ft_transformers

Day 1

Phase 1: The Setup

i install Node.js (LTS Version) and "Native" Docker Engine

i was facing a problem installing Docker Desktop on my endeavouros VM because Docker Desktop tries to create *its own* VM to run containers. since i am a*lready* inside a VM, my computer is blocking this virtualization. and one more thing is Native Docker runs directly on my Linux kernel. It is lighter, faster, and uses less RAM than Docker Desktop. Summary: on Windows or Mac, developers *need* Docker Desktop to get a Linux environment. On Linux, i am the environment. i don't need the Desktop app.

Phase 2: Create my Service

Step 1: Generate the Project

i didn't just open a blank file and start typing. i commanded a tool (the CLI) to build a complete, working skeleton for me instantly

`npm i -g @nestjs/cli`

What it is: CLI stands for Command Line Interface

a specific tool set made by the NestJS team. This tool knows exactly how to build NestJS applications. i installed it "globally" (`-g`), which means this tool is now available everywhere on my computer, not just in this folder

`nest new kanban-service`

It will ask which package manager to use. we Choose npm

NPM stands for Node Package Manager
NPM lets me download code written by other people (like NestJS, React, or Prisma) so i don't have to write everything from scratch

The CLI automatically created a standard folder structure for me. It made the `src` folder, the `test` folder (for checking errors), and configuration files. This ensures my project looks exactly like every other NestJS project in the world

i Generated "Boilerplate" Code. Boilerplate is the standard, boring code that every project needs just to start running. The CLI wrote the first few files for me automatically:

- `main.ts`: The Entry Point
- `app.module.ts`: The Root Module (the brain that organizes the code).
- `package.json`: The Manifest (the ID card that lists what our project needs).

i installed the Dependencies (bundles of code written by other people that my project needs to work). At the very end of the setup, the CLI ran a hidden command to download thousands of small code files and put them in a folder called `node_modules` it was downloading the "engine" parts so i don't have to build them myself.

Step 2: Change the Port

change the port from 3000 to 3003 inside `kanban-service/src/main.ts`

by doing this i changed the Port to give my application a unique door number (3003). This prevents a Port Conflict with other running applications, ensuring my Kanban service has its own dedicated space to listen for requests.
to run it just type `npm run start:dev` in the terminal.

Phase 3: Dockerize It (The Contract)

Step 4: Create the `Dockerfile`

```
# Start with a lightweight Node.js operating system
FROM node:20-alpine

# Install OpenSSL (The missing library)
RUN apk add --no-cache openssl

# Create a folder inside the container
```

```
WORKDIR /app

# Copy the "shopping list" (package.json) wedo this to so we
don't re-installing the libraries next time
COPY package*.json ./

# reads the package.json file and downloads all the libraries
into the node_modules folder inside the container
RUN npm install

# Copy the rest of our code
COPY . .

# Generate the Prisma Client inside the container
RUN npx prisma generate

# Build the app (Node.js cannot run TypeScript files directly
so we translate TypeScript to JavaScript inside a dist folde
r).
RUN npm run build

# Tell the container to expose port 3003
EXPOSE 3003

# The command to start the app (Run my code inside disk folde
r)
CMD ["npm", "run", "start:prod"]
```

i use my computer to *create and edit* the files because it's easier. i use the container only to *run* the files
We need to tell Docker: *"Ignore my local trash files."* This stops it from freezing at Step 5.

1. Create a new file in our root folder called `.dockerignore`
2. Paste this inside:

```
node_modules
dist
Dockerfile
.git
.env
```

Step 5: Create the `docker-compose.yml`

```
version: '3.8'

services:
  kanban-app:
    build: .
    ports:
      - "3003:3003"
    environment:
      - DATABASE_URL=postgresql://user:password@kanban_db:543
2/kanban_db?schema=public
    depends_on:
      - kanban_db

  kanban_db:
    image: postgres:15
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: kanban_db
    ports:
      - "5434:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

Phase 4: The Database

- Before Phase 4: If i started my NestJS app and created a task, it would exist in the computer's RAM (short-term memory). If i restarted the server, the task would vanish.
- After Phase 4: my application will have "Long-Term Memory." i can create a task today, turn off my computer, come back next week, and the task will still be there.

before start explaining what we are doing exactly on this phase i should explain what is exactly postgresql (or postgres)

- Technically: It is a software program that saves data into tables (rows and columns), just like Excel.
- Practically: When our users create a Task, the code sends it to Postgres to put it on a shelf. When we turn off the computer, Postgres keeps that data safe on our hard drive.

Step 6: Install Prisma

Prisma play the translator role. my code speaks TypeScript, but my database speaks SQL. Prisma translates between them so i don't have to write raw SQL queries like `SELECT * FROM ...`

- `npm install prisma --save-dev`

It downloads the Prisma CLI (Command Line Interface). i use it to create database tables and generate files. i use `--save-dev` because we use prisma cli to create database tables and generate files. we need it *while we are coding*. It tells the computer: *"Use this tool to build the project, but when we go to production (the finished app), throw it away to save space." (if we sell a chair we* do not put the electric drill,the glue, the screws,... inside the box,The customer just wants the chair, they don't need the tools used to build it*)*

- `npx prisma init`

this command automatically created two critical things in the project that didn't exist before:

1. a folder named `prisma` containing a file called `schema.prisma`.

2. a file named `.env` in our root folder.

`npx` stands for Node Package Execute. It allows us to run the Prisma tool we just installed locally without needing to install it globally on the entire computer. `npm install -g prisma` this installs it on the *entire computer*. we can run it from any folder. but `npm install prisma --save-dev` installs it ONLY inside the `node_modules` folder of the current project. If we open a new terminal in a different folder, the command `prisma` will not work. without `npx` we should run this command `./node_modules/.bin/prisma init` to tell the terminal were is prisma. With `npx`: we type `npx prisma init` . npx automatically looks inside `node_modules`, finds the hidden `prisma` tool, and runs it. It is a "finder" helper.

A. `prisma/schema.prisma` (The Blueprint)

This is the most important file in the entire database phase.

- What it is: The Declarative Schema.
- Function: It is where we define our "Models" (e.g., "I want a User", "I want a Task").
- Why we need it: This is the Source of Truth. Prisma reads this file to know how to build our SQL tables automatically. Without it, Prisma is blind.

B. `.env` (The Secrets)

- What it is: Environment Configuration.
- Function: It holds sensitive variables, specifically the `DATABASE_URL`.
- Why we need it: Hard coding passwords inside the code is a security risk. the `.env` file keeps the password safe on the computer. Prisma looks here to find the address of the database (the port `5434` we set up earlier).

Step 7: Define the "Task" Model

in this step we setup prisma. it take `schema.prisma` and translate it into SQL code so postgreSQL understand it and translate it into typescript so my code understand it. `schema.prisma` is the Universal Translator. we are writing in PSL (Prisma Schema Language) inside it. it is not a programming language it is a DSL (Domain Specific Language) It was invented by the Prisma team specifically to be Human Readable.

How does Prisma "read" this file ?

1. Prisma reads my `model Task` and converts it into SQL commands (`CREATE TABLE...`) to send to the PostgreSQL container. This physically builds the table in the database.
2. our code runs in TypeScript. It has no idea that a database exists. It doesn't know that a table named "Task" exists. It doesn't know that a "Task" has a "title”. Prisma reads `model Task` and writes TypeScript code into the `node_modules` folder.
    - It creates a TypeScript type called `Task`.
    - It creates functions like `createTask`, `findTask`, `deleteTask`.

    So, this one file is used to **build our database** AND **write my code** automatically (no SQL needed inside TypeScript code)

this is our data structure inside prisma/schema.prisma:

```
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

model Task {
  id          Int      @id @default(autoincrement()) // "id
(PK)"
  project_id  Int      // "project_id" (No relation, just a n
umber)
  title       String   // "title"
  priority    String   @default("Medium")
  description String?  // "description" (The ? means it can b
e empty)
```

```
  status      String   // "status" (TODO, DONE)
  position    Int      // "position"
  assigned_to Int?     // "assigned_to" (User ID)
  created_by  Int      // "created_by" (User ID)
  due_date    DateTime? // "due_date"
  created_at  DateTime @default(now())
  updated_at  DateTime @updatedAt
}
```

▼ Part 1: The Setup (Connecting the Lines)

- `datasource db`: defining a **Data Source** and named it `db`.
- `provider = "postgresql"`: This tells Prisma which language to speak. Since we are using a Postgres database, Prisma needs to speak the "Postgres dialect" of SQL. If we were using MySQL or MongoDB, we would change this word here.
- `url = env("DATABASE_URL")`: This is the **Phone Number** of our database.
  - `env(...)`: This function tells Prisma, *"Don't look here for the password. Go look in the secure `.env` file."*
  - It grabs the URL (which contains the username, password, port `5434`, and database name) so it knows where to connect.

▼ Part 2: The Tool Maker (The Generator)

- `generator client`: telling Prisma, *"I want we to build me a tool."*
- `provider = "prisma-client-js"`: we are specifying *which* tool. "*Build me a JavaScript/TypeScript client.*"
  - This is the instruction that tells Prisma to read our model and write that helpful TypeScript code into `node_modules` folder (the "Menu" we talked about earlier).

▼ Part 3: The Blueprint (The Task Model)

- `model Task`: This defines a new **Entity**. In our code, it will be a Class/Object. In the database, it will be a Table.

A. The Primary Key

- `id          Int       @id @default(autoincrement())`
    - `id`: The name of the column.
    - `Int`: The data type (Integer/Number).
    - `@id`: (symbol starting with `@`) marks this field as the **Primary Key**. It means this is the unique ID card for every task.
    - `@default(autoincrement())`: This is an automation rule.
        - `default`: If we don't provide a value, use this rule.
        - `autoincrement()`: Start at 1, then 2, then 3... we never have to manually number our tasks.

B. The Data Fields

- `project_id Int`: Just a number. This links the task to a specific project (Project #1, Project #2).
- `title String`: Text. The name of the task.
- `description String?`:
    - `String`: Text.
    - `?` **(Question Mark)**: This means **Nullable** (Optional). A task *can* exist without a description. If i remove the `?`, the database will force me to write a description every time.
- `status String`: Text. e.g., "TODO", "IN_PROGRESS".
- `position Int`: A number used for sorting. E.g., The top task is position `0`, the next is `100`.

C. Foreign Keys

- `assigned_to Int?`: A number representing a User ID. It is **Optional (`?`)** because a task might not be assigned to anyone yet.
- `created_by Int`: A number representing a User ID. It is **Required** because *someone* must have created the task.

D. The Time Travel Fields

- `due_date DateTime?` : A calendar date and time. It is **Optional (`?`)** because not all tasks have deadlines.
- `created_at` : Records when the task was born.
  - `@default(now())` : The moment the row is created, the database looks at its own clock and stamps the time. we never touch this field.
- `updated_at` : Records when the task was last changed.
  - `@updatedAt` : This is a special Prisma trigger. If we change the `title` of a task a week from now, Prisma will automatically update this field to that new time.

now we wrote the `schema.prisma` file, but we didn't tell Prisma to generate the actual TypeScript code from it. The "header file" (`@prisma/client`) doesn't exist yet.

1. Install the runtime library (The Object Code):

   we installed the "Dev" tools (`prisma`), but we forgot the "Runtime" library that runs when the app is live. we install the specific version that works perfectly with standard code

   `npm install prisma@5.22.0 @prisma/client@5.22.0 --save-exact`

2. Generate the Code (The Compiler):

   This reads our `schema.prisma` file and *writes* the TypeScript files for us automatically

   `npx prisma generate`

Step 8: Connect the Wires

int this step i change the DATABASE_URL inside .env file to match the docker password. when i edited this file, i was configuring the Database Connection String. This single line of text is the "bridge" that allows my NestJS application (running on my laptop) to find and talk to the PostgreSQL database (running inside the Docker container).

`DATABASE_URL="postgresql://user:password@localhost:5434/kanban_db?schema=public”`

1. `postgresql://` (The Protocol)

   - **Technical Term:** Protocol Scheme.
   - **Explanation:** This tells my application *how* to talk. It is like choosing a language. It says: "We are not talking to a website (`http://`), we are talking to a Postgres database."
2. `user:password` (The Credentials)
   - **Technical Term:** Authentication Token.
   - **Explanation:** When the app knocks on the database connection, the database checks these words. If they don't match what is in the `docker-compose.yml`, the database will reject the connection with an "Authentication Failed" error.
3. `@localhost` (The Host)
   - **Technical Term:** Hostname.
   - **Explanation:** This tells the app *where* the database lives.
     - Since i are running the app on my own computer (EndeavourOS), `localhost` means "this computer."
4. `:5434` (The Port)
   - **Technical Term:** Port Forwarding Target.
   - **Explanation:**
     - my app lives on my laptop, so it must call the "laptop number" (**5434**) to reach the container
5. `/kanban_db` (The Database Name)
   - **Technical Term:** Database Instance.
   - **Explanation:** A single Postgres server can hold many different projects (Kanban, User, Chat). This specifies exactly *which* project folder to open.

Day 2

In the early days, different languages (like a C++ server and a Java client) couldn't talk to each other easily because they stored data in memory differently (e.g., how many bytes is an `int`?). JSON was invented as a universal language for data. It is just a text string that looks like this:

```
{
  "title": "Fix the server",
  "status": "PENDING",
  "priority": 1
}
```

Every language (C++, Python, JavaScript) has a library to read this text and convert it into variables.

so it is purely a text format used to store and transport data. It is not a programming language. It cannot calculate `1 + 1`. It just sits there, holding information.

JavaScript was the original language built to run inside web browsers (like Chrome). JavaScript is loosely typed (we don't declare `int` or `string`), which makes it very annoying. as web apps got huge (like Facebook or Google Docs), the lack of types in JavaScript became a nightmare. Microsoft invented TypeScript to add Static Typing (like C++) to JavaScript. TypeScript is just like C++. It has loops, functions, and most importantly, Types (`string`, `number`, `boolean`). matter fact, the computer cannot run TypeScript directly. It compiles the TypeScript code into JavaScript, just like `gcc` compiles C code into assembly/machine code.

Step 1: Define the "Header File" (The DTO)

In C++, before we write a function, we define the `struct` or `class` in a `.h` file so the compiler knows what the data looks like. in NestJS, we call this a DTO (Data Transfer Object). We need to tell the code: *"Expect the user to send a Title and a Description."*

1. we create a file: `src/tasks/dto/create-task.dto.ts`.
   in this step i had one big question “If JSON is just data (text), why do we need a TypeScript file (`create-task.dto.ts`) for it?”

so i try to link it with c++ because it is my closest language. in C++, if we expect data from a user, we must define a `struct` or a `class` first so the compiler knows how much memory to allocate. the `.ts` file (DTO) is exactly this `struct`

When the user sends that JSON text to our server, our code doesn't know what it contains yet. It's just a text. We write the DTO (Data Transfer Object) in TypeScript to tell the program: "When we receive that JSON text, I expect it to look like this Class"

so in summary in `create-task.dto.ts`, we are writing TypeScript code that defines the shape we expect the JSON data to have.

2. inside `create-task.dto.ts` we have:

```
export class CreateTaskDto {
  title: string;

  description?: string;

  projectId: number;
}
```

Step 2: Write the Logic (The Service)

we gonna create the function that create a TODO task for us so when the browser send a request for making a TODO task we call this function and it speak with the DataBase using prisma and create a task data inside the database.

in C++, we would write a function like `void createTask(Task t) { database.save(t); }`. In NestJS, we put these functions inside a Service Class.

in this step, we are writing a single **Function Definition**.

Technically, we are defining a method inside a class. When this function is eventually called (at Runtime), the following chain of events happens in the CPU and Memory:

1. **Function Call:** The function `create()` starts executing.

2. **Argument Passing:** It receives the `CreateTaskDto` (the object we just defined) as an argument.
3. **Database Driver Invocation:** The function calls `this.prisma.task.create()`. This is the "driver" that knows how to talk to the database.
4. **Serialization:** The driver converts our TypeScript object (`{ title: "Hi" }`) into a database command (`INSERT INTO tasks...`).
5. **I/O Wait (Async):** The function pauses and waits for the hard drive to write the data.
    a. **Async/Await:**
        - Sending data to a database is an **I/O Operation** (Input/Output). It is slow compared to CPU processing.
        - In C++, a standard function would block (freeze) the thread until the database confirms the save.
        - In Node.js/TypeScript, we use `async/await`. We tell the CPU: *"Send this data to the DB, and while we wait for the confirmation, go handle other users' requests."*
6. **Return:** Once the database confirms the write, the function returns the newly created data (including the generated `id`) to the caller.

1. Run this command to generate the files automatically (inside the project folder)

   `nest g service tasks`

   The command `nest g service tasks` does **two** things.

    1. It creates the file `tasks.service.ts` with the boilerplate code
    2. It Registers the service in the `tasks.module.ts`

2. and we change `src/tasks/tasks.service.ts` into:

