScript

const Empresa = require('../models/empresa');  
  
exports.crearEmpresa = async (req, res) => {  
 try {  
 const empresa = new Empresa(req.body);  
 await empresa.save();  
 res.status(201).json(empresa);  
 } catch (error) {  
 res.status(400).json({ message: error.message });  
 }  
};  
  
exports.obtenerEmpresas = async (req, res) => {  
 try {  
 const empresas = await Empresa.find();  
 res.json(empresas);  
 } catch (error) {  
 res.status(500).json({ message: error.message });  
 }  
};  
  
exports.obtenerEmpresaPorId = async (req, res) => {  
 try {  
 const empresa = await Empresa.findById(req.params.id);  
 if (!empresa) {  
 return res.status(404).json({ message: 'Empresa no encontrada' });  
 }  
 res.json(empresa);  
 } catch (error) {  
 res.status(500).json({ message: error.message });  
 }  
};  
  
exports.actualizarEmpresa = async (req, res) => {  
 try {  
 const empresa = await Empresa.findByIdAndUpdate(req.params.id, req.body, { new: true });  
 if (!empresa) {  
 return res.status(404).json({ message: 'Empresa no encontrada' });  
 }  
 res.json(empresa);  
 } catch (error) {  
 res.status(400).json({ message: error.message });  
 }  
};

exports.eliminarEmpresa = async (req, res) => {  
 try {  
 const empresa = await Empresa.findByIdAndDelete(req.params.id);  
 if (!empresa) {  
 return res.status(404).json({ message: 'Empresa no encontrada' });  
 }  
 res.json({ message: 'Empresa eliminada' });  
 } catch (error) {  
 res.status(500).json({ message: error.message });  
 }};

* **transferenciaController.js** JavaScript

const Transferencia = require('../models/transferencia');  
  
exports.crearTransferencia = async (req, res) => {  
 try {  
 const transferencia = new Transferencia(req.body);  
 await transferencia.save();  
 res.status(201).json(transferencia);  
 } catch (error) {  
 res.status(400).json({ message: error.message });  
 }  
};  
  
exports.obtenerTransferencias = async (req, res) => {  
 try {  
 const transferencias = await Transferencia.find();  
 res.json(transferencias);  
 } catch (error) {  
 res.status(500).json({ message: error.message });  
 }  
};  
  
exports.obtenerTransferenciaPorId = async (req, res) => {  
 try {  
 const transferencia = await Transferencia.findById(req.params.id);  
 if (!transferencia) {  
 return res.status(404).json({ message: 'Transferencia no encontrada' });  
 }  
 res.json(transferencia);  
 } catch (error) {  
 res.status(500).json({ message: error.message });  
 }  
};

exports.actualizarTransferencia = async (req, res) => {  
 try {  
 const transferencia = await Transferencia.findByIdAndUpdate(req.params.id, req.body, { new: true });  
 if (!transferencia) {  
 return res.status(404).json({ message: 'Transferencia no encontrada' });  
 }  
 res.json(transferencia);  
 } catch (error) {  
 res.status(400).json({ message: error.message });  
 }};  
  
exports.eliminarTransferencia = async (req, res) => {

try {  
 const transferencia = await Transferencia.findByIdAndDelete(req.params.id);  
 if (!transferencia) {  
 return res.status(404).json({ message: 'Transferencia no encontrada' });  
 }  
 res.json({ message: 'Transferencia eliminada' });  
 } catch (error) {  
 res.status(500).json({ message: error.message });  
 }};

* **empresaTransferenciaController.js** JavaScript

const EmpresaTransferencia = require('../models/empresaTransferencia');  
  
exports.crearEmpresaTransferencia = async (req, res) => {

try {  
 const empresaTransferencia = new EmpresaTransferencia(req.body);  
 await empresaTransferencia.save();  
 res.status(201).json(empresaTransferencia);  
 } catch (error) {  
 res.status(400).json({ message: error.message });  
 }};  
  
exports.obtenerEmpresasTransferencias = async (req, res) => {  
 try {  
 const empresasTransferencias = await EmpresaTransferencia.find();  
 res.json(empresasTransferencias);  
 } catch (error) {  
 res.status(500).json({ message: error.message });  
 }  
};  
  
