Node.js

Ch Tagore

package.json

```
{
  "name": "crm-api",
  "version": "1.0.0",
  "main": "app.js",
  "scripts": { "start": "node app.js" },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.0",
    "sequelize": "^6.31.0",
    "sqlite3": "^5.1.6"
  }
}
```

```
// app.js
const express = require('express');
const { Sequelize, DataTypes } = require('sequelize');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

const JWT_SECRET = process.env.JWT_SECRET || 'your_jwt_secret_change_this';
const PORT = process.env.PORT || 3000;

// --- Database (SQLite) ---
const sequelize = new Sequelize({
  dialect: 'sqlite',
  storage: './crm.sqlite',
  logging: false,
});

// --- Models ---
const Employee = sequelize.define('Employee', {
  name: { type: DataTypes.STRING, allowNull: true },
  email: { type: DataTypes.STRING, allowNull: false, unique: true },
  passwordHash: { type: DataTypes.STRING, allowNull: false },
});

const Enquiry = sequelize.define('Enquiry', {
  name: { type: DataTypes.STRING, allowNull: false },
```

```
  email: { type: DataTypes.STRING, allowNull: false },
  courseInterest: { type: DataTypes.STRING, allowNull: true },
  phone: { type: DataTypes.STRING, allowNull: true },
  notes: { type: DataTypes.TEXT, allowNull: true },
});

// If an enquiry is claimed, claimedBy will reference Employee.id
Enquiry.belongsTo(Employee, { as: 'claimedBy', foreignKey: 'claimedById' });
Employee.hasMany(Enquiry, { foreignKey: 'claimedById' });

// --- Express setup ---
const app = express();
app.use(express.json());

// --- Auth helpers ---
async function hashPassword(password) {
  return bcrypt.hash(password, 10);
}
async function comparePassword(password, hash) {
  return bcrypt.compare(password, hash);
}
function generateToken(employee) {
  return jwt.sign({ id: employee.id, email: employee.email }, JWT_SECRET, { expiresIn: '12h'
});
}
async function authMiddleware(req, res, next) {
  const auth = req.headers.authorization;
  if (!auth || !auth.startsWith('Bearer ')) return res.status(401).json({ message: 'Missing token'
});
  const token = auth.slice(7);
  try {
    const payload = jwt.verify(token, JWT_SECRET);
    const user = await Employee.findByPk(payload.id);
    if (!user) return res.status(401).json({ message: 'Invalid token' });
    req.user = user;
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Invalid or expired token' });
  }
}

// --- Routes ---
// 1) Register
app.post('/auth/register', async (req, res) => {
  try {
    const { name, email, password } = req.body;
    if (!email || !password) return res.status(400).json({ message: 'email & password required'
});
```

```
    const existing = await Employee.findOne({ where: { email } });
    if (existing) return res.status(409).json({ message: 'Email already registered' });
    const passwordHash = await hashPassword(password);
    const emp = await Employee.create({ name, email, passwordHash });
    return res.status(201).json({ id: emp.id, email: emp.email, name: emp.name });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: 'Server error' });
  }
});

// 2) Login
app.post('/auth/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    if (!email || !password) return res.status(400).json({ message: 'email & password required'
});
    const user = await Employee.findOne({ where: { email } });
    if (!user) return res.status(401).json({ message: 'Invalid credentials' });
    const ok = await comparePassword(password, user.passwordHash);
    if (!ok) return res.status(401).json({ message: 'Invalid credentials' });
    const token = generateToken(user);
    return res.json({ token });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: 'Server error' });
  }
});

// 3) Public form API (no auth) - create enquiry
app.post('/public/enquiries', async (req, res) => {
  try {
    const { name, email, courseInterest, phone, notes } = req.body;
    if (!name || !email) return res.status(400).json({ message: 'name & email required' });
    const enquiry = await Enquiry.create({ name, email, courseInterest, phone, notes });
    // Return created enquiry (without claimedBy)
    return res.status(201).json(enquiry);
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: 'Server error' });
  }
});

// 4) Fetch unclaimed leads (public enquiries)
app.get('/enquiries/public', authMiddleware, async (req, res) => {
  try {
    const publicEnquiries = await Enquiry.findAll({
      where: { claimedById: null },
```

```
      order: [['createdAt', 'DESC']],
    });
    return res.json(publicEnquiries);
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: 'Server error' });
  }
});

// 5) Claim a lead (assign to logged-in employee)
app.post('/enquiries/:id/claim', authMiddleware, async (req, res) => {
  try {
    const id = req.params.id;
    const enquiry = await Enquiry.findByPk(id);
    if (!enquiry) return res.status(404).json({ message: 'Enquiry not found' });
    if (enquiry.claimedById) return res.status(409).json({ message: 'Already claimed' });

    enquiry.claimedById = req.user.id;
    await enquiry.save();
    return res.json({ message: 'Claimed successfully', enquiry });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: 'Server error' });
  }
});

// 6) Fetch leads claimed by logged-in user (private enquiries)
app.get('/enquiries/mine', authMiddleware, async (req, res) => {
  try {
    const mine = await Enquiry.findAll({
      where: { claimedById: req.user.id },
      order: [['updatedAt', 'DESC']],
    });
    return res.json(mine);
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: 'Server error' });
  }
});

// --- DB sync & start ---
(async () => {
  await sequelize.sync(); // for production prefer migrations
  app.listen(PORT, () => console.log(`CRM API running on http://localhost:${PORT}`));
})();
```