```
import { Injectable } from '@nestjs/common';
import { PrismaClient } from '@prisma/client';
import { CreateTaskDto } from './dto/create-task.dto';
```

```
const prisma = new PrismaClient();

@Injectable()
export class TasksService {
  async create(createTaskDto: CreateTaskDto) {
    return await prisma.task.create({
      data: {
        title: createTaskDto.title,
        description: createTaskDto.description,
        project_id: createTaskDto.projectId,
        status: 'TODO',
        position: 1,
        created_by: 1
      },
    });
  }
}
```

Step 3: The Entry Point (The Controller)

In a web server, a Controller is a class responsible for handling incoming HTTP requests and sending back HTTP responses. we are creating a Controller that will listen for a specific type of request.

1. **The Trigger:** A user (or browser) sends an **HTTP POST** request to the URL `http://localhost:3000/tasks`.
2. **The Match:** The Framework (NestJS) looks at all our Controllers. It sees that `TasksController` is assigned to listen to `/tasks`.
3. **The Handler:** Inside the Controller, we have a specific function decorated with `@Post()`. This function "catches" the request.
4. **The Extraction:** The Controller looks inside the request **Body** (the JSON data) and converts it into a `CreateTaskDto` object.

5. **The Delegation:** The Controller calls `this.tasksService.create(dto)`. (It passes the work to the Service we just wrote).
6. **The Response:** When the Service finishes, the Controller receives the result and sends it back to the user with a **201 Created** status code.

we gonna do this by modifying `src/tasks/tasks.controller.ts` we will tell it:

1. **Listen** for a `POST` request at `/tasks`.
2. **Expect** the data to look like `CreateTaskDto`.
3. **Call** the `tasksService.create()` function when that happens.

1. Run this command:

   `nest g controller tasks`

   The command automates two steps:

   1. It makes the `tasks.controller.ts` file.
   2. It opens `src/tasks/tasks.module.ts` and adds `TasksController` to the `controllers: [...]` list.

2. open `src/tasks/tasks.controller.ts`

```
import { Controller, Post, Body } from '@nestjs/common';
import { TasksService } from './tasks.service';
import { CreateTaskDto } from './dto/create-task.dto';

@Controller('tasks')
export class TasksController {
  constructor(private readonly tasksService: TasksService) {}

  @Post()
  create(@Body() createTaskDto: CreateTaskDto) {
    return this.tasksService.create(createTaskDto);
  }
}
```

now that we handle *future* events ("Hey, something just changed!"). we should handles the *past* ("Show me everything that happened before I logged in").

without `GET`, if we refresh the page, our board would be empty until someone creates a *new* task.
1. open `src/tasks/tasks.service.ts`

```
// ... inside TasksService class ...

  async findAll() {
    return await this.prisma.task.findMany({
      orderBy: { id: 'asc' }
    });
  }
```

2. The Endpoint (Controller)

   open `src/tasks/tasks.controller.ts` and add the `@Get` decorator.

   ```
   // ... inside TasksController class ...

     @Get()
     async findAll() {
       return await this.tasksService.findAll();
     }
   ```

3. The Final Verification

   ```
   curl http://localhost:3003/tasks
   ```

   If we see a JSON list `[...]` of tasks, we are done

Step 4: Test It

we don't need a frontend to test this. we can use `curl`

1. Make sure our app is running: `npm run start:dev`
2. Make sure our Docker database is running: `docker-compose up -d`
3. Step 3: Push the Tables (Create the Furniture)

`docker compose exec kanban-app npx prisma db push`

4. paste this command:

   `curl -X POST http://localhost:3003/tasks -H "Content-Type: application/json" -d '{"title": "I am connected!", "projectId": 1, "project_id": 1}'`

If the JSON response like this: `{"id": 1, "title": "Learn TypeScript", ...}` we just built our first REST API endpoint

Day 3

In a normal website (HTTP), the Client (the browser) asks a question, the Server (NestJS) answers, and then they stop talking.

In a WebSocket, we perform a **Handshake**. This is a special "Hello" that keeps the connection open. Instead of hanging up the phone, they stay on the line forever. This way, the Server can send a Payload (a packet of data) to the Client whenever it wants, without being asked first. This is called Real-Time Bi-directional Communication.

The goal for this Day is to transform our server from a standard "request-response" model into a "live" system, ensuring that when one user moves a task on the Kanban board, everyone else sees it happen instantly.

step 1: Install the "Walkie-Talkie" (The Libraries)

To make our server talk in real-time, we need to add two specific sets of code called Libraries. We use `npm` (Node Package Manager) to download them into our project.
use this command:

`npm install @nestjs/websockets @nestjs/platform-socket.io socket.io`

By running this command, we are adding the Dependencies to our `package.json` file. This tells the computer: *"From now on, this project is capable of keeping a line of communication open at all times."*

1. @nestjs/websockets: This is the Module. By default, NestJS doesn't know how to do WebSockets. This library gives NestJS the instructions it needs to understand how to handle persistent connections.

2. socket.io: This is the Engine. While "WebSocket" is the name of the technology, `socket.io` is the actual machine that does the heavy lifting. It manages the tiny details of how data travels back and forth between the computer and the browser.
3. @nestjs/platform-socket.io: This is the Adapter. It allows the NestJS "brain" to talk perfectly to the `socket.io` "engine." It makes sure they speak the exact same language.

Step 2: Create the Gateway

In NestJS, we don't use Controllers for WebSockets. Instead, we use a **Gateway**.

In the world of WebSockets, a Gateway is a special type of Class decorated with `@WebSocketGateway()`. While a Controller handles standard web addresses (URLs), a Gateway handles Socket Connections.

1. The Decorator (`@WebSocketGateway`): In code, a "Decorator" is like a Label. When we put this label on a file, we are telling the NestJS system: *"This specific file is not for normal web pages. it is the boss of the WebSocket connections."* It tells the server to open a specific Port to listen for people trying to connect. When the NestJS compiler sees `@WebSocketGateway()`, it automatically writes the extra low-level code needed to turn our simple class into a network server.
2. The Instance: When the server starts, it creates an Instance of this Gateway. This is a living piece of memory that stays awake as long as the server is running. It sits there and waits.
3. The Server Member: Inside this Gateway, we define a Server Variable. This is the actual Socket.io Server. It is the "Master Controller" that has a list of every single person (Client) who is currently connected to our app.
4. Events: The Gateway is designed to handle Events. An Event is a specific message with a name. For example, if a user moves a task, the browser sends an event named `taskMoved`. The Gateway is the thing that "hears" that name and decides what code to run next.

we do this by running this command:

`nest g gateway tasks`

This creates `src/tasks/tasks.gateway.ts` file. We create this file so the server has a central place to manage "Events." Without a Gateway, the server wouldn't know which messages to listen for or how to send information back to the users instantly.

Step 3: Write the Connection Code

```
import {
  WebSocketGateway,
  WebSocketServer,
  OnGatewayConnection,
  OnGatewayDisconnect
} from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';

// 1. We open the Gateway on Port 3003 (same as the app)
// 2. "cors: origin *" means "Allow anyone to connect" (Cruci
al for development!)
@WebSocketGateway({ cors: { origin: '*' } })
export class TasksGateway implements OnGatewayConnection, OnG
atewayDisconnect {

  // This gives us access to the "Big Microphone" to shout to
everyone
  @WebSocketServer()
  server: Server;

  // When a user connects (opens the website)
  handleConnection(client: Socket) {
    console.log(`Client connected: ${client.id}`);
  }

  // When a user closes the tab
  handleDisconnect(client: Socket) {
    console.log(`Client disconnected: ${client.id}`);
  }
}
```

This code is the "Receptionist." It welcomes users when they connect and prints a log message.

Step 4: Plug in the Radio

our server started the API (the Mailbox), but it completely ignored the WebSocket (the Walkie-Talkie). It doesn't even know it exists. This usually happens because when we generated the gateway, it wasn't automatically added to our Module file. We need to tell the `TasksModule`: *"Hey, we have a new Gateway. Please turn it on."*

1. open `src/app.module.ts`
2. Add the Gateway to "providers"

```
import { Module } from '@nestjs/common';
import { TasksService } from './tasks.service';
import { TasksController } from './tasks.controller';
import { TasksGateway } from './tasks.gateway'; // <--- Im
port this!

@Module({
  controllers: [TasksController],
  providers: [
    TasksService,
    TasksGateway // <--- Add this line!
  ],
})
export class TasksModule {}
```

Step 5: The Test

we can't use `curl` for WebSockets. We need a "Fake Frontend."

1. Create a file named `client.html`

2. Paste this code inside:

```html
<!DOCTYPE html>
<html>
<head>
    <title>WebSocket Test</title>
    <script src="https://cdn.socket.io/4.7.2/socket.io.min.js"></script>
</head>
<body>
    <h2>Status: <span id="status" style="color:red">Disconnected</span></h2>
    <script>
        // Connect to our NestJS server
        const socket = io('http://localhost:3003');

        socket.on('connect', () => {
            document.getElementById('status').innerText = "Connected! 🟢";
            document.getElementById('status').style.color = "green";
            console.log("I am connected with ID:", socket.id);
        });

        socket.on('disconnect', () => {
            document.getElementById('status').innerText = "Disconnected 🔴";
            document.getElementById('status').style.color = "red";
        });
    </script>
</body>
</html>
```

Step 6: The Moment of Truth

1. Make sure our local server is running.

   `npm run start:dev` (Make sure Docker is only running the DB in this case, or we'll get a port conflict. for Docker to runs only the DB: `docker compose up kanban_db -d`).

2. Double-click `client.html` to open it in Chrome/Firefox.

we should see:

- **In the Browser:** The text turns **Green**.
- **In our Terminal:** we see `Client connected: xxxxxxxx`.

Step 7: The Broadcast

Right now, the connection is open, but nobody is saying anything. we need to make it so that when we create a task via `curl` (HTTP), the server instantly tells the Browser (WebSocket) about it.

a. give our Gateway a function that sends a message to everyone. so we should add a function inside `src/tasks/tasks.gateway.ts` :

```
broadcastTaskCreated(task: any) {
    this.server.emit('task:created', task);
  }
```

b. Connect the Controller to the Gateway. now we need to tell the Controller (which handles the POST request) to use the Gateway. open `src/tasks/tasks.controller.ts`. we need to change the constructor to "Inject" the gateway, and then call it.

```
import { Controller, Post, Body } from '@nestjs/common';
import { TasksService } from './tasks.service';
import { CreateTaskDto } from './dto/create-task.dto';
import { TasksGateway } from './tasks.gateway'; // <--- 1. Import this
```

```
@Controller('tasks')
export class TasksController {
  constructor(
    private readonly tasksService: TasksService,
    private readonly tasksGateway: TasksGateway // <---
2. Inject the Gateway
  ) {}

  @Post()
  async create(@Body() createTaskDto: CreateTaskDto) {
    // 1. Save to Database (The Old Way)
    const task = await this.tasksService.create(createT
askDto);

    // 2. Shout to everyone (The New Way)
    this.tasksGateway.broadcastTaskCreated(task);

    return task;
  }
}
```

c. Update the fake frontend

```
// ... socket.on('connect') ...

        socket.on('task:created', (data) => {
            console.log("New Task Received:", data);

            const msg = document.createElement('div');
            msg.innerText = `🆕 New Task: ${data.title}
(ID: ${data.id})`;
            document.body.appendChild(msg);
        });
```

Recap for Day 1, Day 2 and Day 3

we built a real time, containerized microservice.

- **Day 1: The Setup (The Garage)**
  - [x] ~~Installed NestJS, Docker.~~
  - [x] ~~Designed the Database Schema (User, Project, Task) using Prisma.~~
- **Day 2: The Logic (The Engine)**
  - [x] ~~Created the DTOs (The Header Files) to validate input.~~
  - [x] ~~Built the Service (The Logic) to talk to the Database.~~
  - [x] ~~Built the Controller (The Receptionist) to handle HTTP requests.~~
- **Day 3: The Voice (Real-Time)**
  - [x] ~~Installed WebSockets (The Walkie-Talkie).~~
  - [x] ~~Built a Gateway to shout "Task Created!" to all connected users.~~
  - [x] ~~Connected a "Fake Frontend" (`client.html`) to prove it works.~~

another think is.right now if we upload our project to GitHub, **everyone in the world** will see our password (`password`). This is called "Hardcoding Secrets

so we gonna use `.env` file (this file stays on the computer because it is in `.dockerignore` and we gonna add it in `.gitignore`)

1. Update our .env file:

```
# Database Secrets
POSTGRES_USER=myuser
POSTGRES_PASSWORD=mypassword
POSTGRES_DB=kanban_db

# The Connection Strings
# For Local App (running on host) -> Uses port 5434
DATABASE_URL="postgresql://myuser:mypassword@localhost:5434/kanban_db?schema=public"
```

```
# For Docker App (running inside container) -> Uses port 5
432
DATABASE_URL_DOCKER="postgresql://myuser:mypassword@kanban
_db:5432/kanban_db?schema=public"
```

2. update `docker-compose.yml` to use Variables

```
version: '3.8'

services:
  kanban-app:
    build: .
    ports:
      - "3003:3003"
    environment:
      # We use the specific Docker URL variable here
      - DATABASE_URL=${DATABASE_URL_DOCKER}
    depends_on:
      - kanban_db

  kanban_db:
    image: postgres:15
    environment:
      # Read these from the .env file
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    ports:
      - "5434:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

to check if everything is working as expected run `docker-compose config` this will shows we how Docker "sees" the file

3. cleaning up

   let's wipe the slate clean and start fresh with our new secure settings.

   a. Kill the Zombie (Free Port 3003)

      `fuser -k 3003/tcp`

   b. Delete the Old Volume

      `docker-compose down -v`

   c. Start Fresh

      `docker-compose up -d`

   d. push the tables one last time

      `npx prisma db push`

Day 4

we aren't just storing data anymore. we are controlling *how* it changes.

Target 1: The Bouncer

right now, if I send a task with `"status": "TEST"`, our database accepts it. This will break our frontend later. We need to force it to only accept specific values.