exports.obtenerEmpresaTransferenciaPorId = async (req, res) => {  
 try {  
 const empresaTransferencia = await EmpresaTransferencia.findById(req.params.id);  
 if (!empresaTransferencia) {  
 return res.status(404).json({ message: 'EmpresaTransferencia no encontrada' });  
 }  
 res.json(empresaTransferencia);  
 } catch (error) {  
 res.status(500).json({ message: error.message });  
 }  
};  
  
exports.actualizarEmpresaTransferencia = async (req, res) => {  
 try {  
 const empresaTransferencia = await EmpresaTransferencia.findByIdAndUpdate(req.params.id, req.body, { new: true });  
 if (!empresaTransferencia) {  
 return res.status(404).json({ message: 'EmpresaTransferencia no encontrada' });  
 }  
 res.json(empresaTransferencia);  
 } catch (error) {  
 res.status(400).json({ message: error.message });  
 }  
};  
  
exports.eliminarEmpresaTransferencia = async (req, res) => {  
 try {  
 const empresaTransferencia = await EmpresaTransferencia.findByIdAndDelete(req.params.id);  
 if (!empresaTransferencia) {  
 return res.status(404).json({ message: 'EmpresaTransferencia no encontrada' });  
 }  
 res.json({ message: 'EmpresaTransferencia eliminada' });  
 } catch (error) {  
 res.status(500).json({ message: error.message });  
 }  
};

**7. Rutas (routes/)**

* **empresaRoutes.js** JavaScript

const express = require('express');  
const router = express.Router();  
const empresaController = require('../controllers/empresaController');  
/\*\*  
 \* @swagger  
 \* /empresas:  
 \* post:  
 \* summary: Crea una nueva empresa.  
 \* requestBody:  
 \* required: true  
 \* content:  
 \* application/json:  
 \* schema:  
 \* type: object  
 \* properties:  
 \* razonSocial:  
 \* type: string  
 \* cuit:  
 \* type: string  
 \* responses:  
 \* 201:  
 \* description: Empresa creada con éxito.  
 \* 400:  
 \* description: Error en la solicitud.  
 \*/  
router.post('/', empresaController.crearEmpresa);  
router.get('/', empresaController.obtenerEmpresas);  
router.get('/:id', empresaController.obtenerEmpresaPorId);  
router.put('/:id', empresaController.actualizarEmpresa);  
router.delete('/:id', empresaController.eliminarEmpresa);

module.exports = router;

* **transferenciaRoutes.js** JavaScript

const express = require('express');  
const router = express.Router();  
const transferenciaController = require('../controllers/transferenciaController');  
  
router.post('/', transferenciaController.crearTransferencia);  
router.get('/', transferenciaController.obtenerTransferencias);  
router.get('/:id', transferenciaController.obtenerTransferenciaPorId);  
router.put('/:id', transferenciaController.actualizarTransferencia);  
router.delete('/:id', transferenciaController.eliminarTransferencia);  
  
module.exports = router;

* **empresaTransferenciaRoutes.js** JavaScript

const express = require('express');  
const router = express.Router();  
const empresaTransferenciaController = require('../controllers/empresaTransferenciaController');  
  
router.post('/', empresaTransferenciaController.crearEmpresaTransferencia);  
router.get('/', empresaTransferenciaController.obtenerEmpresasTransferencias);  
router.get('/:id', empresaTransferenciaController.obtenerEmpresaTransferenciaPorId);  
router.put('/:id', empresaTransferenciaController.actualizarEmpresaTransferencia);  
router.delete('/:id', empresaTransferenciaController.eliminarEmpresaTransferencia);  
  
module.exports = router;

**8. WebSockets (utils/socket.js)** JavaScript

const socketIO = require('socket.io');  
  
const setupSocket = (server) => {  
 const io = socketIO(server, {  
 cors: {  
 origin: '\*',  
 },  
 });  
  
 io.on('connection', (socket) => {  
 console.log('Nuevo cliente conectado');  
  
 socket.on('nuevaTransferencia', (transferencia) => {  
 io.emit('transferenciaActualizada', transferencia);  
 });  
  
 socket.on('disconnect', () => {  
 console.log('Cliente desconectado');  
 });  
 });  
};  
  
module.exports = setupSocket;

