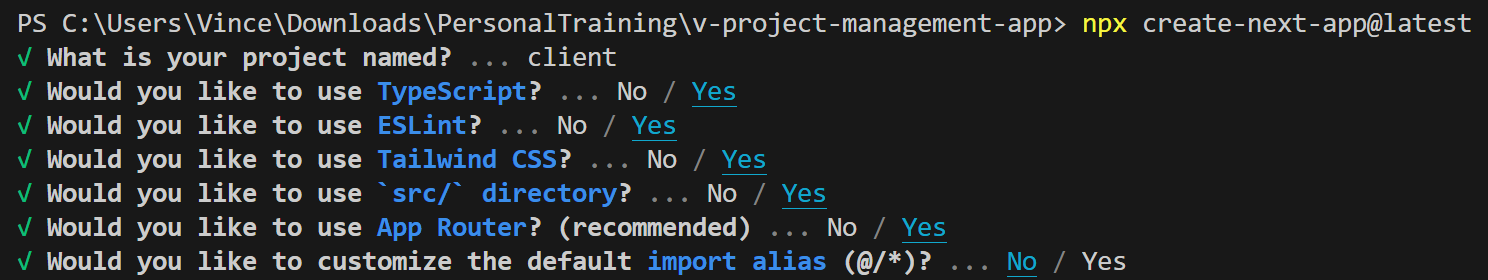
**Project Management App**

**Setup:**

* Create a client folder
  + npx create-next-app@latest



* VSCode Extension
  + ES7+ React/Redux/React-Native snippets
  + Prettier - Code formatter
* Frontend Packages Installation
  + (inside **client** folder)
  + (Terminal) npm install @mui/material @emotion/react @emotion/styled lucide-react numeral date-fns axios recharts react-dnd react-dnd-html5-backend gantt-task-react @mui/x-data-grid
  + Material UI (MUI) <https://mui.com/material-ui/getting-started/installation/>
  + Lucide React (Icons) <https://lucide.dev/guide/packages/lucide-react>
  + Numeral <https://www.npmjs.com/package/numeral>
  + date-fns <https://www.npmjs.com/package/date-fns>
  + axios <https://axios-http.com/docs/intro>
  + recharts <https://recharts.org/en-US/guide>
  + React Drag and Drop (DnD)
    - <https://www.npmjs.com/package/react-dnd>
    - <https://www.npmjs.com/package/react-dnd-html5-backend>
  + Interactive Gantt Chart for React with TypeScript <https://www.npmjs.com/package/gantt-task-react>
  + MUI Data Grid <https://mui.com/x/react-data-grid/getting-started/#installation>
* Dependencies Installation for TypeScript
  + (inside **client** folder)
  + (Terminal) npm i -D @types/node @types/uuid @types/numeral
* global.css modification

A screen shot of a computer program

Description automatically generated

* Dependencies Installation for Tailwind CSS (i.e., Tailwind x Prettier)
  + (inside **client** folder)
  + (Terminal) npm install -D prettier prettier-plugin-tailwindcss