1. **define the Rules**

   create a file: `src/tasks/dto/task-status.enum.ts`

   ```
   export enum TaskStatus {
     TODO = 'TODO',
     IN_PROGRESS = 'IN_PROGRESS',
     REVIEW = 'REVIEW',
     DONE = 'DONE',
   }
   ```

2. **Enforce the Rules**

first Install the Missing Tools by running this command:

`npm install class-validator class-transformer @nestjs/mapped-types`

we gonna update `src/tasks/dto/create-task.dto.ts`

```
import { IsString, IsNotEmpty, IsOptional, IsInt, IsEnum }
from 'class-validator';
import { TaskStatus } from './task-status.enum';

export class CreateTaskDto {
  @IsString()
  @IsNotEmpty()
  title: string;

  @IsString()
  @IsOptional()
  description?: string;

  @IsEnum(TaskStatus)
  status: TaskStatus;

  @IsString()
  @IsOptional()
  priority?: string;

  @IsInt()
  @IsOptional()
  projectId?: number;

  @IsInt()
  @IsOptional()
  createdBy?: number;

  @IsInt()
  @IsOptional()
  assignedTo?: number;
}
```

we installed the tools (`class-validator`) and we added the rules (`@IsEnum`), but we never turned the validation system ON. In NestJS, validation is not automatic. we have to explicitly tell the app: *"Hey, check every request against the DTO rules before letting it in.". so w*e need to enable the `ValidationPipe` globally in our `main.ts` file.

open `src/main.ts`

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { ValidationPipe } from '@nestjs/common';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(new ValidationPipe({
    whitelist: true,
    transform: true
  }));

  app.enableCors();

  await app.listen(process.env.PORT ?? 3003);
}
bootstrap();
```

Target 2: The Mover

we gonna allow users to move tasks (`TODO` → `DONE`)

1. **the Update Ticket (DTO)**

   Create a file: `src/tasks/dto/update-task.dto.ts`

```
import { PartialType } from '@nestjs/mapped-types';
import { CreateTaskDto } from './create-task.dto';

export class UpdateTaskDto extends PartialType(CreateTa
skDto) {}
```

that single line `extends PartialType(...)` saves us from rewriting all the properties and manually adding `@IsOptional()` to every single one

2. **The Logic (Service)**

   now we need to write the function inside `TasksService` that actually modifies the database.
   In raw SQL, we would write: `UPDATE task SET title = 'New' WHERE id = 1;` in Prisma (our ORM), this looks like a function call. It takes two main arguments:

   - `where`: The unique identifier (the `id`) to find the specific row.
   - `data`: The object containing the new values (our `UpdateTaskDto`).

   we gonna do by updating `src/tasks/tasks.service.ts`

```
// ... existing create() function ...

  async update(id: number, updateTaskDto: UpdateTaskDt
o) {
    return await this.prisma.task.update({
      where: { id: id },
      data: {
        title: updateTaskDto.title,
        description: updateTaskDto.description,
        status: updateTaskDto.status,
      },
    });
  }
```

3. **The Endpoint (Controller)**

We have the DTO (the ticket) and the Service (the logic). Now we need the Controller (the receptionist) to accept the request from the outside world. so we gonna handle `PATCH /tasks/:id` inside `src/tasks/tasks.controller.ts`

```
import { Controller, Get, Post, Body, Patch, Param, Del
ete, ParseIntPipe } from '@nestjs/common';
import { TasksService } from './tasks.service';
import { CreateTaskDto } from './dto/create-task.dto';
import { TasksGateway } from './tasks.gateway';
import { UpdateTaskDto } from './dto/update-task.dto';

@Controller('tasks')
export class TasksController {
  constructor(
    private readonly tasksService: TasksService,
    private readonly tasksGateway: TasksGateway
  ) {}

  @Post()
  async create(@Body() createTaskDto: CreateTaskDto) {
    const task = await this.tasksService.create(createT
askDto);
    this.tasksGateway.broadcastTaskCreated(task);
    return task;
  }

  @Patch(':id')
  async update(
    @Param('id', ParseIntPipe) id: number,
    @Body() updateTaskDto: UpdateTaskDto
  ) {
    const updatedTask = await this.tasksService.update
(id, updateTaskDto);
    this.tasksGateway.server.emit('task:updated', updat
edTask);
    return updatedTask;
  }
}
```

Target 3: The Cleaner (DELETE)

in C++, `delete` frees up memory that is no longer needed. In a database, `DELETE` removes a row permanently.

1. **Step 3.1: The Logic (Service)**

   We will add a `remove` function to our Service

   Prisma Delete: It works exactly like `update`, but it doesn't need `data`. It only needs `where`.
   `DELETE FROM task WHERE id = 5;`

   open `src/tasks/tasks.service.ts` file and add this function:

   ```
   async remove(id: number) {
       return await this.prisma.task.delete({
         where: { id: id },
       });
     }
   ```

2. **Step 3.2: The Endpoint (Controller)**

   We need to map the HTTP `DELETE` verb to our service logic by adding the `remove` function inside `src/tasks/tasks.service.ts` we gonna some more update to avoid some errors:

   - **Import** the `UpdateTaskDto`.
   - **Import** the `PrismaService` (assuming we have one, or `PrismaClient`).
   - **Add a Constructor** to inject Prisma so `this.prisma` works.
   - we made some changes in create too

   ```
   import { Injectable } from '@nestjs/common';
   import { CreateTaskDto } from './dto/create-task.dto';
   import { UpdateTaskDto } from './dto/update-task.dto';
   ```

```
import { PrismaClient } from '@prisma/client';

@Injectable()
export class TasksService {
  prisma = new PrismaClient();

    async create(createTaskDto: CreateTaskDto) {
    return await this.prisma.task.create({
      data: {
        title: createTaskDto.title,
        description: createTaskDto.description || '',
        status: createTaskDto.status,
        priority: createTaskDto.priority || 'Medium',

        project_id: createTaskDto.projectId || 1,
        created_by: createTaskDto.createdBy || 1,

        position: 0,
        assigned_to: createTaskDto.assignedTo || null,
      },
    });
  }


  async update(id: number, updateTaskDto: UpdateTaskDt
o) {
    return await this.prisma.task.update({
      where: { id: id },
      data: {
        ...updateTaskDto,
      },
    });
  }

  async remove(id: number) {
    return await this.prisma.task.delete({
      where: { id: id },
    });
  }
}
```

The Grand Test

1. Run this command to start just the database

   `docker-compose up kanban_db -d`

2. Start the server:

   `npm run start:dev` (Make sure Docker App is stopped first: `docker stop kanban-service-kanban-app-1`)

3. Test 1: Create a Task (To get an ID)

   ```
   curl -X POST http://localhost:3003/tasks \
      -H "Content-Type: application/json" \
      -d '{"title": "Test Task", "projectId": 1, "project_i
   d": 1}'curl -X POST http://localhost:3003/tasks \
      -H "Content-Type: application/json" \
      -d '{"title": "Test Task", "projectId": 1, "project_i
   d": 1}'
   ```

4. **Test 2: Try to break it (Validation):** Try to set status to "BANANA". It should fail (400 Bad Request).

   ```
   curl -X PATCH http://localhost:3003/tasks/1 \
      -H "Content-Type: application/json" \
      -d '{"status": "BANANA"}'
   ```

5. **Test 3: Move the Task (Update):** Set status to "DONE"

   for this test we gonna made some change on the `client.html` file

   ```
   <!DOCTYPE html>
   <html>
   ```

```
<head>
    <title>Kanban Real-Time Test</title>
    <script src="https://cdn.socket.io/4.7.2/socket.io.mi
n.js"></script>
    <style>
        body { font-family: monospace; padding: 20px; back
ground: #222; color: #eee; }
        .log { margin-bottom: 5px; padding: 5px; border-ra
dius: 4px; }
        .green { background: #1b5e20; } /* Created */
        .blue { background: #0d47a1; }  /* Updated */
        .red { background: #b71c1c; }   /* Deleted */
    </style>
</head>
<body>
    <h2>Status: <span id="status" style="color:gray">Conne
cting...</span></h2>
    <div id="messages"></div>

    <script>
        const socket = io('http://localhost:3003');

        // 1. Connection Status
        socket.on('connect', () => {
            document.getElementById('status').innerText =
"Connected! 🟢";
            document.getElementById('status').style.color
= "#4caf50";
            console.log("Connected to server");
        });

        socket.on('disconnect', () => {
            document.getElementById('status').innerText =
"Disconnected 🔴";
            document.getElementById('status').style.color
= "#f44336";
```

```
        });

        // 2. Listen for NEW Tasks (Green)
        socket.on('task:created', (data) => {
            console.log("Created:", data);
            addMessage(`🆕 New Task: ${data.title} (ID:
${data.id})`, 'green');
        });

        // 3. Listen for UPDATED Tasks (Blue) <--- This wa
s likely missing!
        socket.on('task:updated', (data) => {
            console.log("Updated:", data);
            addMessage(`🔄 Updated Task ${data.id}: Status
is now ${data.status}`, 'blue');
        });

        // 4. Listen for DELETED Tasks (Red)
        socket.on('task:deleted', (data) => {
            console.log("Deleted:", data);
            addMessage(`🗑️ Deleted Task ID: ${data.id}`,
'red');
        });

        function addMessage(text, className) {
            const div = document.createElement('div');
            div.className = `log ${className}`;
            div.innerText = text;
            document.getElementById('messages').appendChil
d(div);
        }
    </script>
</body>
</html>
```

Create a Task (to get a fresh ID, likely ID #4):

```
curl -X POST http://localhost:3003/tasks \
   -H "Content-Type: application/json" \
   -d '{"title": "Sleep Time", "projectId": 1, "project_i
d": 1}'
```

Update the Task (Change ID to match the one we just created, e.g., 4):

```
curl -X PATCH http://localhost:3003/tasks/4 \
   -H "Content-Type: application/json" \
   -d '{"status": "DONE"}'
```

6. The Cleaner (DELETE)

   The Delete Command Run this to delete Task #4 (the one we just marked as DONE)

```
curl -X DELETE http://localhost:3003/tasks/4
```

To prove it is really gone, try to update it again. we should get an "Internal Server Error" (because the database will say "I can't find ID 1").

```
curl -X PATCH http://localhost:3003/tasks/1 \
   -H "Content-Type: application/json" \
   -d '{"status": "DONE"}'
```

here we gonna get an error because we say "Hey Database, please update Task #4 to DONE." Database: "I looked everywhere. Task #1 does not exist. I cannot update a ghost."

why it says "500 Internal Server Error" instead of "404 Not Found". Right now, our code assumes the ID always exists. When it doesn't, Prisma panics and throws an exception (Crash), which NestJS reports as a 500 Error.

The Fix: Catch the Crash

1. Open `src/tasks/tasks.service.ts` and replace `update` and `remove` functions to this:

```
import { Injectable, NotFoundException } from '@nestjs/
common'; // <--- 1. Import NotFoundException
// ... other imports ...

// ... inside the class ...

  async update(id: number, updateTaskDto: UpdateTaskDt
o) {
    try {
      return await this.prisma.task.update({
        where: { id: id },
        data: { ...updateTaskDto },
      });
    } catch (error) {
      // P2025 is Prisma's code for "Record not found"
      if (error.code === 'P2025') {
        throw new NotFoundException(`Task with ID ${id}
not found`);
      }
      // If it's another error, re-throw it so we know
something else broke
      throw error;
    }
  }
  async remove(id: number) {
    try {
      return await this.prisma.task.delete({
        where: { id: id },
      });
    } catch (error) {
      if (error.code === 'P2025') {
        throw new NotFoundException(`Task with ID ${id}
not found`);
      }
      throw error;
```

```
        }
    }
```

2. Test It:
    a. Restart the server.
    b. Run the error command again:

    ```
    curl -X PATCH http://localhost:3003/tasks/4 \
       -H "Content-Type: application/json" \
       -d '{"status": "DONE"}'
    ```

Day 4 Recap

we have successfully transitioned from "Configuration" to "Coding".

- [x] ~~**Validation:** our blocked "BANANA" statuses.~~
- [x] ~~**Update:** we moved a task from TODO to DONE.~~
- [x] ~~**Delete:** we removed a task permanently.~~
- [x] ~~**Real-Time:** All of these actions sent WebSocket messages to the browser~~

Day 5/6

we are leaving the dark terminal behind. we are going to build a visual interface where we can actually see our tasks, move them around, and feel like a real user.

In the Backend we write code that talks to a database. In the Frontend, we write code that talks to the **DOM** (the Document Object Model), which is just a fancy word for "the things you see on the screen."

Step 1: Initialize the Frontend

we will create a new folder `kanban-web` right next to our `kanban-service`

run this command:

```
npm create vite@latest kanban-web -- --template react-ts
```

Step 2: Install Dependencies

we need a few tools:

- `axios`: to make HTTP requests (GET, POST). this is the tool that lets our website call our backend to get data. It handles the request and the response. `socket.io-client`: To talk to our WebSocket backend.
- `lucide-react`: For "Search, Bell, Board, Settings" icons

Run these commands inside the new `kanban-web` folder:

```
npm install
npm install axios socket.io-client lucide-react
```

Step 3: Install Tailwind CSS

It is the industry standard for styling right now (it stops us from writing thousands of lines of custom CSS).

1. **install it:**

   ```
   npm install -D tailwindcss@3.4 postcss autoprefixer
   npx tailwindcss init -p
   ```

   If that fails, do this (The Manual Fix):

   The command `init -p` is just a shortcut to create two text files. If the command is broken, we can just create them ourselves.

   - Create a new file in `kanban-web` folder named `tailwind.config.js`

     ```
     export default {
       content: [
         "./index.html",
         "./src/**/*.{js,ts,jsx,tsx}",
     ```

```
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

- Create a new file in `kanban-web` folder named `postcss.config.js`

```
export default {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
}
```

2. **Configure it:**

   we need to tell Tailwind *where* to look for our code so it knows which styles to generate.
   open `tailwind.config.js` and replace `content: []` with this:

```
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

3. **Activate it:**

   open `src/index.css` and delete everything. Replace it with just these 3 lines:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

These are special commands that are not real CSS. They tell the PostCSS tool to "inject" the actual Tailwind code here during the build process.

Step 4: The First "Hello World”

open `src/App.tsx` :

```
import { useEffect, useState } from 'react';
import axios from 'axios';

interface Task {
  id: number;
  title: string;
  status: string;
}

function App() {
  const [tasks, setTasks] = useState<Task[]>([]);

  useEffect(() => {
    axios.get('http://localhost:3003/tasks')
      .then(response => {
        setTasks(response.data);
      })
      .catch(error => {
        console.error("Error connecting to backend:", error);
      });
  }, []);

  return (
    <div className="p-10 bg-gray-900 min-h-screen text-whit
e">
      <h1 className="text-3xl font-bold mb-5">Kanban Setup</h
1>

      {tasks.length === 0 ? (
        <p className="text-gray-400">No tasks found (or backe
nd is down)...</p>
      ) : (
        <ul className="space-y-2">
          {tasks.map(task => (
            <li key={task.id} className="p-4 bg-gray-800 roun
ded border border-gray-700">
              {task.title} <span className="text-xs bg-blue-6
00 px-2 py-1 rounded ml-2">{task.status}</span>
            </li>
          ))}
        </ul>
      )}
    </div>
  );
}

export default App;
```

Step 5: Run it

```
npm run dev
```

Click the link it gives you (usually `http://localhost:5173`).

we should see a dark screen with "Kanban Setup" and a list of our tasks (like "Test Task").

we now have a Full Stack Application.

- **Backend:** NestJS (Port 3003)

- **Frontend:** React (Port 5173)
- **Connection:** Axios (Fetching data successfully)

Now let's turn that list into the dashboard

Step 1: the folder structure

1. go to `kanban-web/src`
2. create new folder named `components`
3. create a file named `Sidebar.tsx` inside this folder

Step 2: Build the Sidebar Component

We will use Tailwind

```
import { LayoutDashboard, Settings, MessageSquare, ListTodo,
ChevronDown } from 'lucide-react';

export default function Sidebar() {
  return (
    <div className="w-72 h-screen bg-[#12131a] text-slate-400
flex flex-col border-r border-gray-900/50">
      <div className="p-6">
        <div className="h-8 w-32 bg-gray-800/50 rounded mb-6
opacity-50"></div>
        <div className="text-xs font-bold text-slate-500 mb-3
uppercase tracking-wider">Workspace Switcher</div>
        <button className="flex items-center justify-between
w-full bg-[#1f212d] text-white p-3 rounded-xl hover:bg-[#2a2d
3d] transition-colors border border-gray-800/50">
          <span className="font-medium text-sm">Product Launc
h</span>
          <ChevronDown size={18} className="text-slate-400" /
>
```

```
        </button>
      </div>
      <nav className="flex-1 px-4 space-y-1">
        <SidebarItem icon={<LayoutDashboard size={20} />} lab
el="Board" active />
        <SidebarItem icon={<Settings size={20} />} label="Set
tings" />
        <SidebarItem icon={<MessageSquare size={20} />} label
="Chat" />
        <SidebarItem icon={<ListTodo size={20} />} label="My
Tasks" />
      </nav>
    </div>
  );
}

function SidebarItem({ icon, label, active = false }: { icon:
any, label: string, active?: boolean }) {
  return (
    <div className={`flex items-center gap-4 px-4 py-3 rounde
d-lg cursor-pointer transition-colors font-medium text-sm ${a
ctive ? 'bg-[#1f212d] text-white' : 'hover:bg-[#1f212d]/50 ho
ver:text-white'}`}>
      {icon}
      <span>{label}</span>
    </div>
  );
}
```

Step 3: Use the Sidebar in App.tsx

Now let's replace our "Hello World" screen with this layout

```
import { useEffect, useState } from 'react';
import axios from 'axios';
import Sidebar from './components/Sidebar';

interface Task {
  id: number;
  title: string;
  status: string;
}

function App() {
  const [tasks, setTasks] = useState<Task[]>([]);

  useEffect(() => {
    axios.get('http://localhost:3003/tasks').then(res => setT
asks(res.data));
  }, []);

  return (
    <div className="flex min-h-screen bg-white">
      <Sidebar />

      <main className="flex-1 p-8 bg-gray-50">
        <div className="flex justify-between items-center mb-
8">
          <h1 className="text-3xl font-bold text-gray-800">Pr
oduct Launch</h1>
          <button className="bg-blue-600 text-white px-4 py-2
rounded-lg hover:bg-blue-700 font-medium">
            + New Task
          </button>
        </div>

