ownAI Assessment API

A **Node.js API** built with **Express**, **TypeORM**, and **PostgreSQL** for user management. Supports **registration, login, JWT authentication**, and full **CRUD operations** with **role-based access control**.

Table of Contents

Project Overview

This API allows you to:

- **Register Users:** Add new users with name, email, password, phone, city, country, and role.
- **Login Users:** Authenticate and get a JWT token.
- **User Management:**
  - Admins can view, update, and delete any user.
  - Normal users can view or update their own profile.
- **Role-based Access:** Only Admins can perform sensitive actions like deleting or listing all users.

Technologies Used

- Node.js & Express.js
- TypeORM (for database ORM)
- PostgreSQL (or fallback SQLite)
- bcrypt (for password hashing)
- JWT (JSON Web Token for authentication)
- dotenv (for environment variables)

Setup Instructions

Step 1: Clone the repository

```
git clone <your-repo-url>
cd ownAI_Task
```

Step 2: Install dependencies

```
npm install
npm install -g nodemon
```

Step 3: Configure environment variables

Create a `.env` file in the project root:

```
PORT=4000

# JWT config
JWT_SECRET=your_secret
JWT_EXPIRES_IN=1d

# PostgreSQL config
PG_DB=ownAI
PG_USER=postgres      # Enter your PostgreSQL username
PG_PASS=Database      # Enter your PostgreSQL password
PG_HOST=localhost
PG_PORT=5432
```

Note: Replace `your_secret` with any secret string. Replace `PG_USER` and `PG_PASS` with your PostgreSQL credentials.

Step 4: Start the server

```
nodemon index.js
```

- The server will attempt to connect to PostgreSQL.
- If the database does not exist, it will **create automatically**.
- You should see:

```
☑ Database connected
Server running on http://localhost:4000
```

Database Structure

Users Table

Column	Type	Notes
id	uuid	Primary key
name	varchar	
email	varchar	Unique
password	varchar	Hashed
role	varchar	Default: 'Staff'
phone	varchar	Nullable
city	varchar	Nullable
country	varchar	Nullable
createdAt	timestamp	Auto-generated
updatedAt	timestamp	Auto-updated

API Endpoints

1. Register User

```
POST /api/users/register
```

Body:

```
{
  "name": "Nitesh Sharma",
  "email": "niteshkumarsharma831@gmail.com",
  "password": "123456",
  "role": "Admin",
  "phone": "9572861917",
  "city": "Gaya",
  "country": "India"
}
```

Response:

```
{
  "message": "User registered successfully",
  "user": {
    "id": "...",
    "name": "Nitesh Sharma",
    "email": "niteshkumarsharma831@gmail.com",
```

```
    "role": "Admin",
    "phone": "9572861917",
    "city": "Gaya",
    "country": "India",
    "createdAt": "...",
    "updatedAt": "..."
  }
}
```

---

2. Login

```
POST /api/users/login
```

Body:

```
{
  "email": "niteshkumarsharma831@gmail.com",
  "password": "123456"
}
```

Response:

```
{
  "message": "Login successful",
  "token": "<JWT_TOKEN>",
  "user": {
    "id": "...",
    "name": "Nitesh Sharma",
    "email": "niteshkumarsharma831@gmail.com",
    "role": "Admin"
  }
}
```

---

3. Get All Users (Admin Only)

```
GET /api/users
```

Headers:

```
Authorization: Bearer <JWT_TOKEN>
```

Response:

```
[
  {
    "id": "...",
    "name": "Nitesh Sharma",
    "email": "...",
    "role": "Admin",
    "phone": "...",
    "city": "...",
    "country": "...",
    "createdAt": "...",
    "updatedAt": "..."
  }
]
```

---

4. Get Single User

```
GET /api/users/:id
```

Headers:

```
Authorization: Bearer <JWT_TOKEN>
```

Response:

```
{
  "id": "...",
  "name": "Nitesh Sharma",
  "email": "...",
  "role": "Admin",
  "phone": "...",
  "city": "...",
  "country": "..."
}
```

---

5. Update User

```
PATCH /api/users/:id
```

Headers:

```
Authorization: Bearer <JWT_TOKEN>
```

Body (only fields to update):

```
{
  "name": "Updated Name",
  "city": "Patna",
  "password": "newpassword123"
}
```

Response:

```
{
  "message": "User updated successfully",
  "user": {
    "id": "...",
    "name": "Updated Name",
    "email": "...",
    "role": "Admin",
    "phone": "...",
    "city": "Patna",
    "country": "...",
    "createdAt": "...",
    "updatedAt": "..."
  }
}
```

---

6. Delete User

```
DELETE /api/users/:id
```

Headers:

```
Authorization: Bearer <JWT_TOKEN>
```

Response:

```
{
  "message": "User deleted successfully"
```

```
}
```

Testing Using Postman

1. Register a user with `POST /api/users/register`.
2. Login with `POST /api/users/login` to get JWT token.
3. Use token to access:
    - `GET /api/users` → list all users (Admin only)
    - `GET /api/users/:id` → view single user
    - `PATCH /api/users/:id` → update user
    - `DELETE /api/users/:id` → delete user

Notes for Beginners

- Start server using:

```
nodemon index.js
```

- PostgreSQL must be running.
- JWT tokens expire in 1 day.
- Only Admin role can list, update, or delete other users.
- Passwords are securely hashed.
- TypeORM auto-creates tables based on entity schema.
- Optional fields: `city`, `country`, `phone`.