A screen shot of a computer

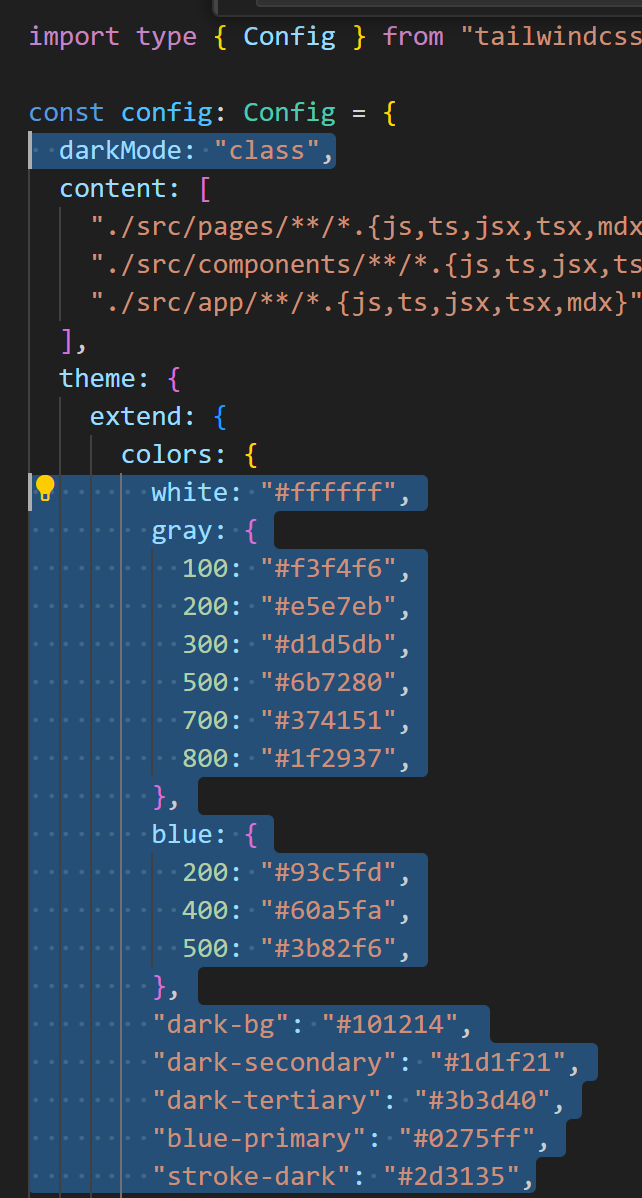
Description automatically generated

* + <https://tailwindcss.com/blog/automatic-class-sorting-with-prettier>
  + <https://github.com/tailwindlabs/prettier-plugin-tailwindcss>

