<https://chatgpt.com/c/68ff2b83-417c-8320-aa14-b435cf9667f1>

Perfect! Let’s build a clean, working setup for your /users JSON API using **PostgreSQL**, **TypeScript**, and **Express**. I’ll organize it step by step so it will work both locally and on Render.

**1️. Directory structure**

node\_project/

├── core/

│ ├── controllers/

│ │ └── UserController.ts

│ ├── models/

│ │ └── UserModel.ts

│ └── services/

│ └── UserService.ts

├── database.ts

├── server.ts

├── app.ts

├── package.json

├── tsconfig.json

└── .env

**2️.env (your database settings)**

PORT=3001

DB\_HOST=dpg-d3punnk9c44c73ccjbpg-a.singapore-postgres.render.com

DB\_PORT=5432

DB\_USER=mysql\_xdom\_user

DB\_PASSWORD=Ve6UBQ5psFmwRghDekXMFkILRwTXB902

DB\_NAME=mysql\_xdom

**3️⃣ database.ts**

import { Pool } from 'pg';

import dotenv from 'dotenv';

dotenv.config();

const pool = new Pool({

user: process.env.DB\_USER,

host: process.env.DB\_HOST,

database: process.env.DB\_NAME,

password: process.env.DB\_PASSWORD,

port: Number(process.env.DB\_PORT) || 5432,

});

export const dbConnect = async () => {

try {

await pool.connect();

console.log('✅ PostgreSQL connected successfully');

} catch (err) {

console.error('❌ Database connection error:', err);

}

};

export default pool;

**4️⃣ core/models/UserModel.ts**

import { QueryResult } from 'pg';

import db from '../../database';

export interface IUser {

id?: number;

name: string;

email: string;

age?: number;

class?: number;

section: string;

created\_at?: Date;

}

export class UserModel {

static async findAll(): Promise<IUser[]> {

const result: QueryResult<IUser> = await db.query(

'SELECT \* FROM users WHERE status=1 ORDER BY id DESC'

);

return result.rows;

}

static async findById(id: number): Promise<IUser | null> {

const result: QueryResult<IUser> = await db.query(

'SELECT \* FROM users WHERE id=$1 AND status=1',

[id]

);

return result.rows[0] || null;

}

static async create(user: IUser): Promise<IUser> {

const result: QueryResult<IUser> = await db.query(

`INSERT INTO users (name,email,age,class,section)

VALUES ($1,$2,$3,$4,$5) RETURNING \*`,

[user.name, user.email, user.age, user.class, user.section]

);

return result.rows[0];

}

static async update(id: number, user: IUser): Promise<boolean> {

const result: QueryResult = await db.query(

`UPDATE users

SET name=$1,email=$2,age=$3,class=$4,section=$5

WHERE id=$6 AND status=1`,

[user.name, user.email, user.age, user.class, user.section, id]

);

return result.rowCount > 0;

}

static async delete(id: number): Promise<boolean> {

const result: QueryResult = await db.query(

'UPDATE users SET status=0 WHERE id=$1',

[id]

);

return result.rowCount > 0;

}

}

**5️⃣ core/services/UserService.ts**

import { IUser, UserModel } from '../models/UserModel';

export class UserService {

static async getAllUsers(): Promise<IUser[]> {

return await UserModel.findAll();

}

static async getUserById(id: number): Promise<IUser | null> {

return await UserModel.findById(id);

}

static async createUser(userData: IUser): Promise<IUser> {

return await UserModel.create(userData);

}

static async updateUser(id: number, userData: IUser): Promise<boolean> {

return await UserModel.update(id, userData);

}

static async deleteUser(id: number): Promise<boolean> {

return await UserModel.delete(id);

}

}

**6️⃣ core/controllers/UserController.ts**

import { Request, Response } from 'express';

import { IUser } from '../models/UserModel';

import { UserService } from '../services/UserService';

export class UserController {

static async getAll(req: Request, res: Response) {

try {

const users = await UserService.getAllUsers();

res.json(users);

} catch (err) {

res.status(500).json({ error: 'Failed to fetch users' });

}

}

static async show(req: Request, res: Response) {

try {

const id = parseInt(req.params.id);

const user = await UserService.getUserById(id);

if (!user) return res.status(404).json({ error: 'User not found' });

res.json(user);

} catch (err) {

res.status(500).json({ error: 'Failed to fetch user' });

}

}

static async create(req: Request, res: Response) {

try {

const user: IUser = req.body;

const newUser = await UserService.createUser(user);

res.status(201).json(newUser);

} catch (err) {

res.status(400).json({ error: 'Failed to create user' });

}

}

static async update(req: Request, res: Response) {

try {

const id = parseInt(req.params.id);

const success = await UserService.updateUser(id, req.body);

if (!success) return res.status(404).json({ error: 'User not found' });

res.json({ message: 'User updated successfully' });

} catch (err) {

res.status(500).json({ error: 'Failed to update user' });

}

}

static async delete(req: Request, res: Response) {

try {

const id = parseInt(req.params.id);

const success = await UserService.deleteUser(id);

if (!success) return res.status(404).json({ error: 'User not found' });

res.json({ message: 'User deleted successfully' });

} catch (err) {

res.status(500).json({ error: 'Failed to delete user' });

}

}

}

**7️⃣ core/routes/user.routes.ts**

import { Router } from 'express';

import { UserController } from '../controllers/UserController';

const router = Router();

router.get('/', UserController.getAll);

router.get('/:id', UserController.show);

router.post('/', UserController.create);

router.put('/:id', UserController.update);

router.delete('/:id', UserController.delete);

export default router;

**8️⃣ app.ts**

import express from 'express';

import bodyParser from 'body-parser';

import userRoutes from './core/routes/user.routes';

import { dbConnect } from './database';

import dotenv from 'dotenv';

dotenv.config();

const app = express();

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

dbConnect();

app.use('/users', userRoutes);

app.get('/', (req, res) => res.json({ message: 'Server is running!' }));

export default app;

**9️⃣ server.ts**

import app from './app';

import dotenv from 'dotenv';

dotenv.config();

const port = Number(process.env.PORT) || 3001;

app.listen(port, () => {

console.log(`🚀 Server is running on port ${port}`);

});

**10️⃣ package.json scripts**

"scripts": {

"start": "node dist/server.js",

"dev": "nodemon --watch ./ --ext ts,json --exec ts-node server.ts",

"build": "tsc"

}

**✅ Steps to run locally:**

npm install

npm run build

npm start

Test the API:

* GET /users → list all users in JSON
* POST /users → create new user
* GET /users/:id → get single user
* PUT /users/:id → update user
* DELETE /users/:id → soft delete user

This setup **removes EJS views** completely and now everything works as a **JSON API**, which works perfectly for Render.

I can also provide a **ready-to-deploy Render version** including tsconfig.json and PostgreSQL-ready SQL script for users table if you want.

Do you want me to do that next?

Top of Form

Bottom of Form

ChatGPT