        <div className="bg-white p-6 rounded-xl shadow-sm bor
der border-gray-100">
          <h2 className="text-lg font-bold mb-4 text-gray-70
0">Raw Task Data:</h2>
          <div className="space-y-2">
```

```
            {tasks.map(task => (
              <div key={task.id} className="flex items-center
justify-between p-3 bg-gray-50 rounded-lg">
                <span>{task.title}</span>
                <span className="text-xs font-mono bg-gray-20
0 px-2 py-1 rounded">{task.status}</span>
              </div>
            ))}
          </div>
        </div>

      </main>
    </div>
  );
}

export default App;
```

now let's turn that raw list into a 4 column board

Step 1: Create the Task Card

create `src/components/TaskCard.tsx`

```
import { MoreHorizontal } from 'lucide-react';

interface TaskCardProps {
  title: string;
  status: string;
}

export default function TaskCard({ title }: TaskCardProps) {
  return (
    <div className="bg-white p-4 rounded-xl shadow-sm border
border-gray-100 cursor-pointer hover:shadow-md transition-all
```

```
group">

      <div className="flex justify-between items-start mb-3">
        <h3 className="text-sm font-bold text-gray-800 leadin
g-tight">
          {title}
        </h3>
        <button className="text-gray-400 opacity-0 group-hove
r:opacity-100 transition-opacity">
          <MoreHorizontal size={16} />
        </button>
      </div>

      <div className="mb-3">
        <div className="text-[10px] font-bold text-gray-400 u
ppercase tracking-wider mb-1">
          Priority
        </div>
        <span className="inline-block px-2 py-0.5 text-[10px]
font-bold uppercase tracking-wide text-white bg-red-500 round
ed">
          High
        </span>
      </div>

      <div>
        <div className="text-[10px] font-bold text-gray-400 u
ppercase tracking-wider mb-1">
          Assignee
        </div>
        <div className="flex items-center justify-between">
           <span className="text-xs font-semibold text-gray-7
00">user name</span>
           {/* Avatar */}
           <div className="w-6 h-6 rounded-full bg-gray-900 b
order border-white flex items-center justify-center text-[8p
x] text-white">
             UN
           </div>
        </div>
      </div>

    </div>
  );
}
```

Step 2: Create the Column

create `src/components/KanbanColumn.tsx`

```
import { Plus, MoreHorizontal } from 'lucide-react';
import TaskCard from './TaskCard';

interface Task {
  id: number;
  title: string;
  status: string;
}

interface KanbanColumnProps {
  title: string;
  tasks: Task[];
  count: number;
}

export default function KanbanColumn({ title, tasks, count }:
KanbanColumnProps) {
  return (
    <div className="flex-1 min-w-[280px] bg-[#F7F8FA] rounded
-xl p-4 flex flex-col h-full max-h-full">
```

```
      <div className="flex items-center justify-between mb-4 px-1 shrink-0">
        <div className="flex items-center gap-2">
          <h2 className="font-bold text-gray-700 text-sm">{title}</h2>
          <span className="text-gray-400 text-sm font-medium">({count})</span>
        </div>
        <button className="text-gray-400 hover:text-gray-600 transition-colors">
          <MoreHorizontal size={18} />
        </button>
      </div>

      <div className="flex-1 overflow-y-auto min-h-0 space-y-3 pr-1">

        {tasks.map((task) => (
          <TaskCard key={task.id} title={task.title} status={task.status} />
        ))}

        <button className="w-full py-2 border border-gray-300 rounded-lg text-gray-500 text-sm font-medium hover:bg-white hover:border-gray-400 transition-all flex items-center justify-center gap-2 bg-transparent group">
          <Plus size={16} className="text-gray-400 group-hover:text-gray-600" />
          New
        </button>

      </div>
    </div>
  );
}
```

Step 3: update App.tsx

```
import { useEffect, useState } from 'react';
import axios from 'axios';
import Sidebar from './components/Sidebar';
import KanbanColumn from './components/KanbanColumn';
import { Search, Bell, Filter, Home, User } from 'lucide-reac
t';

interface Task {
  id: number;
  title: string;
  status: string;
}

function App() {
  const [tasks, setTasks] = useState<Task[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    axios.get('http://localhost:3003/tasks')
      .then(res => {
        setTasks(res.data);
        setLoading(false);
      })
      .catch(err => console.error(err));
  }, []);

  const todoTasks = tasks.filter(t => t.status === 'TODO');
  const inProgressTasks = tasks.filter(t => t.status === 'IN_
PROGRESS');
  const reviewTasks = tasks.filter(t => t.status === 'REVIE
W');
  const doneTasks = tasks.filter(t => t.status === 'DONE');
  return (
    <div className="flex h-screen bg-white font-sans text-gra
y-900">
      <Sidebar />

      <main className="flex-1 flex flex-col overflow-hidden b
g-white">

        <header className="h-16 flex items-center justify-bet
ween px-8 bg-white border-b border-gray-100 shrink-0">
          <div className="flex items-center gap-3 text-sm tex
t-gray-400">
             <Home size={16} className="cursor-pointer hover:
text-gray-800 transition-colors" />
             <span>/</span>
             <span className="text-gray-900 font-medium">Prod
uct Launch</span>
          </div>

          <div className="flex items-center gap-6">
             <div className="relative">
                <Search className="absolute left-3 top-1/2 -t
ranslate-y-1/2 text-gray-400" size={18} />
                <input
                  type="text"
                  placeholder="Search"
                  className="pl-10 pr-4 py-2 bg-white border 
border-gray-200 rounded-lg text-sm focus:outline-none focus:r
ing-1 focus:ring-blue-500 w-64 transition-all placeholder-gra
y-400 hover:border-gray-300"
                />
             </div>

             <button className="text-gray-400 hover:text-gray
-600 transition-colors">
                <Bell size={22} />
```

```
              </button>

              <div className="w-9 h-9 rounded-full bg-gray-900
border border-gray-200 flex items-center justify-center overf
low-hidden cursor-pointer">
                 <User size={18} className="text-white" />
              </div>
           </div>
         </header>

         <div className="flex-1 p-8 overflow-hidden flex flex-
col">
           <div className="flex justify-between items-end mb-8
shrink-0">
             <h1 className="text-3xl font-bold text-gray-900">
Product Launch</h1>

             <div className="flex gap-3">
                <button className="px-5 py-2 bg-[#5B5CFF] text
-white rounded-lg text-sm font-semibold hover:bg-[#4d4eed] tr
ansition shadow-sm">
                  Add Member
               </button>
               <button className="flex items-center gap-2 px-4
py-2 bg-white border border-gray-200 rounded-lg text-sm font-
medium text-gray-600 hover:bg-gray-50 transition">
                  <Filter size={16} /> Filter by Member, Label
               </button>
             </div>
           </div>

           {loading ? (
              <div className="flex-1 flex items-center justify
-center text-gray-400">Loading board...</div>
           ) : (
             <div className="flex-1 grid grid-cols-1 md:grid-c
ols-2 lg:grid-cols-4 gap-6 overflow-hidden pb-2">
              <KanbanColumn title="To Do" count={todoTasks.le
ngth} tasks={todoTasks} />
              <KanbanColumn title="In Progress" count={inProg
ressTasks.length} tasks={inProgressTasks} />
              <KanbanColumn title="Review" count={reviewTask
s.length} tasks={reviewTasks} />
              <KanbanColumn title="Done" count={doneTasks.len
gth} tasks={doneTasks} />
            </div>
          )}
        </div>
      </main>
    </div>
  );
}

export default App;
```

Day 5/6 Recap

on this day we have officially finished the face

- [x] ~~**Project Setup:** Vite + React + TypeScript installed.~~
- [x] ~~**Styling:** Tailwind CSS configured (v3).~~
- [x] ~~**Backend Connection:** Axios fetching real data from our NestJS API.~~
- [x] ~~**Architecture:** Broken down into `Sidebar`, `KanbanColumn`, and `TaskCard` components.~~

Day 7/8/9

Right now our board is beautiful but frozen.we can't drag cards, we can't click "New," and if we update a task, we won't see it until we refresh. today we change that

Drag & Drop

Step 1: Install the Library

open `kanban-web` and run this command:

```
npm install @hello-pangea/dnd
```

Step 2: Setup the "Drag Context”

in `App.tsx` we need to wrap our entire board in a `<DragDropContext>`. This tells React: "Hey, monitor any mouse movements inside this area."

and add the `onDragEnd` function (logic for what happens when you drop a card) replace `src/App.tsx` with this code:

```
import { useEffect, useState } from 'react';
import axios from 'axios';
import Sidebar from './components/Sidebar';
import KanbanColumn from './components/KanbanColumn';
import { Search, Bell, Filter, Home, User } from 'lucide-reac
t';
import { DragDropContext, type DropResult } from '@hello-pang
ea/dnd';

interface Task {
  id: number;
  title: string;
  status: string;
}

function App() {
  const [tasks, setTasks] = useState<Task[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetchTasks();
  }, []);

  const fetchTasks = () => {
    axios.get('http://localhost:3003/tasks')
      .then(res => {
        setTasks(res.data);
        setLoading(false);
      })
      .catch(err => console.error(err));
  };

  const onDragEnd = async (result: DropResult) => {
    const { destination, source, draggableId } = result;

    if (!destination) return;

    if (
      destination.droppableId === source.droppableId &&
      destination.index === source.index
    ) {
      return;
    }

    const movedTaskId = parseInt(draggableId);
    const newStatus = destination.droppableId;

    const updatedTasks = tasks.map(task =>
      task.id === movedTaskId ? { ...task, status: newStatus
} : task
    );
    setTasks(updatedTasks);

    try {
      await axios.patch(`http://localhost:3003/tasks/${movedT
askId}`, {
        status: newStatus
      });
    } catch (error) {
      console.error("Failed to move task:", error);
      fetchTasks();
    }
  };

  const getTasksByStatus = (status: string) => tasks.filter(t
=> t.status === status);

  return (
    <div className="flex h-screen bg-white font-sans text-gra
y-900">
      <Sidebar />

      <main className="flex-1 flex flex-col overflow-hidden b
g-white">
        <header className="h-16 flex items-center justify-bet
ween px-8 bg-white border-b border-gray-100 shrink-0">
          <div className="flex items-center gap-3 text-sm tex
t-gray-400">
             <Home size={16} className="cursor-pointer hover:
text-gray-800 transition-colors" />
             <span>/</span>
             <span className="text-gray-900 font-medium">Prod
uct Launch</span>
          </div>
          <div className="flex items-center gap-6">
             <div className="relative">
                <Search className="absolute left-3 top-1/2 -t
ranslate-y-1/2 text-gray-400" size={18} />
                <input type="text" placeholder="Search" class
Name="pl-10 pr-4 py-2 bg-white border border-gray-200 rounded
-lg text-sm focus:outline-none focus:ring-1 focus:ring-blue-5
00 w-64 transition-all placeholder-gray-400 hover:border-gray
-300" />
```

```
             </div>
             <button className="text-gray-400 hover:text-gray
-600 transition-colors"><Bell size={22} /></button>
             <div className="w-9 h-9 rounded-full bg-gray-900
border border-gray-200 flex items-center justify-center overf
low-hidden cursor-pointer">
                <User size={18} className="text-white" />
             </div>
          </div>
        </header>

        <div className="flex-1 p-8 overflow-hidden flex flex-
col">
          <div className="flex justify-between items-end mb-8
shrink-0">
            <h1 className="text-3xl font-bold text-gray-900">
Product Launch</h1>
            <div className="flex gap-3">
               <button className="px-5 py-2 bg-[#5B5CFF] text
-white rounded-lg text-sm font-semibold hover:bg-[#4d4eed] tr
ansition shadow-sm">Add Member</button>
               <button className="flex items-center gap-2 px-
4 py-2 bg-white border border-gray-200 rounded-lg text-sm fon
t-medium text-gray-600 hover:bg-gray-50 transition"><Filter s
ize={16} /> Filter</button>
            </div>
          </div>

          <DragDropContext onDragEnd={onDragEnd}>
            {loading ? (
               <div className="flex-1 flex items-center justi
fy-center text-gray-400">Loading board...</div>
            ) : (
              <div className="flex-1 grid grid-cols-1 md:grid
-cols-2 lg:grid-cols-4 gap-6 overflow-hidden pb-2">
                <KanbanColumn id="TODO" title="To Do" count=
{getTasksByStatus('TODO').length} tasks={getTasksByStatus('TO
DO')} />
                <KanbanColumn id="IN_PROGRESS" title="In Prog
ress" count={getTasksByStatus('IN_PROGRESS').length} tasks={g
etTasksByStatus('IN_PROGRESS')} />
                <KanbanColumn id="REVIEW" title="Review" coun
t={getTasksByStatus('REVIEW').length} tasks={getTasksByStatus
('REVIEW')} />
                <KanbanColumn id="DONE" title="Done" count={g
etTasksByStatus('DONE').length} tasks={getTasksByStatus('DON
E')} />
              </div>
            )}
          </DragDropContext>
        </div>
      </main>
    </div>
  );
}

export default App;
```

Step 3: Make the Column "Droppable"

We need to wrap the card list area in a `<Droppable>` component. This tells the library: *"Hey, you can drop things inside this div."*

update `src/components/KanbanColumn.tsx`

```
import { Plus, MoreHorizontal } from 'lucide-react';
import TaskCard from './TaskCard';
import { Droppable } from '@hello-pangea/dnd';

interface Task {
  id: number;
  title: string;
  status: string;
}

interface KanbanColumnProps {
  id: string;
  title: string;
  tasks: Task[];
  count: number;
}

export default function KanbanColumn({ id, title, tasks, coun
t }: KanbanColumnProps) {
  return (
    <div className="flex-1 min-w-[280px] bg-[#F7F8FA] rounded
-xl p-4 flex flex-col h-full max-h-full">

      <div className="flex items-center justify-between mb-4
px-1 shrink-0">
        <div className="flex items-center gap-2">
          <h2 className="font-bold text-gray-700 text-sm">{ti
tle}</h2>
          <span className="text-gray-400 text-sm font-mediu
m">({count})</span>
        </div>
        <button className="text-gray-400 hover:text-gray-600
transition-colors">
          <MoreHorizontal size={18} />
        </button>
      </div>

      <Droppable droppableId={id}>
        {(provided) => (
          <div
            {...provided.droppableProps}
            ref={provided.innerRef}
            className="flex-1 overflow-y-auto min-h-0 space-y
-3 pr-1"
          >
            {tasks.map((task, index) => (
              <TaskCard
                key={task.id}
                id={task.id}
                index={index}
                title={task.title}
              />
            ))}

            {provided.placeholder}

            <button className="w-full py-2 border border-gray
-300 rounded-lg text-gray-500 text-sm font-medium hover:bg-wh
ite hover:border-gray-400 transition-all flex items-center ju
stify-center gap-2 bg-transparent group mt-3">
              <Plus size={16} className="text-gray-400 group-
hover:text-gray-600" />
              New
            </button>
          </div>
        )}
      </Droppable>
    </div>
  );
}
```

i know. this file look very complex. i well explain what exactly i did in this file to clarify (don’t expect me to do that in every file 😀). so first we define the column div and inside it we put two parts. the header and the tasks part. in the header we put the column title and count, and the three dots icon. in the tasks part we work with Droppable it is a prebuilt component that creates a logic zone. this component need a variable to define what exactly get changed when we drag a task. it is the droppableId variable so we assign the column id to it. and inside Droppable we should not just create a normal div because if we do create it

Droppable can't change on it. so we make a function so it can calculate what it need. this function take a variable. it is “provided” and it is a variable that the library give it to us. it contain pointers and data (it contain three datatypes `droppableProps` , `draggableProps` , `dragHandleProps` i well not explain what exactly what inside them but in Summary droppableProps: identification tags "i am the TODO column" , draggableProps: coordinates & style "move me to X: 50, Y: 120" , dragHandleProps: the ignition key. "start dragging when i click here"). and inside the function we return a div. after that we passe provided.droppableProps data inside this div (the ref tells the library where the div is in memory. the droppableProps tells the browser what this div is. if you inspect the HTML, you will see it adds invisible tags like data-rbd-droppable-id="TODO". without these tags, the library sees the div but doesn't realize it is a valid spot for the cards). and we assign a pointer to ref. this ref is a variable that expect a memory address of a html ellement (we call it DOM). we give it innerRef is a variable inside provided. it is a pointer created bu the library to this exact div that we are inside. and the we custom the css using the ClassName. and then we draw our tasks using tasks.map. this .map take two variable. task (i don't now how it know that task is an element of tasks) and an index (it is the task index inside the column). and then we add {provided.placeholder} that make a transparent task. it well help when we drop a task or drag it so the tasks size still the same and after that we add the + new button.

Step 4: Make the Card "Draggable”

Now we wrap the card in a `<Draggable>`. This gives it the "handles" so we can pick it up.

update `src/components/TaskCard.tsx`