A screenshot of a computer

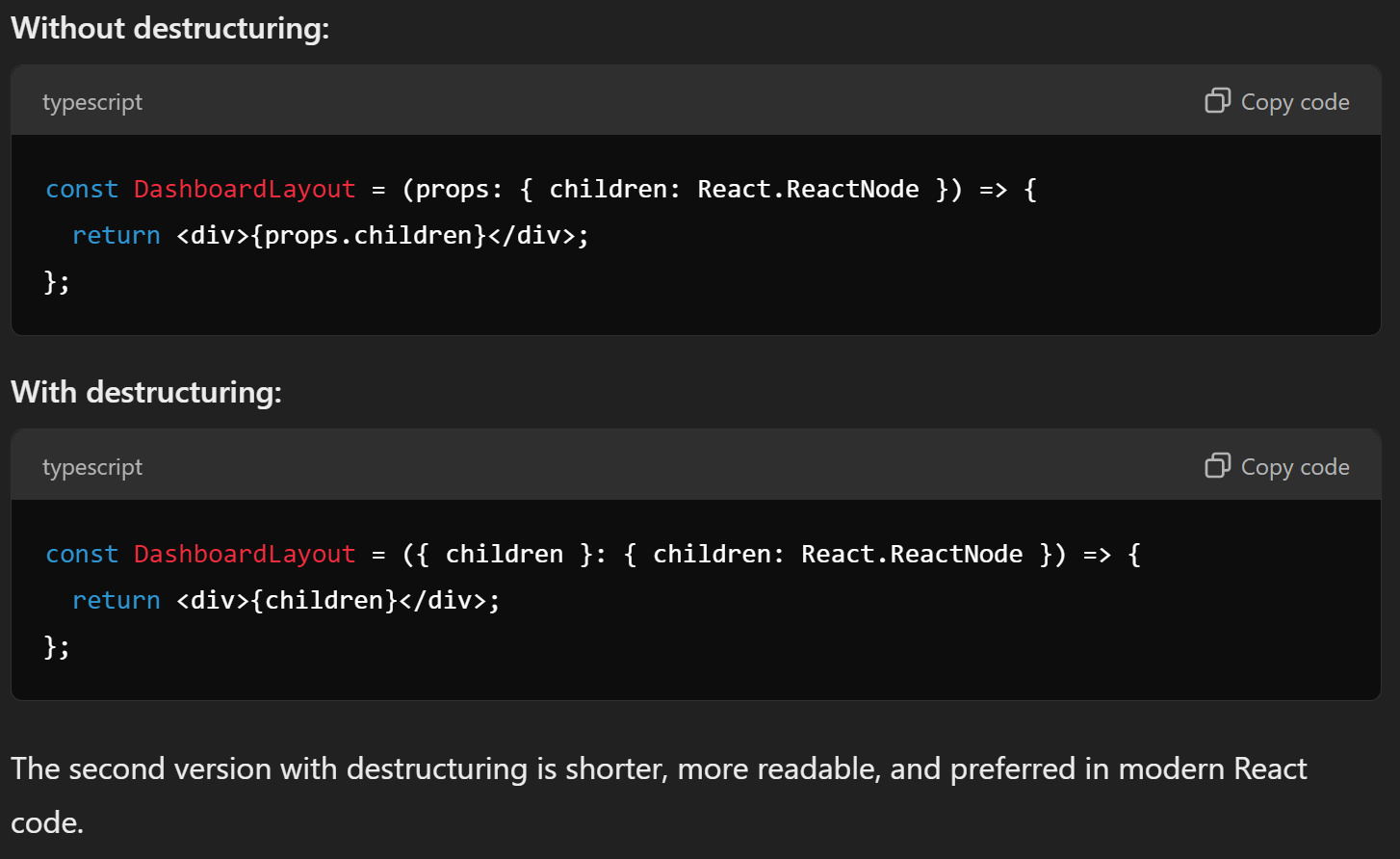
Description automatically generated

* Modify tailwind.config.ts



**dashboardWrapper.tsx**

* Create dashboardWrapper.tsx
  + Use tsrafce
  + Add parameter { children }: {children: React.ReactNode}
  + <https://dev.to/elhamnajeebullah/react-typescript-what-is-reactnode-and-when-to-use-it-3660>



* Modify layout.tsx to include our custom dashboard wrapper

A computer screen with text

Description automatically generated

* Create a Navbar components (client/src/components/Header/index.tsx)

A screenshot of a computer program

Description automatically generated

* Import to dashboardWrapper.tsx

A screen shot of a computer program

Description automatically generated

* What is @?
  + Go to tsconfig.json

A close up of a computer screen

Description automatically generated

* How do you know the import works?
  + **Ctrl + Click on the path**
* Change tab icon and name
  + Change favicon.ico file (src/app/favicon.ico) with your logo
  + Modify name in the layout.tsx

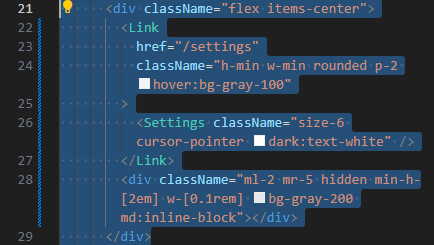
**NavBar**

* Create src/components/Header/index.tsx
* tsrafce
* Create the search bar
  + Import { Search } from “lucide-react”;
  + Create the search bar UI

A screenshot of a computer program

Description automatically generated

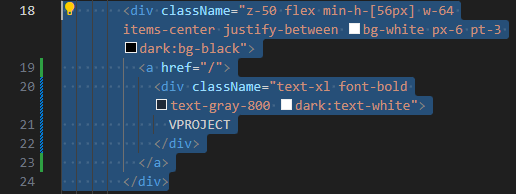
* + Create a settings icon
    - import { Settings } from "lucide-react";
    - import Link from "next/link";



****

**SideBar**

* Create src/components/SideBar/index.tsx
* tsrafce
* Create Top Logo

****

**A black and white logo

Description automatically generated**

* Create Team section

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**Redux Toolkit (State Management) Installation**

The Redux Toolkit package is intended to be the standard way to write Redux logic.