**9. Tests Unitarios (tests/)**

* **empresa.test.js** JavaScript

const chai = require('chai');  
const chaiHttp = require('chai-http');  
const app = require('../app');  
const Empresa = require('../models/empresa');  
  
chai.use(chaiHttp);  
const expect = chai.expect;  
  
describe('Empresas', () => {  
 beforeEach(async () => {  
 await Empresa.deleteMany({});  
 });  
  
 if('debería crear una nueva empresa', async () => {  
 const res = await chai.request(app).post('/empresas').send({  
 razonSocial: 'Empresa de Prueba',  
 cuit: '12-34567890-1',  
 });  
 expect(res).to.have.status(201);  
 expect(res.body).to.be.an('object');  
 expect(res.body).to.have.property('razonSocial', 'Empresa de Prueba');  
 });  
  
 // Implementar otros tests para GET, PUT, DELETE  
});

* **transferencia.test.js** y **empresaTransferencia.test.js**: Similar a empresa.test.js, adaptando los modelos y rutas.

**10. Swagger (swagger.js)** JavaScript

const swaggerUi = require('swagger-ui-express');  
const swaggerJsdoc = require('swagger-jsdoc');  
  
const options = {  
 definition: {  
 openapi: '3.0.0',  
 info: {  
 title: 'API de Empresas y Transferencias',  
 version: '1.0.0',  
 description: 'Documentación de la API para gestionar empresas y transferencias',  
 },  
 },  
 apis: ['./routes/\*.js'],  
};  
const specs = swaggerJsdoc(options);  
  
module.exports = (app) => {  
 app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(specs));  
};

**11. app.js** JavaScript

const express = require('express');  
const http = require('http');  
const connectDB = require('./utils/database');  
const setupSocket = require('./utils/socket');  
const swaggerDocs = require('./swagger');  
const empresaRoutes = require('./routes/empresaRoutes');  
const transferenciaRoutes = require('./routes/transferenciaRoutes');  
const empresaTransferenciaRoutes = require('./routes/empresaTransferenciaRoutes');  
const dotenv = require('dotenv');  
  
dotenv.config();  
  
const app = express();  
const server = http.createServer(app);  
  
connectDB();  
setupSocket(server);  
swaggerDocs(app);  
  
app.use(express.json());  
app.use('/empresas', empresaRoutes);  
app.use('/transferencias', transferenciaRoutes);  
app.use('/empresasTransferencias', empresaTransferenciaRoutes);  
  
const PORT = process.env.PORT || 3000;  
server.listen(PORT, () => {  
 console.log(`Servidor corriendo en el puerto ${PORT}`);  
});  
  
module.exports = app;

**12. package.json**

JSON

{  
 "name": "backend",  
 "version": "1.0.0",  
 "description": "",  
 "main": "app.js",  
 "scripts": {  
 "test": "mocha tests/\*.test.js"  
 },  
 "dependencies": {  
 "express": "^4.17.1",  
 "mongoose": "^5.12.3",  
 "socket.io": "^4.0.0",  
 "swagger-ui-express": "^4.1.6",  
 "swagger-jsdoc": "^6.0.8",  
 "dotenv": "^16.3.1"  
 },  
 "devDependencies": {  
 "chai": "^4.3.4",  
 "chai-http": "^4.3.0",  
 "mocha": "^9.0.0",  
 "supertest": "^6.1.3"  
 }  
}

**13. Ejecución**

* Instala MongoDB y asegúrate de que esté corriendo.
* Ejecuta el servidor: node app.js
* Ejecuta los tests: npm test
* Accede a la documentación de Swagger: http://localhost:3000/api-docs

DIGRAMAS DE ARQUITECTURA

Diagrama

El contenido generado por IA puede ser incorrecto.

DIGRAMAS DE FLUJO DE DATOS

Diagrama

El contenido generado por IA puede ser incorrecto.

DIGRAMAS DE DESPLIEGUE

Diagrama

El contenido generado por IA puede ser incorrecto.