```
import { MoreHorizontal } from 'lucide-react';
import { Draggable } from '@hello-pangea/dnd';

interface TaskCardProps {
  id: number;
  index: number;
  title: string;
}
```

```
export default function TaskCard({ id, index, title }: TaskCa
rdProps) {
  return (
    <Draggable draggableId={id.toString()} index={index}>
      {(provided) => (
        <div
          ref={provided.innerRef}
          {...provided.draggableProps}
          {...provided.dragHandleProps}
          className="bg-white p-4 rounded-xl shadow-sm border
border-gray-100 cursor-grab hover:shadow-md transition-all gr
oup"
        >
          <div className="flex justify-between items-start mb
-3">
            <h3 className="text-sm font-bold text-gray-800 le
ading-tight">
              {title}
            </h3>
            <button className="text-gray-400 opacity-0 group-
hover:opacity-100 transition-opacity">
              <MoreHorizontal size={16} />
            </button>
          </div>

          <div className="mb-3">
            <div className="text-[10px] font-bold text-gray-4
00 uppercase tracking-wider mb-1">Priority</div>
            <span className="inline-block px-2 py-0.5 text-[1
0px] font-bold uppercase tracking-wide text-white bg-red-500
rounded">High</span>
          </div>

          <div>
            <div className="text-[10px] font-bold text-gray-4
00 uppercase tracking-wider mb-1">Assignee</div>
            <div className="flex items-center justify-betwee
n">
              <span className="text-xs font-semibold text-gra
y-700">User Name</span>
              <div className="w-6 h-6 rounded-full bg-gray-90
0 border border-white flex items-center justify-center text-
[8px] text-white">SB</div>
            </div>
          </div>
        </div>
      )}
    </Draggable>
  );
}
```

Step 5: The Moment of Truth

- Make sure the Backend is running (`npm run start:dev` in `kanban-service`).
- Make sure the Frontend is running (`npm run dev` in `kanban-web`).
- Refresh the browser.
- Grab a card from "TODO" and drop it in "DONE".

if it didn't work try to modify the `src/main.tsx` . because the drag-and-drop library (`@hello-pangea/dnd`) hates react's **"Strict Mode"** (a tool used in development that runs everything twice to find bugs). When Strict Mode is on, the drag system gets confused, tries to initialize twice, and crashes the entire app.

```
import ReactDOM from 'react-dom/client'
import App from './App.tsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')!).render(
  // <React.StrictMode>  <-- DELETE THIS LINE
    <App />
  // </React.StrictMode> <-- DELETE THIS LINE
)
```

Day 10

Target 1: the "New" modal:

We will build the pop-up form to create tasks.

Step 1: Create the Modal Component

Create a new file `src/components/NewTaskModal.tsx`

```
import { useState } from 'react';
import { X, Paperclip, Bold, Italic, List, AlignLeft, Link }
from 'lucide-react';

interface NewTaskModalProps {
  isOpen: boolean;
  onClose: () => void;
  onSave: (title: string, description: string, status: strin
g, priority: string) => void;
  defaultStatus: string;
}

export default function NewTaskModal({ isOpen, onClose, onSav
e, defaultStatus }: NewTaskModalProps) {
  const [title, setTitle] = useState('');
  const [description, setDescription] = useState('');
  const [priority, setPriority] = useState('Medium');

  if (!isOpen) return null;

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    if (!title.trim()) return;
    onSave(title, description, defaultStatus, priority);
    setTitle('');
    setDescription('');
    setPriority('Medium');
    onClose();
  };

  return (
    <div className="fixed inset-0 bg-black/40 flex items-cent
er justify-center z-50 backdrop-blur-sm">
      <div className="bg-white w-[650px] rounded-xl shadow-2x
l overflow-hidden flex flex-col max-h-[90vh]">

        <div className="flex justify-between items-center p-5
border-b border-gray-100">
          <h2 className="text-lg font-bold text-gray-800">Tas
k Detail</h2>
          <button onClick={onClose} className="text-gray-400
hover:text-gray-600 transition-colors">
            <X size={20} />
          </button>
        </div>

        <form onSubmit={handleSubmit} className="p-6 flex-1 o
verflow-y-auto">
          <div className="mb-6">
            <input
              autoFocus
              type="text"
              value={title}
              onChange={(e) => setTitle(e.target.value)}
              placeholder="Task Title"
              className="w-full p-3 text-lg font-semibold tex
t-gray-800 border border-gray-300 rounded-lg focus:outline-no
ne focus:ring-2 focus:ring-blue-500 placeholder-gray-400"
```

```
             />
           </div>

           <div className="mb-6">
             <label className="block text-xs font-bold text-gr
ay-500 uppercase tracking-wider mb-2">Priority</label>
             <div className="flex gap-2">
               {['Low', 'Medium', 'High'].map((p) => (
                 <button
                   key={p}
                   type="button"
                   onClick={() => setPriority(p)}
                   className={`px-3 py-1.5 rounded text-sm fon
t-medium border transition-all ${
                     priority === p
                     ? 'bg-blue-50 border-blue-500 text-blue-7
00'
                     : 'bg-white border-gray-200 text-gray-600
hover:bg-gray-50'
                   }`}
                 >
                   {p}
                 </button>
               ))}
             </div>
           </div>

           <div className="mb-6 border border-gray-200 rounded
-lg overflow-hidden focus-within:ring-2 focus-within:ring-blu
e-500 transition-all">
              <div className="bg-white border-b border-gray-20
0 p-2 flex gap-1 text-gray-500">
                 <button type="button" className="p-1.5 hover:
bg-gray-100 rounded"><Bold size={16} /></button>
                 <button type="button" className="p-1.5 hover:
bg-gray-100 rounded"><Italic size={16} /></button>
```

```
                <button type="button" className="p-1.5 hover:
bg-gray-100 rounded"><AlignLeft size={16} /></button>
                <button type="button" className="p-1.5 hover:
bg-gray-100 rounded"><List size={16} /></button>
                <button type="button" className="p-1.5 hover:
bg-gray-100 rounded"><Link size={16} /></button>
             </div>
            <textarea
              value={description}
              onChange={(e) => setDescription(e.target.valu
e)}
              placeholder="Add more details to this task..."
              rows={6}
              className="w-full p-4 resize-none focus:outline
-none text-gray-700 text-sm leading-relaxed"
            />
          </div>

          <div className="mb-6">
             <label className="block text-xs font-bold text-g
ray-500 uppercase tracking-wider mb-2">Attachments</label>
             <div className="border border-dashed border-gray
-300 rounded-lg p-4 flex items-center justify-center gap-2 te
xt-gray-500 hover:bg-gray-50 cursor-pointer transition-color
s">
                <Paperclip size={16} />
                <span className="text-sm">Add File</span>
             </div>
          </div>
          <div className="flex justify-end gap-3 pt-2">
             <button type="button" onClick={onClose} classNam
e="px-5 py-2 border border-gray-300 rounded-lg text-gray-700
hover:bg-gray-50 font-medium transition-colors">
               Close
             </button>
             <button type="submit" className="px-6 py-2 bg-[#
5B5CFF] text-white rounded-lg font-bold hover:bg-[#4d4eed] sh
adow-md transition-all">
              Save
            </button>
          </div>
        </form>
      </div>
    </div>
  );
}
```

Step 2: Update the Column to Handle "Add" Clicks

we need to tell the `KanbanColumn` component: *"When someone clicks the '+ New' button, tell the App component to open the modal."*

update `src/components/KanbanColumn.tsx`:

```
import { Plus, MoreHorizontal } from 'lucide-react';
import TaskCard from './TaskCard';
import { Droppable } from '@hello-pangea/dnd';


interface KanbanColumnProps {
  id: string;
  title: string;
  tasks: Task[];
  count: number;
  onAdd: () => void;
}

export default function KanbanColumn({ id, title, tasks, coun
t, onAdd }: KanbanColumnProps) {
  return (
    <div className="flex-1 min-w-[280px] bg-[#F7F8FA] rounded
-xl p-4 flex flex-col h-full max-h-full">
```

```
      <div className="flex items-center justify-between mb-4
px-1 shrink-0">
        <div className="flex items-center gap-2">
          <h2 className="font-bold text-gray-700 text-sm">{ti
tle}</h2>
          <span className="text-gray-400 text-sm font-mediu
m">({count})</span>
        </div>
        <button className="text-gray-400 hover:text-gray-600
transition-colors">
          <MoreHorizontal size={18} />
        </button>
      </div>

      <Droppable droppableId={id}>
        {(provided) => (
          <div
            {...provided.droppableProps}
            ref={provided.innerRef}
            className="flex-1 overflow-y-auto min-h-0 space-y
-3 pr-1"
          >
            {tasks.map((task, index) => (
              <TaskCard key={task.id} id={task.id} index={ind
ex} title={task.title} />
            ))}
            {provided.placeholder}

            <button
              onClick={onAdd}
              className="w-full py-2 border border-gray-300 r
ounded-lg text-gray-500 text-sm font-medium hover:bg-white ho
ver:border-gray-400 transition-all flex items-center justify-
center gap-2 bg-transparent group mt-3"
            >
              <Plus size={16} className="text-gray-400 group-
hover:text-gray-600" />
              New
            </button>
          </div>
        )}
      </Droppable>
    </div>
  );
}
```

Step 3: Connect Everything in `App.tsx`

Now we update the main App to:

1. Track if the modal is open.
2. Track which column (status) we are adding to.
3. Send the `POST` request when you click "Save".

Update `src/App.tsx`:

```
import { useEffect, useState } from 'react';
import axios from 'axios';
import Sidebar from './components/Sidebar';
import KanbanColumn from './components/KanbanColumn';
import NewTaskModal from './components/NewTaskModal';
import { Search, Bell, Filter, Home, User } from 'lucide-reac
t';
import { DragDropContext, type DropResult } from '@hello-pang
ea/dnd';

interface Task {
  id: number;
  title: string;
  status: string;
  description?: string;
  priority?: string;
```

```
}

function App() {
  const [tasks, setTasks] = useState<Task[]>([]);
  const [loading, setLoading] = useState(true);

  const [isModalOpen, setIsModalOpen] = useState(false);
  const [activeStatus, setActiveStatus] = useState('TODO');

  useEffect(() => {
    fetchTasks();
  }, []);

  const fetchTasks = () => {
    axios.get('http://localhost:3003/tasks')
      .then(res => {
        setTasks(res.data);
        setLoading(false);
      })
      .catch(err => console.error(err));
  };

  const openModal = (status: string) => {
    setActiveStatus(status);
    setIsModalOpen(true);
  };

  const handleSaveTask = async (title: string, description: s
tring, status: string, priority: string) => {
    try {
      const res = await axios.post('http://localhost:3003/tas
ks', {
            title,
            status,
            description,
            priority
            });
      setTasks([...tasks, res.data]);
    } catch (error) {
      console.error("Error creating task:", error);
    }
  };

  const onDragEnd = async (result: DropResult) => {
    const { destination, source, draggableId } = result;
    if (!destination) return;
    if (destination.droppableId === source.droppableId && des
tination.index === source.index) return;

    const movedTaskId = parseInt(draggableId);
    const newStatus = destination.droppableId;

    const updatedTasks = tasks.map(task =>
      task.id === movedTaskId ? { ...task, status: newStatus
} : task
    );
    setTasks(updatedTasks);

    try {
      await axios.patch(`http://localhost:3003/tasks/${movedT
askId}`, { status: newStatus });
    } catch (error) {
      console.error("Failed to move task:", error);
      fetchTasks();
    }
  };

  const getTasksByStatus = (status: string) => tasks.filter(t
=> t.status === status);

  return (
    <div className="flex h-screen bg-white font-sans text-gra
```

```
y-900">
      <Sidebar />
      <main className="flex-1 flex flex-col overflow-hidden b
g-white">
        <header className="h-16 flex items-center justify-bet
ween px-8 bg-white border-b border-gray-100 shrink-0">
          <div className="flex items-center gap-3 text-sm tex
t-gray-400">
             <Home size={16} className="cursor-pointer hover:
text-gray-800 transition-colors" />
             <span>/</span>
             <span className="text-gray-900 font-medium">Prod
uct Launch</span>
          </div>
          <div className="flex items-center gap-6">
             <div className="relative">
                <Search className="absolute left-3 top-1/2 -t
ranslate-y-1/2 text-gray-400" size={18} />
                <input type="text" placeholder="Search" class
Name="pl-10 pr-4 py-2 bg-white border border-gray-200 rounded
-lg text-sm focus:outline-none focus:ring-1 focus:ring-blue-5
00 w-64 transition-all placeholder-gray-400 hover:border-gray
-300" />
             </div>
             <button className="text-gray-400 hover:text-gray
-600 transition-colors"><Bell size={22} /></button>
             <div className="w-9 h-9 rounded-full bg-gray-900
border border-gray-200 flex items-center justify-center overf
low-hidden cursor-pointer">
                <User size={18} className="text-white" />
             </div>
          </div>
        </header>

        <div className="flex-1 p-8 overflow-hidden flex flex-
col">
```

```
          <div className="flex justify-between items-end mb-8
shrink-0">
            <h1 className="text-3xl font-bold text-gray-900">
Product Launch</h1>
            <div className="flex gap-3">
               <button className="px-5 py-2 bg-[#5B5CFF] text
-white rounded-lg text-sm font-semibold hover:bg-[#4d4eed] tr
ansition shadow-sm">Add Member</button>
               <button className="flex items-center gap-2 px-
4 py-2 bg-white border border-gray-200 rounded-lg text-sm fon
t-medium text-gray-600 hover:bg-gray-50 transition"><Filter s
ize={16} /> Filter</button>
            </div>
          </div>

          <DragDropContext onDragEnd={onDragEnd}>
            {loading ? (
               <div className="flex-1 flex items-center justi
fy-center text-gray-400">Loading board...</div>
            ) : (
              <div className="flex-1 grid grid-cols-1 md:grid
-cols-2 lg:grid-cols-4 gap-6 overflow-hidden pb-2">
                <KanbanColumn id="TODO" title="To Do" count=
{getTasksByStatus('TODO').length} tasks={getTasksByStatus('TO
DO')} onAdd={() => openModal('TODO')} />
                <KanbanColumn id="IN_PROGRESS" title="In Prog
ress" count={getTasksByStatus('IN_PROGRESS').length} tasks={g
etTasksByStatus('IN_PROGRESS')} onAdd={() => openModal('IN_PR
OGRESS')} />
                <KanbanColumn id="REVIEW" title="Review" coun
t={getTasksByStatus('REVIEW').length} tasks={getTasksByStatus
('REVIEW')} onAdd={() => openModal('REVIEW')} />
                <KanbanColumn id="DONE" title="Done" count={g
etTasksByStatus('DONE').length} tasks={getTasksByStatus('DON
E')} onAdd={() => openModal('DONE')} />
              </div>
            )}
          </DragDropContext>
        </div>
      </main>

      <NewTaskModal
        isOpen={isModalOpen}
        onClose={() => setIsModalOpen(false)}
        onSave={handleSaveTask}
        defaultStatus={activeStatus}
      />
    </div>
  );
}

export default App;
```

our task card component is still hardcoded to always show "High" (we did this in day 5 just to make it look nice). It is ignoring the real data.

update `TaskCard.tsx`

```
import { MoreHorizontal } from 'lucide-react';
import { Draggable } from '@hello-pangea/dnd';

interface TaskCardProps {
  id: number;
  index: number;
  title: string;
  priority?: string
}

const getPriorityColor = (priority: string) => {
  switch (priority?.toLowerCase()) {
    case 'high': return 'bg-red-500';
    case 'medium': return 'bg-amber-500';
    case 'low': return 'bg-emerald-500';
    default: return 'bg-gray-400';
  }
};

export default function TaskCard({ id, index, title, priority
= 'Medium' }: TaskCardProps) {
  return (
    <Draggable draggableId={id.toString()} index={index}>
      {(provided) => (
        <div
          ref={provided.innerRef}
          {...provided.draggableProps}
          {...provided.dragHandleProps}
          className="bg-white p-4 rounded-xl shadow-sm border
border-gray-100 cursor-grab hover:shadow-md transition-all gr
oup"
        >
          <div className="flex justify-between items-start mb
-3">
            <h3 className="text-sm font-bold text-gray-800 le
ading-tight">
              {title}
            </h3>
            <button className="text-gray-400 opacity-0 group-
hover:opacity-100 transition-opacity">
              <MoreHorizontal size={16} />
            </button>
          </div>

          <div className="mb-3">
            <div className="text-[10px] font-bold text-gray-4
00 uppercase tracking-wider mb-1">Priority</div>
            <span className={`inline-block px-2 py-0.5 text-
[10px] font-bold uppercase tracking-wide text-white rounded
${getPriorityColor(priority)}`}>
```

```
              {priority}
            </span>
          </div>

          <div>
            <div className="text-[10px] font-bold text-gray-4
00 uppercase tracking-wider mb-1">Assignee</div>
            <div className="flex items-center justify-betwee
n">
              <span className="text-xs font-semibold text-gra
y-700">User Name</span>
              <div className="w-6 h-6 rounded-full bg-gray-90
0 border border-white flex items-center justify-center text-
[8px] text-white">SB</div>
            </div>
          </div>
        </div>
      )}
    </Draggable>
  );
}
```

now we need to tell the column to actually *give* the priority data to the card.