* <https://redux-toolkit.js.org/usage/nextjs>
* Installation (Inside **client** folder)
* (Terminal) npm i react-redux @reduxjs/toolkit redux-persist dotenv

**react-redux**:

* This is the official **Redux binding** for React. It allows React components to interact with a Redux store and simplifies connecting your components to the store by providing useSelector and useDispatch hooks, among other utilities.
* React-Redux ensures that your React app automatically re-renders in response to state changes.
* <https://redux-toolkit.js.org/usage/nextjs>

**@reduxjs/toolkit**:

* This is the official, recommended way to write **Redux logic**. It includes a set of tools and utilities that simplify common Redux tasks, like creating reducers and actions, configuring the store, and handling middleware.
* It comes with functions like createSlice, configureStore, and createAsyncThunk to streamline development and reduce boilerplate code.
* <https://redux-toolkit.js.org/usage/nextjs>

**redux-persist**:

* This is a library that helps **persist and rehydrate** the Redux store. It allows the Redux state to be saved in local storage (or other storage options) so that the state is not lost when the page is refreshed.
* Redux Persist takes the information in the Redux state (using JSON. stringify) and saves it to a specified storage engine (e.g., localStorage or AsyncStorage for React Native). This saved information stays even if you close the app or browser.
* It is often used in applications that need to maintain user session data or settings across page reloads.
* **For example (In this case)**, the users set the website into dark mode, it carries on to the next session (even though they close the browser or tab, it will still be dark mode).
* <https://www.npmjs.com/package/redux-persist>

**dotenv**:

* This is a zero-dependency module that loads **environment variables** from a .env file into process.env. It is commonly used to manage sensitive or configurable values like API keys, database connection strings, or feature flags, allowing you to keep these values separate from your codebase.
* This helps to keep your sensitive data secure and also provides a clean way to manage different environments (e.g., development, production).
* **dotenv** is a **Node.js library** that allows you to load variables from a **.env** file into your Node.js application's **process.env.**
* (However, most of projects-creation scripts handle environment variables reading for you out of the box (i.e., use dotenv under the hood). <https://stackoverflow.com/questions/76078204/why-do-i-have-to-download-dotenv-package-if-i-can-use-env-file-without-it>

**Check if a package is installed?**

* Open package.json and check dependencies
* (Terminal) npm ls dotenv

**Redux Toolkit Setup**

* <https://redux-toolkit.js.org/usage/nextjs>
* Create src/app/redux.tsx
* Copy and paste the template **with redux-persist** (i.e., the code contains inside it was generated from the link)

**globalReducer**

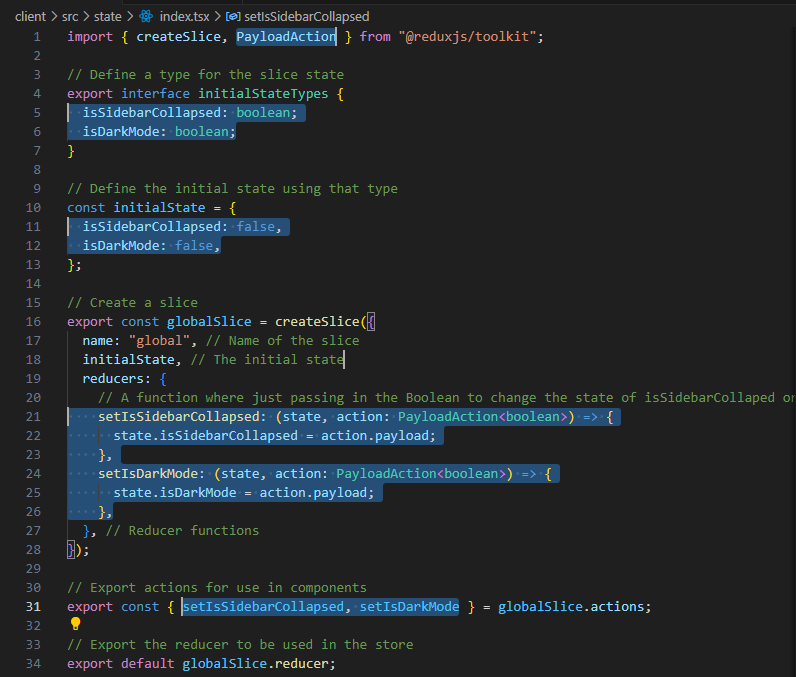
* Create our first logic for import globalReducer from "@/state";
* Create src/state/index.ts
* Create a Redux Slice. (Below is the Redux Slice skeleton for TypeScript)

**A screen shot of a computer program

Description automatically generated**

The **initialStateTypes** and **globalSlice** interface is exported, making it accessible to any other module that needs to refer to the structure of the global slice's state.

