NOTES ON ADDING AN API IN TO MY TO DO LIST DATABASE CODE

**Step 1: db.js — database connection**

js

CopyEdit

import dotenv from 'dotenv';

dotenv.config();

import pg from 'pg';

const { Client } = pg;

const db = new Client({

user: process.env.DB\_USER,

host: process.env.DB\_HOST,

database: process.env.DB\_NAME,

password: process.env.DB\_PASSWORD,

port: process.env.DB\_PORT,

});

db.connect();

export default db;

**Step 2: api/items.js — API layer for DB queries**

js

CopyEdit

import db from '../db.js';

// Function to add a new to-do item

export async function addItem({ deadline, description, notes }) {

const query = `INSERT INTO items (deadline, description, notes) VALUES ($1, $2, $3)`;

await db.query(query, [deadline, description, notes]);

}

// Function to get all items

export async function getItems() {

const query = `

SELECT id, TO\_CHAR(deadline, 'DD/MM/YYYY') AS formatted\_deadline, description, notes

FROM items

ORDER BY id ASC

`;

const result = await db.query(query);

return result.rows.map(row => ({

id: row.id,

deadline: row.formatted\_deadline,

description: row.description,

notes: row.notes,

}));

}

// Function to delete an item by id

export async function deleteItem(id) {

const query = `DELETE FROM items WHERE id = $1`;

await db.query(query, [id]);

}

//NEED TO DO SOME RESEARCH INTO HOW EACH LAYER WORKS IN THIS ADDING OF AN API LAYER

**Step 3: app.js — Express app using the API**

js

CopyEdit

import express from 'express';

import bodyParser from 'body-parser';

import { addItem, getItems, deleteItem } from './api/items.js';

const app = express();

app.set('view engine', 'ejs');

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

app.use(express.static('public'));

// Render blank to-do list form

app.get('/', (req, res) => {

res.render('index.ejs', {

deadline: '',

description: '',

notes: '',

});

});

// Handle form submission and add item via API

app.post('/submit', async (req, res) => {

try {

const { deadline, description, notes } = req.body;

console.log('Received new item:', { deadline, description, notes });

await addItem({ deadline, description, notes }); // Use API function

res.redirect('/table');

} catch (err) {

console.error('Error inserting item:', err);

res.status(500).send('Server error');

}

});

// Display table with items from API

app.get('/table', async (req, res) => {

try {

const items = await getItems(); // Use API function

res.render('toDoListTable.ejs', { items });

} catch (err) {

console.error('Error fetching items:', err);

res.status(500).send('Server error');

}

});

// Render contact page

app.get('/contact', (req, res) => {

res.render('contact.ejs');

});

// Delete item via API

app.post('/delete', async (req, res) => {

try {

const id = req.body.id;

await deleteItem(id); // Use API function

res.redirect('/table');

} catch (err) {

console.error('Error deleting item:', err);

res.status(500).send('Server error');

}

});

app.listen(3000, () => {

console.log('Server running on port 3000');

});

**Explanation line-by-line**

**db.js**

* dotenv.config() — Loads env vars from .env.
* Imports pg and creates a Client using env variables.
* Connects to the database.
* Exports the db client for other modules.

**api/items.js**

* Imports the shared db client.
* addItem: Inserts a new to-do item into the database using parameterized query.
* getItems: Queries all to-do items, formats the deadline, returns clean objects.
* deleteItem: Deletes a to-do item by its ID.

This layer **encapsulates all DB logic** so your Express routes don’t need SQL directly.

**app.js**

* Sets up Express app with EJS, static files, and body-parser.
* Imports API functions.
* / route: Renders a blank form.
* /submit POST: Takes form data, calls addItem API, then redirects.
* /table GET: Calls getItems API and renders the table.
* /delete POST: Calls deleteItem API to delete an item.
* /contact: Renders a static contact page.
* Starts server on port 3000.

**Benefits of this structure:**

* **Separation of concerns:** DB access is isolated in the API module.
* **Cleaner routes:** Express routes just handle HTTP logic and call API functions.
* **Reusability:** You can use the API module in other parts (e.g., CLI scripts, future REST endpoints).
* **Easier testing:** You can test API functions independently.

If you want, I can help you generate these files or guide you step-by-step to implement!

**You said:**

what does export default db; do

**ChatGPT said:**

Great question!

**What does export default db; do?**

In ES modules (the import/export system), export default lets you export **one main value** from a file so other files can import it easily.

**In your case:**

js

CopyEdit

const db = new Client({ /\* config \*/ });

db.connect();

export default db;

* You create a db object — your connected PostgreSQL client.
* Then you **export it as the default export** from this module (file).

**How do you import it elsewhere?**

Because you used export default, other files can do:

js

CopyEdit

import db from './db.js';

And db will be the exact client object you exported.

**Why is this useful?**

* It makes sharing a single instance of db easy across your app.
* You don’t need to import with {} or worry about the variable name — the importer decides the name (db in this case).
* Keeps your database connection centralized and consistent.