```
// ... imports

interface Task {
  id: number;
  title: string;
  status: string;
  priority?: string;
}

interface KanbanColumnProps {
  id: string;
  title: string;
  tasks: Task[];
  count: number;
  onAdd: () => void;
}

export default function KanbanColumn({ id, title, tasks, coun
t, onAdd }: KanbanColumnProps) {
  return (
    <div className="flex-1 min-w-[280px] bg-[#F7F8FA] rounded
-xl p-4 flex flex-col h-full max-h-full">
      <div className="flex items-center justify-between mb-4
px-1 shrink-0">
        <div className="flex items-center gap-2">
          <h2 className="font-bold text-gray-700 text-sm">{ti
tle}</h2>
          <span className="text-gray-400 text-sm font-mediu
m">({count})</span>
        </div>
        <button className="text-gray-400 hover:text-gray-600
transition-colors">
          <MoreHorizontal size={18} />
        </button>
      </div>

      <Droppable droppableId={id}>
        {(provided) => (
          <div
            {...provided.droppableProps}
            ref={provided.innerRef}
            className="flex-1 overflow-y-auto min-h-0 space-y
-3 pr-1"
          >
            {tasks.map((task, index) => (
              <TaskCard
                key={task.id}
                id={task.id}
```

```
                  index={index}
                  title={task.title}
                  priority={task.priority}
                />
              ))}
              {provided.placeholder}

              <button
                onClick={onAdd}
                className="w-full py-2 border border-gray-300 r
ounded-lg text-gray-500 text-sm font-medium hover:bg-white ho
ver:border-gray-400 transition-all flex items-center justify-
center gap-2 bg-transparent group mt-3"
              >
                <Plus size={16} className="text-gray-400 group-
hover:text-gray-600" />
                New
              </button>
            </div>
          )}
        </Droppable>
      </div>
    );
}
```

Target 2: WebSockets

Right now, if we move a card, only us see it. We want everyone to see it instantly.

Step 1: update the gateway

our current `src/tasks/tasks.gateway.ts` file only has `broadcastTaskCreated`. we need to add methods for Updated (moved/edited) and Deleted tasks.

we should add some functions in `src/tasks/tasks.gateway.ts`

```
import {
  WebSocketGateway,
  WebSocketServer,
  OnGatewayConnection,
  OnGatewayDisconnect
} from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';

@WebSocketGateway({ cors: { origin: '*' } })
export class TasksGateway implements OnGatewayConnection, OnG
atewayDisconnect {
  @WebSocketServer()
  server: Server;

  handleConnection(client: Socket) {
    console.log(`Client connected: ${client.id}`);
  }

  handleDisconnect(client: Socket) {
    console.log(`Client disconnected: ${client.id}`);
  }

  broadcastTaskCreated(task: any) {
    this.server.emit('task:created', task);
  }

  broadcastTaskUpdated(task: any) {
    this.server.emit('task:updated', task);
  }

  broadcastTaskDeleted(id: number) {
    this.server.emit('task:deleted', { id });
  }
}
```

Step 2: Update the Service

we need to tell `TasksService` to actually **use** these new gateway tools when things change.

so we need to add this inside `src/tasks/tasks.service.ts`

```
import { CreateTaskDto } from './dto/create-task.dto';
import { UpdateTaskDto } from './dto/update-task.dto';
import { PrismaClient } from '@prisma/client';
import { Injectable, NotFoundException } from '@nestjs/commo
n';
import { TasksGateway } from './tasks.gateway';

@Injectable()
export class TasksService {
  prisma = new PrismaClient();

  constructor(private tasksGateway: TasksGateway) {}

  async findAll() {
    return await this.prisma.task.findMany({
      orderBy: { id: 'asc' }
    });
  }

  async create(createTaskDto: CreateTaskDto) {
    const task = await this.prisma.task.create({
      data: {
        title: createTaskDto.title,
        description: createTaskDto.description || '',
        status: createTaskDto.status,
        priority: createTaskDto.priority || 'Medium',
        project_id: createTaskDto.projectId || 1,
        created_by: createTaskDto.createdBy || 1,
        position: 0,
        assigned_to: createTaskDto.assignedTo || null,
      },
    });

    this.tasksGateway.broadcastTaskCreated(task);

    return task;
  }

  async update(id: number, updateTaskDto: UpdateTaskDto) {
    try {
      const task = await this.prisma.task.update({
        where: { id: id },
        data: { ...updateTaskDto },
      });

      this.tasksGateway.broadcastTaskUpdated(task);

      return task;
    } catch (error) {
      if (error.code === 'P2025') {
        throw new NotFoundException(`Task with ID ${id} not f
ound`);
      }
      throw error;
    }
  }

  async remove(id: number) {
    try {
      const task = await this.prisma.task.delete({
        where: { id: id },
      });

      this.tasksGateway.broadcastTaskDeleted(id);

      return task;
    } catch (error) {
      if (error.code === 'P2025') {
        throw new NotFoundException(`Task with ID ${id} not f
ound`);
      }
      throw error;
    }
  }
}
```

Step 3: Listen on Frontend

now we teach our React app to listen for these shouts.

1. Install the library
   `npm install socket.io-client`
2. update `src/App.tsx`

```
// ... imports
import { io } from 'socket.io-client';

// ... inside App function ...

  useEffect(() => {
    fetchTasks();

    const socket = io('http://localhost:3003');

    socket.on('task:created', (newTask: Task) => {
      setTasks((prevTasks) => {
        if (prevTasks.some(t => t.id === newTask.id)) retu
rn prevTasks;
        return [...prevTasks, newTask];
      });
    });
    socket.on('task:updated', (updatedTask: Task) => {
      setTasks((prevTasks) =>
        prevTasks.map((t) => (t.id === updatedTask.id ? up
datedTask : t))
      );
    });

    socket.on('task:deleted', ({ id }: { id: number }) =>
{
      setTasks((prevTasks) => prevTasks.filter((t) => t.id
!== id));
    });

    return () => {
      socket.disconnect();
    };
  }, []);

// ... rest of the code
```

Step 4: The Final Test

1. restart the Backend (`Ctrl + C` → `npm run start:dev`)
2. open Two Browser Windows
3. drag a card in Window A
4. watch Window B
5. if window B updates automatically we have successfully built a real time KanbanBoard

Day 7/8/9/10 Recap

we have officially finished the hardest day of the project.

- [x] ~~Drag & Drop: @hello-pangea/dnd is working.~~
- [x] ~~Modal UI: "New Task" popup matches the design.~~

- [x] ~~Database: Schema matches the team PDF.~~
- [x] ~~Real-Time: WebSockets are broadcasting live updates.ts.~~

Day 11

target 1: The "Delete" Button

Step 1: Create the Delete Function in `App.tsx`

Open `src/App.tsx`.

1. Find your `handleSaveTask` function.
2. Add this **new function** right below it:

```
// ... inside App component

  const handleDeleteTask = async (id: number) => {
    if (!confirm('Are you sure you want to delete this tas
k?')) return;
    try {
      await axios.delete(`http://localhost:3003/tasks/${id}
`);
    } catch (error) {
      console.error("Failed to delete task:", error);
      alert("Could not delete task.");
    }
  };
```

Step 2: Pass the Function Down

We need to pass this function from App.tsx to `KanbanColumn` and `TaskCard`

1. update `KanbanColumn` usage in `App.tsx`

```
<KanbanColumn
  id="TODO"
  title="To Do"
  tasks={getTasksByStatus('TODO')}
  count={getTasksByStatus('TODO').length}
  onAdd={() => openModal('TODO')}
  onDelete={handleDeleteTask} // <--- ADD THIS
/>
```

(do this for all 4 columns)

2. update `src/components/KanbanColumn.tsx`

```
// ... imports

interface KanbanColumnProps {
  id: string;
  title: string;
  tasks: Task[];
  count: number;
  onAdd: () => void;
  onDelete: (id: number) => void; // <--- 1. Add Prop Type
}

export default function KanbanColumn({ id, title, tasks, count, onAdd, onDelete }: KanbanColumnProps) { // <--- 2. Receive Prop
  return (
    <div className="...">
      <Droppable droppableId={id}>
        {(provided) => (
          <div {...provided.droppableProps} ref={provided.innerRef} className="...">
            {tasks.map((task, index) => (
              <TaskCard
                key={task.id}
                id={task.id}
                index={index}
                title={task.title}
                priority={task.priority}
                onDelete={onDelete} // <--- 3. Pass to Car
d
              />
            ))}
          </div>
        )}
      </Droppable>
    </div>
  );
}
```

3. add the button to TaskCard.tsx

   open `src/components/TaskCard.tsx`

```
import { Trash2 } from 'lucide-react'; // <--- 1. Import T
rash2
import { Draggable } from '@hello-pangea/dnd';

interface TaskCardProps {
  id: number;
  index: number;
  title: string;
  priority?: string;
  onDelete: (id: number) => void; // <--- 2. Add Prop Type
}

// ... helper function ...

export default function TaskCard({ id, index, title, prior
ity = 'Medium', onDelete }: TaskCardProps) { // <--- 3. Re
ceive Prop
  return (
    <Draggable draggableId={id.toString()} index={index}>
```

```
      {(provided) => (
        <div
          ref={provided.innerRef}
          {...provided.draggableProps}
          {...provided.dragHandleProps}
          className="bg-white p-4 rounded-xl shadow-sm bor
der border-gray-100 cursor-grab hover:shadow-md transition
-all group relative" // <--- Added "relative"
        >
          <div className="flex justify-between items-start
mb-3">
            <h3 className="text-sm font-bold text-gray-800
leading-tight pr-6">
              {title}
            </h3>

            <button
              onClick={(e) => {
                e.stopPropagation();
                onDelete(id);
              }}
              className="text-gray-300 hover:text-red-500
transition-colors absolute top-4 right-4 opacity-0 group-h
over:opacity-100"
            >
              <Trash2 size={16} />
            </button>
          </div>
        </div>
      )}
    </Draggable>
  );
}
```

Target 2: Edit Modal

clicking a card to change its details.

Step 1: Upgrade `NewTaskModal.tsx`

we need to change the modal so it can handle **Editing** mode

- **If `task` is provided:** Fill the inputs with that task's data.
- **If `task` is null:** Show empty inputs (Create mode).

```
import { useState, useEffect } from 'react';
import { X, Paperclip, Bold, Italic, List, AlignLeft, Link }
from 'lucide-react';

interface Task {
  id?: number;
  title: string;
  description?: string;
  status: string;
  priority?: string;
}

interface NewTaskModalProps {
  isOpen: boolean;
  onClose: () => void;
  onSave: (task: Task) => void;
  taskToEdit?: Task | null;
  defaultStatus: string;
}

export default function NewTaskModal({ isOpen, onClose, onSav
e, taskToEdit, defaultStatus }: NewTaskModalProps) {
  const [title, setTitle] = useState('');
  const [description, setDescription] = useState('');
  const [priority, setPriority] = useState('Medium');

  useEffect(() => {
    if (isOpen) {
      if (taskToEdit) {
        setTitle(taskToEdit.title);
        setDescription(taskToEdit.description || '');
        setPriority(taskToEdit.priority || 'Medium');
      } else {
        setTitle('');
        setDescription('');
        setPriority('Medium');
      }
    }
  }, [isOpen, taskToEdit]);

  if (!isOpen) return null;

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    if (!title.trim()) return;

    onSave({
      id: taskToEdit?.id,
      title,
      description,
      status: taskToEdit ? taskToEdit.status : defaultStatus,
      priority,
    });

    onClose();
  };

  return (
    <div className="fixed inset-0 bg-black/40 flex items-cent
er justify-center z-50 backdrop-blur-sm">
      <div className="bg-white w-[650px] rounded-xl shadow-2x
l overflow-hidden flex flex-col max-h-[90vh]">
```

```
        {/* Header */}
        <div className="flex justify-between items-center p-5
border-b border-gray-100">
          <h2 className="text-lg font-bold text-gray-800">
            {taskToEdit ? 'Edit Task' : 'New Task'}
          </h2>
          <button onClick={onClose} className="text-gray-400
hover:text-gray-600 transition-colors">
            <X size={20} />
          </button>
        </div>

        <form onSubmit={handleSubmit} className="p-6 flex-1 o
verflow-y-auto">
          <div className="mb-6">
            <input
              autoFocus
              type="text"
              value={title}
              onChange={(e) => setTitle(e.target.value)}
              placeholder="Task Title"
              className="w-full p-3 text-lg font-semibold tex
t-gray-800 border border-gray-300 rounded-lg focus:outline-no
ne focus:ring-2 focus:ring-blue-500 placeholder-gray-400"
            />
          </div>

          <div className="mb-6">
            <label className="block text-xs font-bold text-gr
ay-500 uppercase tracking-wider mb-2">Priority</label>
            <div className="flex gap-2">
              {['Low', 'Medium', 'High'].map((p) => (
                <button
                  key={p}
                  type="button"
                  onClick={() => setPriority(p)}
                  className={`px-3 py-1.5 rounded text-sm fon
t-medium border transition-all ${
                    priority === p
                    ? 'bg-blue-50 border-blue-500 text-blue-7
00'
                    : 'bg-white border-gray-200 text-gray-600
hover:bg-gray-50'
                  }`}
                >
                  {p}
                </button>
              ))}
            </div>
          </div>

          <div className="mb-6 border border-gray-200 rounded
-lg overflow-hidden focus-within:ring-2 focus-within:ring-blu
e-500 transition-all">
             <div className="bg-white border-b border-gray-20
0 p-2 flex gap-1 text-gray-500">
                <button type="button" className="p-1.5 hover:
bg-gray-100 rounded"><Bold size={16} /></button>
                <button type="button" className="p-1.5 hover:
bg-gray-100 rounded"><Italic size={16} /></button>
                <button type="button" className="p-1.5 hover:
bg-gray-100 rounded"><AlignLeft size={16} /></button>
                <button type="button" className="p-1.5 hover:
bg-gray-100 rounded"><List size={16} /></button>
                <button type="button" className="p-1.5 hover:
bg-gray-100 rounded"><Link size={16} /></button>
             </div>
            <textarea
              value={description}
              onChange={(e) => setDescription(e.target.valu
e)}
              placeholder="Add more details to this task..."
              rows={6}
              className="w-full p-4 resize-none focus:outline
-none text-gray-700 text-sm leading-relaxed"
            />
          </div>

          <div className="flex justify-end gap-3 pt-2">
             <button type="button" onClick={onClose} classNam
e="px-5 py-2 border border-gray-300 rounded-lg text-gray-700
hover:bg-gray-50 font-medium transition-colors">
               Cancel
             </button>
             <button type="submit" className="px-6 py-2 bg-[#
5B5CFF] text-white rounded-lg font-bold hover:bg-[#4d4eed] sh
adow-md transition-all">
               {taskToEdit ? 'Save Changes' : 'Create Task'}
{/* Dynamic Button Text */}
             </button>
          </div>
        </form>
      </div>
    </div>
  );
}
```

Step 2: Handle the Logic in `App.tsx`

we need to:

1. Add state to remember which task we are editing (`editingTask`).
2. Update `handleSaveTask` to handle both POST (Create) and PATCH (Update).