* After adding **isSidebarCollapsed** and **isDarkMode**



**api state**

* Create our second logic for import { api } from "@/state/api";
* Create src/state/api.ts
* Create a .env.local file

****

* Create a Redux Api. (Below is the Redux Api skeleton for TypeScript)

**A screen shot of a computer code

Description automatically generated**

**Integrate Redux to React**

* Modify dashboardWrapper.tsx (StoreProvider is a function from redux.tsx)

A screen shot of a computer

Description automatically generated

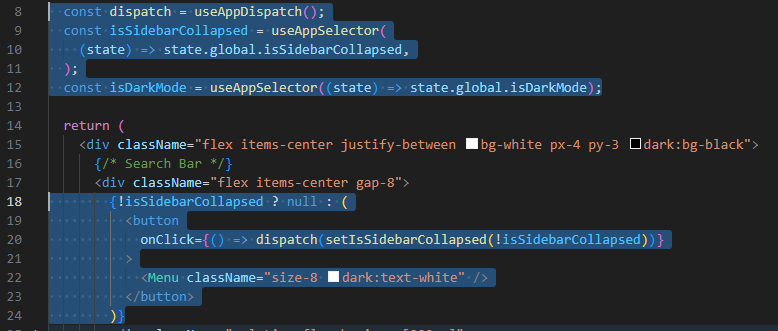
**DarkMode**

* Modify dashboardWrapper.tsx

A computer screen shot of a program code

Description automatically generated

* Modify NavBar/index.tsx



A screenshot of a computer program

Description automatically generated

**SideBar**

* Modify SideBar/index.tsx
  + Create SidebarLinkProps interface and SidebarLink function

A screenshot of a computer program

Description automatically generated

* + Create close Sidebar logic

A screen shot of a computer program

Description automatically generated

* + Create NavBar Links, Projects Links, and Priorities Links

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

**Data Model and Database Setup**

A screenshot of a computer

Description automatically generated

* Install PostgreSQL (The first link provides both PostgreSQL server and pgAdmin 4)
  + <https://www.postgresql.org/download/>
  + <https://www.youtube.com/watch?v=4qH-7w5LZsA>
* Install pgAdmin
  + <https://www.pgadmin.org/download/>
  + pgAdmin is the most popular and feature rich Open-Source administration and development platform for PostgreSQL.
* Create a server for the project (After opening pgAdmin 4 platform)