```
// ... imports (keep existing)

// ... Interface (keep existing)
```

```
function App() {
  const [tasks, setTasks] = useState<Task[]>([]);
  const [loading, setLoading] = useState(true);

  const [isModalOpen, setIsModalOpen] = useState(false);
  const [activeStatus, setActiveStatus] = useState('TODO');
  const [editingTask, setEditingTask] = useState<Task | null>
(null);

  // ... useEffect (keep existing) ...
  // ... fetchTasks (keep existing) ...

  const openNewTaskModal = (status: string) => {
    setActiveStatus(status);
    setEditingTask(null);
    setIsModalOpen(true);
  };

  const openEditTaskModal = (task: Task) => {
    setEditingTask(task)
    setIsModalOpen(true);
  };

  const handleSaveTask = async (taskData: Partial<Task>) => {
    try {
      if (taskData.id) {
        await axios.patch(`http://localhost:3003/tasks/${task
Data.id}`, {
          title: taskData.title,
          description: taskData.description,
          priority: taskData.priority,
        });
      } else {
        await axios.post('http://localhost:3003/tasks', {
          title: taskData.title,
          description: taskData.description,
          status: taskData.status,
          priority: taskData.priority
        });
      }
      setIsModalOpen(false);
      setEditingTask(null);
    } catch (error) {
      console.error("Error saving task:", error);
    }
  };

  // ... handleDeleteTask (keep existing) ...
  // ... onDragEnd (keep existing) ...

  // ... getTasksByStatus ...

  return (
    <div className="flex h-screen bg-white font-sans text-gra
y-900">
      <Sidebar />
      <main className="flex-1 flex flex-col overflow-hidden b
g-white">
        {/* ... Header ... */}

        <div className="flex-1 p-8 overflow-hidden flex flex-
col">
          {/* ... Title Section ... */}

          <DragDropContext onDragEnd={onDragEnd}>
            {loading ? (
               <div className="flex-1 flex items-center justi
fy-center text-gray-400">Loading board...</div>
            ) : (
              <div className="flex-1 grid grid-cols-1 md:grid
-cols-2 lg:grid-cols-4 gap-6 overflow-hidden pb-2">
                {['TODO', 'IN_PROGRESS', 'REVIEW', 'DONE'].ma
```

```
p(status => (
                  <KanbanColumn
                    key={status}
                    id={status}
                    title={status.replace('_', ' ')}
                    count={getTasksByStatus(status).length}
                    tasks={getTasksByStatus(status)}
                    onAdd={() => openNewTaskModal(status)}
                    onDelete={handleDeleteTask}
                    onEdit={openEditTaskModal}
                  />
                ))}
              </div>
            )}
          </DragDropContext>
        </div>
      </main>

      <NewTaskModal
        isOpen={isModalOpen}
        onClose={() => setIsModalOpen(false)}
        onSave={handleSaveTask}
        taskToEdit={editingTask}
        defaultStatus={activeStatus}
      />
    </div>
  );
}

export default App;
```

Step 3: Pass the "Edit Click" Down

update `src/components/KanbanColumn.tsx`

```
// ... imports

interface KanbanColumnProps {
  id: string;
  title: string;
  tasks: Task[];
  count: number;
  onAdd: () => void;
  onDelete: (id: number) => void;
  onEdit: (task: Task) => void;
}

export default function KanbanColumn({ id, title, tasks, coun
t, onAdd, onDelete, onEdit }: KanbanColumnProps) { // <--- Re
ceive Prop
  return (
    // ... container ...
      <Droppable droppableId={id}>
        {(provided) => (
          <div {...provided.droppableProps} ref={provided.inn
erRef} className="...">
            {tasks.map((task, index) => (
              <TaskCard
                key={task.id}
                id={task.id}
                index={index}
                title={task.title}
                priority={task.priority}
                onDelete={onDelete}
                onClick={() => onEdit(task)}
              />
            ))}
            {/* ... placeholder & add button ... */}
          </div>
        )}
      </Droppable>
    // ...
  );
}
```

update `src/components/TaskCard.tsx`:

```
// ... imports

interface TaskCardProps {
  id: number;
  index: number;
  title: string;
  priority?: string;
  onDelete: (id: number) => void;
  onClick: () => void;
}

export default function TaskCard({ id, index, title, priority
= 'Medium', onDelete, onClick }: TaskCardProps) { // <--- Rec
eive Prop
  return (
    <Draggable draggableId={id.toString()} index={index}>
      {(provided) => (
        <div
          ref={provided.innerRef}
          {...provided.draggableProps}
          {...provided.dragHandleProps}
          onClick={onClick}
          className="bg-white p-4 rounded-xl shadow-sm border
border-gray-100 cursor-pointer hover:shadow-md transition-all
group relative" // Changed cursor-grab to cursor-pointer
        >
          {/* ... rest of the card ... */}
        </div>
      )}
    </Draggable>
  );
}
```

Step 4: Try it out!

- Click a task. (The modal should open with the Title and Priority filled in).
- Change the Title.
- Click "Save Changes".
- It should update instantly on our screen (and any other open window).

Target 3: Assignee Feature

right now every task says "User Name” and has "UN" in the circle. we want to be able to pick "User 1"user 2", or "user 3" when creating a task and see *their* real initials on the card

Step 1: Create the User Data

- Create a new folder: `src/data` inside `kanban-web`.
- Create a file inside it: `users.ts`.
- Paste this code:

```
// src/data/users.ts

export const USERS = [
  { id: 1, name: 'Sarah Bisara', initials: 'SB', color: 'bg-emerald-500' },
  { id: 2, name: 'John Doe', initials: 'JD', color: 'bg-blue-500' },
  { id: 3, name: 'Mike Ross', initials: 'MR', color: 'bg-purple-500' },
  { id: 4, name: 'Rachel Zane', initials: 'RZ', color: 'bg-amber-500' },
];
```

```
export const getUserById = (id: number) => {
  return USERS.find(user => user.id === id);
};
```

Step 2: Add User Selection to the Modal

update `src/components/NewTaskModal.tsx`

```
import { useState , useEffect} from "react";
import {
  X,
  Paperclip,
  Bold,
  Italic,
  List,
  AlignLeft,
  Link,
  ChevronDown,
} from "lucide-react";
import { USERS } from '../data/users';

interface Task{
  id?: number;
  title: string;
  description: string;
  status:string;
  priority: string;
  assignedTo?: number;
  assigned_to?: number;
}

interface NewTaskModalProps {
  isOpen: boolean;
  onClose: () => void;
  onSave: (task: Task) => void;
  taskToEdit?: Task | null;
  defaultStatus: string;
}

export default function NewTaskModal({
  isOpen,
  onClose,
  onSave,
  taskToEdit,
  defaultStatus,
}: NewTaskModalProps) {
  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [priority, setPriority] = useState("Medium");
  const [assignedTo, setAssignedTo] = useState<number | undef
ined>(undefined);

  useEffect(() => {
    if (isOpen) {
      if (taskToEdit) {
        setTitle(taskToEdit.title);
        setDescription(taskToEdit.description || '');
        setPriority(taskToEdit.priority || 'Medium');
        setAssignedTo(taskToEdit.assignedTo || taskToEdit.ass
igned_to);
      } else {
        setTitle('');
        setDescription('');
        setPriority('Medium');
        setAssignedTo(undefined);
      }
    }
  }, [isOpen, taskToEdit]);

  if (!isOpen) return null;
  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    if (!title.trim()) return;

    onSave({
      id: taskToEdit?.id,
      title,
      description,
      status: taskToEdit ? taskToEdit.status : defaultStatus,
      priority,
      assignedTo,
    });

    onClose();
  };

  return (
    <div className="fixed inset-0 bg-black/40 flex items-cent
er justify-center z-50 backdrop-blur-sm">
      <div className="bg-white w-[650px] rounded-xl shadow-2x
l overflow-hidden flex flex-col max-h-[90vh]">

        <div className="flex justify-between items-center p-5
border-b border-gray-100">
          <h2 className="text-lg font-bold text-gray-800">{ta
skToEdit ? 'Edit Task' : 'New Task'}</h2>
          <button
            onClick={onClose}
            className="text-gray-400 hover:text-gray-600 tran
sition-colors"
          >
            <X size={20} />
          </button>
        </div>

        <form onSubmit={handleSubmit} className="p-6 flex-1 o
```

```
verflow-y-auto">

          <div className="mb-6">
            <input
              autoFocus
              type="text"
              value={title}
              onChange={(e) => setTitle(e.target.value)}
              placeholder="Task Title"
              className="w-full p-3 text-lg font-semibold tex
t-gray-800 border border-gray-300 rounded-lg focus:outline-no
ne focus:ring-2 focus:ring-blue-500 placeholder-gray-400"
            />
          </div>

          <div className="mb-6">
            <label className="block text-xs font-bold text-gr
ay-500 uppercase tracking-wider mb-2">
              Priority
            </label>
            <div className="flex gap-2">
              {["Low", "Medium", "High"].map((p) => (
                <button
                  key={p}
                  type="button"
                  onClick={() => setPriority(p)}
                  className={`px-3 py-1.5 rounded text-sm fon
t-medium border transition-all ${
                    priority === p
                      ? "bg-blue-50 border-blue-500 text-blue
-700"
                      : "bg-white border-gray-200 text-gray-6
00 hover:bg-gray-50"
                  }`}
                >
                  {p}
                </button>
              ))}
            </div>
          </div>

          <div className="mb-6">
            <label className="block text-xs font-bold text-gr
ay-500 uppercase tracking-wider mb-2">Assign To</label>
            <div className="relative">
              <select
                value={assignedTo || ""}
                onChange={(e) => setAssignedTo(e.target.value
? Number(e.target.value) : undefined)}
                className="w-full appearance-none bg-white bo
rder border-gray-200 text-gray-700 py-2 px-3 pr-8 rounded-lg
text-sm font-medium focus:outline-none focus:ring-2 focus:rin
g-blue-500 hover:border-gray-300 cursor-pointer"
              >
                <option value="">Unassigned</option>
                {USERS.map((user) => (
                  <option key={user.id} value={user.id}>
                    {user.name}
                  </option>
                ))}
              </select>
              <div className="pointer-events-none absolute in
set-y-0 right-0 flex items-center px-2 text-gray-400">
                <ChevronDown size={16} />
              </div>
            </div>
          </div>

          <div className="mb-6 border border-gray-200 rounded
-lg overflow-hidden focus-within:ring-2 focus-within:ring-blu
e-500 transition-all">
            <div className="bg-white border-b border-gray-200
```

```
p-2 flex gap-1 text-gray-500">
              <button type="button" className="p-1.5 hover:bg
-gray-100 rounded">
                <Bold size={16} />
              </button>
              <button type="button" className="p-1.5 hover:bg
-gray-100 rounded">
                <Italic size={16} />
              </button>
              <button type="button" className="p-1.5 hover:bg
-gray-100 rounded">
                <AlignLeft size={16} />
              </button>
              <button type="button" className="p-1.5 hover:bg
-gray-100 rounded">
                <List size={16} />
              </button>
              <button type="button" className="p-1.5 hover:bg
-gray-100 rounded">
                <Link size={16} />
              </button>
            </div>
            <textarea
              value={description}
              onChange={(e) => setDescription(e.target.valu
e)}
              placeholder="Add more details to this task..."
              rows={6}
              className="w-full p-4 resize-none focus:outline
-none text-gray-700 text-sm leading-relaxed"
            />
          </div>

          <div className="mb-6">
            <label className="block text-xs font-bold text-gr
ay-500 uppercase tracking-wider mb-2">
```

```
              Attachments
            </label>
            <div className="border border-dashed border-gray-
300 rounded-lg p-4 flex items-center justify-center gap-2 tex
t-gray-500 hover:bg-gray-50 cursor-pointer transition-color
s">
              <Paperclip size={16} />
              <span className="text-sm">Add File</span>
            </div>
          </div>

          <div className="flex justify-end gap-3 pt-2">
            <button
              type="button"
              onClick={onClose}
              className="px-5 py-2 border border-gray-300 rou
nded-lg text-gray-700 hover:bg-gray-50 font-medium transition
-colors"
            >
              Cancel
            </button>
            <button
              type="submit"
              className="px-6 py-2 bg-[#5B5CFF] text-white ro
unded-lg font-bold hover:bg-[#4d4eed] shadow-md transition-al
l"
            >
              {taskToEdit ? 'Save Changes' : 'Create Task'}
            </button>
          </div>
        </form>
      </div>
    </div>
  );
}
```

Step 3: Handle the Data in `App.tsx`

now `App.tsx` needs to receive this `assignedTo` value and send it to the backend.

```
// ... imports

interface Task {
  id: number;
  title: string;
  description?: string;
  status: string;
  priority?: string;
  assignedTo?: number;
}

// ... inside App component ...

  const handleSaveTask = async (taskData: Partial<Task>) => {
    try {
      if (taskData.id) {
        await axios.patch(`http://localhost:3003/tasks/${task
Data.id}`, {
          title: taskData.title,
          description: taskData.description,
          priority: taskData.priority,
          assignedTo: taskData.assignedTo
        });
      } else {
        await axios.post('http://localhost:3003/tasks', {
          title: taskData.title,
          description: taskData.description,
          status: taskData.status,
          priority: taskData.priority,
          assignedTo: taskData.assignedTo
        });
      }
      setIsModalOpen(false);
      setEditingTask(null);
    } catch (error) {
      console.error("Error saving task:", error);
    }
  };
```

before we verify this: the backend DTO (`create-task.dto.ts`) uses `assignedTo` (camelCase) to receive it but the service (`tasks.service.ts`) needs to save it as `assigned_to` (snake_case) in the database

Step 4: fix backend logic (`src/tasks/tasks.service.ts`)

we need to manually catch `assignedTo` and map it to `assigned_to` before giving it to Prisma.

```
async update(id: number, updateTaskDto: UpdateTaskDto) {
    try {
      const { assignedTo, ...rest } = updateTaskDto;

      const dataToUpdate: any = { ...rest };

      if (assignedTo !== undefined) {
        dataToUpdate.assigned_to = assignedTo;
      }

      const task = await this.prisma.task.update({
        where: { id: id },
        data: dataToUpdate,
      });

      this.tasksGateway.broadcastTaskUpdated(task);

      return task;
    } catch (error) {
      if (error.code === 'P2025') {
        throw new NotFoundException(`Task with ID ${id} not f
ound`);
      }
      throw error;
    }
  }
```

Step 5: update `TaskCard.tsx`

we need to teach the card to:

1. Take the `assignedTo` ID.
2. Look up the user in our "Phonebook" (`users.ts`).
3. Show *their* initials and color.

```
import { Trash2 } from 'lucide-react';
import { Draggable } from '@hello-pangea/dnd';
import { getUserById } from '../data/users';

interface TaskCardProps {
  id: number;
  index: number;
  title: string;
  priority?: string;
  assignedTo?: number;
  onDelete: (id: number) => void;
  onClick: () => void;
}

const getPriorityColor = (priority: string) => {
  switch (priority?.toLowerCase()) {
    case 'high': return 'bg-red-500';
    case 'medium': return 'bg-amber-500';
    case 'low': return 'bg-emerald-500';
    default: return 'bg-gray-400';
  }
};

export default function TaskCard({ id, index, title, priority
= 'Medium', assignedTo, onDelete, onClick }: TaskCardProps) {
  const user = assignedTo ? getUserById(assignedTo) : null;
  return (
    <Draggable draggableId={id.toString()} index={index}>
      {(provided) => (
        <div
          ref={provided.innerRef}
          {...provided.draggableProps}
          {...provided.dragHandleProps}
          onClick={onClick}
          className="bg-white p-4 rounded-xl shadow-sm border
border-gray-100 cursor-grab hover:shadow-md transition-all gr
oup relative"
        >
          <div className="flex justify-between items-start mb
-3">
            <h3 className="text-sm font-bold text-gray-800 le
ading-tight pr-6">
              {title}
            </h3>

            <button
              onClick={(e) => {
                e.stopPropagation();
                onDelete(id);
              }}
              className="text-gray-300 hover:text-red-500 tra
nsition-colors absolute top-4 right-4 opacity-0 group-hover:o
pacity-100"
            >
```

```
              <Trash2 size={16} />
            </button>
          </div>

          <div className="mb-3">
            <div className="text-[10px] font-bold text-gray-4
00 uppercase tracking-wider mb-1">Priority</div>
            <span className={`inline-block px-2 py-0.5 text-
[10px] font-bold uppercase tracking-wide text-white rounded
${getPriorityColor(priority)}`}>
              {priority}
            </span>
          </div>

          <div>
            <div className="text-[10px] font-bold text-gray-4
00 uppercase tracking-wider mb-1">Assignee</div>
            <div className="flex items-center justify-betwee
n">

            {user ? (
                <>
                  <span className="text-xs font-semibold text
-gray-700">{user.name}</span>
                  <div className={`w-6 h-6 rounded-full borde
r border-white flex items-center justify-center text-[8px] te
xt-white ${user.color}`}>
                    {user.initials}
                  </div>
                </>
              ) : (
                <>
                  <span className="text-xs font-medium text-g
ray-400 italic">Unassigned</span>
                  <div className="w-6 h-6 rounded-full bg-gra
y-100 border border-white flex items-center justify-center te
xt-[10px] text-gray-400">
                    ?
                  </div>
                </>
              )}

            </div>
          </div>
        </div>
      )}
    </Draggable>
  );
}
```

Step 6: pass the data from `KanbanColumn.tsx`

now `KanbanColumn` needs to pass the `assignedTo` value down to the card.

- Update the `Task` interface to include `assignedTo` (and the backup `assigned_to`).
- Pass the prop in the `<TaskCard />` component.

```
// ... imports

interface Task {
  id: number;
  title: string;
  status: string;
  priority?: string;
  assignedTo?: number;
  assigned_to?: number;
}

// ... props interface ...

export default function KanbanColumn({ id, title, tasks, coun
t, onAdd, onDelete, onEdit }: KanbanColumnProps) {
  return (
    // ...
      <Droppable droppableId={id}>
        {(provided) => (
          <div {...provided.droppableProps} ref={provided.innerRef} className="...">
            {tasks.map((task, index) => (
              <TaskCard
                key={task.id}
                id={task.id}
                index={index}
                title={task.title}
                priority={task.priority}
                assignedTo={task.assignedTo || task.assigned_to}
                onDelete={onDelete}
                onClick={() => onEdit(task)}
              />
            ))}
            {/* ... */}
          </div>
        )}
      </Droppable>
    // ...
  );
}
```

Step 7: try it out:

1. Restart Backend (so it learns how to update assignees).
2. Open a task.
3. Click the "Unassigned" dropdown.
4. Pick "User 1" (or anyone else).
5. Click Save.

Day 11 Recap

- ✓ ~~**Edit Modal:** we can open existing tasks, and the form remembers the data.~~
- ✓ ~~**Smart Logic:** our app knows the difference between "Creating" (POST) and "Updating" (PATCH).~~
- ✓ ~~**User Switcher:** A clean dropdown to pick real (mock) users.~~
- ✓ ~~**Dynamic UI:** The cards now show the real assignee's name and color.~~

Day 12/13

right now if we drag a task to the top of its column it snaps back to where it was that is because the database says "i don't care where you put it visually. I only know it by ID."

Drag & Drop Reordering

Step 1: update the backend controller

we need a new endpoint that accepts a list of IDs and updates their positions. so we need to add this patch in `backend/src/tasks/tasks.controller.ts`

```
@Patch('reorder')
  async reorder(@Body() body: { taskIds: number[] }) {
    return this.tasksService.reorder(body.taskIds);
  }

  //@Patch(':id')
```

Step 2: update the backend service

now we need the logic to loop through those IDs and update the database. we add the reorder method in `backend/src/tasks/tasks.service.ts`

```
// ... inside TasksService class
```

```
  async reorder(taskIds: number[]) {
    const updates = taskIds.map((id, index) => {
      return this.prisma.task.update({
        where: { id },
        data: { position: index },
      });
    });
    const results = await this.prisma.$transaction(updates);
    results.forEach(task => this.tasksGateway.broadcastTaskUp
dated(task));
    return results;
  }
```

Step 3: sort tasks by position in `findAll`

```
async findAll() {
    return await this.prisma.task.findMany({
      orderBy: { position: 'asc' }, // <--- change this from
'id' to 'position'
    });
  }
```

currently when we drag a task to a new column, we only update its status we ignore the destination.index so the database says, "okay, it's DONE now," but it leaves the position value alone which usually sends it to the bottom or a random spot.

Step 1: update `App.tsx`

```
  const onDragEnd = async (result: DropResult) => {
    const { destination, source, draggableId } = result;
    if (!destination) return;
    if (destination.droppableId === source.droppableId && des
tination.index === source.index) return;
```

```
    const movedTaskId = parseInt(draggableId);

    if (source.droppableId === destination.droppableId){
      const columnId = source.droppableId;
      const columnTasks = getTasksByStatus(columnId);
      const newColumnTasks = Array.from(columnTasks);
      const [movedTask] = newColumnTasks.splice(source.index,
1);

      newColumnTasks.splice(destination.index, 0, movedTask);
      const otherTasks = tasks.filter(t => t.status !== colum
nId);
      setTasks([...otherTasks, ...newColumnTasks]);
      try {
        const taskIds = newColumnTasks.map(t => t.id);
        await axios.patch('http://localhost:3003/tasks/reorde
r', { taskIds });
      } catch (error) {
          console.error("Failed to reorder:", error);
          toast.error("Failed to save new order. Please try a
gain.");
          fetchTasks();
        }
        return;
    }

    const newStatus = destination.droppableId;
    const taskToMove = tasks.find(t => t.id === movedTaskId);
    if(!taskToMove) return ;
    const updatedTask = {...taskToMove, status: newStatus};
    const destColumnTasks = tasks.filter(t => t.status === ne
wStatus && t.id !== movedTaskId);

    destColumnTasks.splice(destination.index, 0, updatedTas
k);
    const otherTasks = tasks.filter(t => t.status !== newStat
us && t.id !== movedTaskId);
    setTasks([...otherTasks, ...destColumnTasks]);

    try {
      await axios.patch(`http://localhost:3003/tasks/${movedT
askId}`, { status: newStatus });
        const taskIds = destColumnTasks.map(t => t.id);
      await axios.patch(`http://localhost:3003/tasks/reorder
`, {taskIds: taskIds});
    } catch (error) {
        console.error("Failed to move task:", error);
        fetchTasks();
      }
  };
```

NOTE: managing database schema changes

when we add a new property (like `priority` or `created_at`) to our code, we must update the "Prisma Layer" in two specific ways to avoid errors.

1. the definition mismatch

   If we update our code to use a new field (e.g., in `tasks.service.ts`) but forget to add it to `schema.prisma` the build will fail

   TypeScript checks the generated prisma client types and sees the field is missing

   so we need to update `schema.prisma` to include the new field.

   ```
   model Task {
     // ...
     priority    String   @default("Medium")
   }
   ```

2. update TypeScript definitions

   our code (TypeScript) does not automatically know about changes in the schema file. we must tell Prisma to re read the schema and update the types in `node_modules`

```
npx prisma generate
```

3. the database sync error (runtime error)

   If we update `schema.prisma` and the TypeScript types but do not push changes to the database the app will crash at runtime

   the TypeScript code is correct, but the actual PostgreSQL table is outdated and missing the new column.

   we need to run the sync command to update the database structure:

   ```
   npx prisma db push
   ```

Day 12

right now, our search bar and filter button do nothing. let's fix that.

Target 1: Search bar

Step 1: add the search state

open `src/App.tsx` and add this:

```
const [searchQuery, setSearchQuery] = useState('')
```

Step 2: connect the input

inside `App.tsx` find the search input and replace it with this

```
<div className="relative">
  <Search className="absolute left-3 top-1/2 -translate-y-1/2
text-gray-400" size={18} />
  <input
    type="text"
    placeholder="Search tasks..."
```

```
    value={searchQuery}
    onChange={(e) => setSearchQuery(e.target.value)}
    className="pl-10 pr-4 py-2 bg-white border border-gray-20
0 rounded-lg text-sm focus:outline-none focus:ring-1 focus:ri
ng-blue-500 w-64 transition-all placeholder-gray-400 hover:bo
rder-gray-300"
  />
</div>
```

Step 3: the logic

no find `getTasksByStatus` currently it just filters by status. we will upgrade it to also check if the title matches our search

```
const getTasksByStatus = (status: string) => {
    return tasks
      .filter(task => task.status === status)
      .filter(task => {
        if (!searchQuery) return true;
        const searchLower = searchQuery.toLowerCase();
        return (
          task.title.toLowerCase().includes(searchLower) ||
          (task.description && task.description.toLowerCase
().includes(searchLower))
        );
      });
  };
```

Target 2: Filter button

Step 1: add filter state to `App.tsx`

```
const [filterPriority, setFilterPriority] = useState<string |
null>(null);
```

```
const [isFilterOpen, setIsFilterOpen] = useState(false);
```

Step 2: build the filter dropdown UI

inside `App.tsx` find the `<button>` that says "Filter" and replace it with this:

```
           <div className="relative">
             <button
               onClick={() => setIsFilterOpen(!isFilterOpen)}
               className={`flex items-center gap-2 px-4 py-2 b
order rounded-lg text-sm font-medium transition-colors ${
                 filterPriority ? 'bg-blue-50 border-blue-200
text-blue-600' : 'bg-white border-gray-200 text-gray-700 hove
r:bg-gray-50'
               }`}
             >
               <Filter size={16} />
               {filterPriority ? `${filterPriority} Priority`
: 'Filter'}
               {filterPriority && (
                 <X
                   size={14}
                   className="ml-1 hover:text-red-500"
                   onClick={(e) => {
                     e.stopPropagation();
                     setFilterPriority(null); // Clear filter
                   }}
                 />
               )}
             </button>

             {isFilterOpen && (
               <div className="absolute right-0 mt-2 w-48 bg-w
hite rounded-xl shadow-lg border border-gray-100 py-1 z-50 an
imate-in fade-in zoom-in-95 duration-200">
                 <div className="px-4 py-2 text-xs font-semibo
```

```
ld text-gray-500 uppercase tracking-wider">
                  Filter by Priority
                </div>
                {['High', 'Medium', 'Low'].map((p) => (
                  <button
                    key={p}
                    onClick={() => {
                      setFilterPriority(p);
                      setIsFilterOpen(false);
                    }}
                    className="w-full text-left px-4 py-2 tex
t-sm text-gray-700 hover:bg-gray-50 flex items-center gap-2"
                  >
                    <span className={`w-2 h-2 rounded-full ${
                      p === 'High' ? 'bg-red-500' : p === 'Me
dium' ? 'bg-amber-500' : 'bg-emerald-500'
                    }`} />
                    {p}
                  </button>
                ))}

                {filterPriority && (
                  <button
                    onClick={() => {
                      setFilterPriority(null);
                      setIsFilterOpen(false);
                    }}
                    className="w-full text-left px-4 py-2 tex
t-sm text-red-600 hover:bg-red-50 border-t border-gray-50 mt-
1"
                  >
                    Clear Filter
                  </button>
                )}
              </div>
```

```
          )}
        </div>
```

Step 3: connect the logic to `getTasksByStatus`

now we update the filter function to respect both search AND priority.

```
const getTasksByStatus = (status: string) => {
    return tasks
      .filter(task => task.status === status)
      .filter(task => {
        if (!searchQuery) return true;
        const searchLower = searchQuery.toLowerCase();
        return (
          task.title.toLowerCase().includes(searchLower) ||
          (task.description && task.description.toLowerCase
().includes(searchLower))
        );
      })
      .filter(task => {
        if (!filterPriority) return true;
        return task.priority === filterPriority;
      });
  };
```

Day 13

right now when we save a task nothing happens. we are going to make our app feel smooth and professional.

Target 1: toast notifications:

Step 1: Install `react-hot-toast`

this is the industry standard for React notifications. run this on the frontend terminal:

```
npm install react-hot-toast
```

Step 2: add the "Toaster" to the app

1. Import `Toaster` and `toast`
2. add `<Toaster />` inside the return statement

```
import { Toaster, toast } from 'react-hot-toast';

// ... inside App function ...

  return (
    <div className="flex h-screen bg-gray-50 font-sans text-g
ray-900">
      <Toaster position="bottom-right" />

      {/* ... Sidebar ... */}
      {/* ... Main Content ... */}

    </div>
  );
```

Step 3: replace alerts with toasts

```
  const handleSaveTask = async (taskData: Partial<Task>) => {
    try {
      if (taskData.id) {
        await axios.patch(`http://localhost:3003/tasks/${task
Data.id}`, {
          title: taskData.title,
        description: taskData.description,
        priority: taskData.priority,
        assignedTo: taskData.assignedTo
        });
        toast.success("Task updated successfully!");
      } else {
        await axios.post('http://localhost:3003/tasks', {
          title: taskData.title,
        description: taskData.description,
        status: taskData.status,
        priority: taskData.priority,
        assignedTo: taskData.assignedTo
        });
        toast.success("New task created!");
      }
      setIsModalOpen(false);
      setEditingTask(null);
    } catch (error) {
      console.error("Error saving task:", error);
      toast.error("Something went wrong. Please try again.");
    }
  };

  const handleDeleteTask = async (id: number) => {
    if (!confirm('Are you sure you want to delete this tas
k?')) return;

    try {
      await axios.delete(`http://localhost:3003/tasks/${id}
`);
      toast.success("Task deleted successfully");
    } catch (error) {
        console.error("Failed to delete task:", error);
        toast.error("Could not delete task. Please try agai
n.");
      }
  };
```

Target 2: flash of empty content.

we are going to add Skeleton Loaders. these are those gray, pulsing bars you see on facebook or youTube while data is loading

Step 1: Pass loading to the columns props

update `src/App.tsx`

```
<KanbanColumn
  ...
  isLoading={isLoading}
/>
```

Step 2: build the skeleton in `KanbanColumn.tsx`

1. Add to Interface and receive it

```
interface KanbanColumnProps {
  // ... existing props ...
  isLoading?: boolean;
}

export default function KanbanColumn({
  id, title, tasks, count, onAdd, onDelete, onEdit, isLoad
ing
}: KanbanColumnProps) {
```

2. add the skeleton logic

```
      <Droppable droppableId={id}>
        {(provided) => (
          <div
            {...provided.droppableProps}
            ref={provided.innerRef}
            className="flex-1 overflow-y-auto min-h-0 p-3
flex flex-col gap-3"
          >
```

```
            {isLoading ? (
              <>
                {[1, 2, 3].map((i) => (
                  <div key={i} className="bg-white p-4 rou
nded-xl border border-gray-100 shadow-sm animate-pulse">
                    <div className="h-4 bg-gray-200 rounde
d w-3/4 mb-3"></div>
                    <div className="h-3 bg-gray-200 rounde
d w-1/2 mb-2"></div>
                    <div className="flex justify-between i
tems-center mt-4">
                      <div className="h-5 w-12 bg-gray-200
rounded"></div>
                      <div className="h-6 w-6 rounded-full
bg-gray-200"></div>
                    </div>
                  </div>
                ))}
              </>
            ) : (
              tasks.map((task, index) => (
                <TaskCard
                  key={task.id}
                  id={task.id}
                  index={index}
                  title={task.title}
                  priority={task.priority}
                  assignedTo={task.assignedTo || task.assi
gned_to}
                  onDelete={onDelete}
                  onClick={() => onEdit(task)}
                />
              ))
            )}
            {provided.placeholder}
            <button
              onClick={onAdd}
              className="w-full py-2 border border-gray-30
0 rounded-lg text-gray-500 text-sm font-medium hover:bg-wh
ite hover:border-gray-400 transition-all flex items-center
justify-center gap-2 bg-transparent group mt-3"
            >
              <Plus size={16} className="text-gray-400 gro
up-hover:text-gray-600" />
              New
            </button>
          </div>
        )}
```

NOTE:

right now our column grows as tall as the tasks inside it which pushes the "New" button off the screen and forces the *whole page* to scroll. we need to tell the column: *"you are only allowed to be X pixels tall. if your tasks don't fit, show a scrollbar inside the list, but keep the 'New' button pinned to the bottom."*

to fix this we need to update `src/components/KanbanColumn.tsx`

```
// ... imports

export default function KanbanColumn({ ...props }: KanbanC
olumnProps) {
  return (
        <div className="flex-1  rounded-xl p-4 min-w-[280p
x] shrink-0 bg-[#F7F8FA] rounded-xl border border-gray-100
flex flex-col max-h-[calc(100vh-12rem)]">

            ...

      <Droppable droppableId={props.id}>
        {(provided) => (
          <div
            {...provided.droppableProps}
            ref={provided.innerRef}
            className="flex-1 overflow-y-auto min-h-0 p-3
flex flex-col gap-3"
          >
            {props.tasks.map((task, index) => (
              ...

  );
}
```