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Password**: ToxicDev5

* Create a database

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

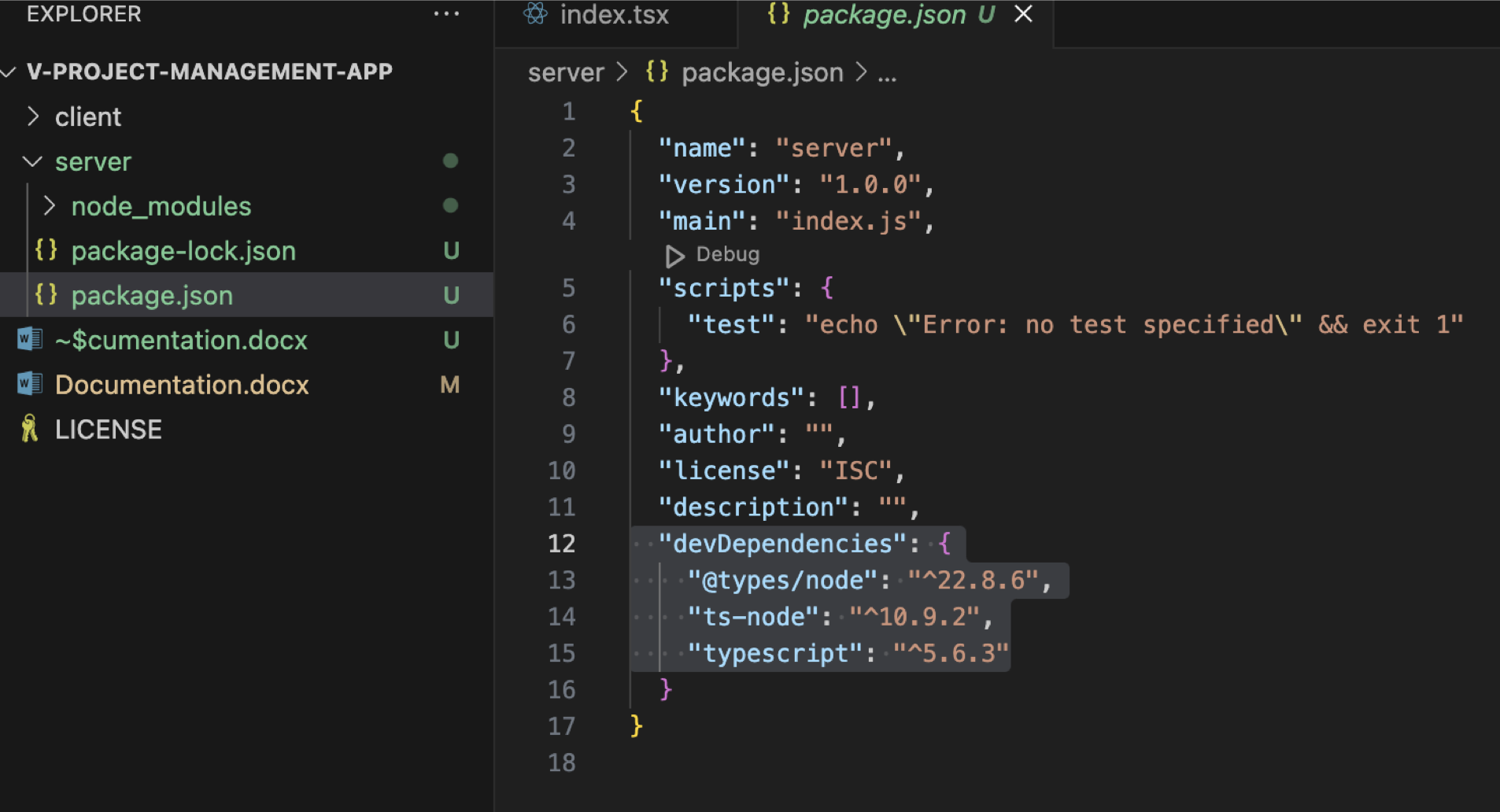
**Server Setup (Typescript Install)**

* Create a **server** folder.
* (Terminal) npm init -y
* <https://stackoverflow.com/questions/62725481/what-is-the-meaning-of-npm-init-y>
* The terminal command will generate a package.json file.

A screen shot of a computer program

Description automatically generated

* Add dependencies.
* (Terminal) npm i -D ts-node typescript @types/node



The command npm i -D ts-node typescript @types/node is used to install three packages in a Node.js project. Here's what each part does:

1. **npm i -D**: This command installs the packages as **development dependencies**. The -D (or --save-dev) flag tells npm to add these packages to the devDependencies section of your package.json. Development dependencies are only needed during development and not in production, so they won’t be included in the final bundle.
2. **ts-node**: This package allows you to run TypeScript code directly in Node.js without needing to compile it to JavaScript first. It's commonly used in development to make it easier to execute TypeScript scripts and to run TypeScript-based CLI tools.
3. **typescript**: This is the TypeScript compiler, which is necessary for compiling .ts (TypeScript) files into .js (JavaScript) files. Installing this package allows you to use TypeScript syntax in your project and to compile TypeScript code into JavaScript for production.
4. **@types/node**: This package contains TypeScript type definitions for Node.js. It provides type information for Node.js built-in modules (like fs, http, path, etc.), so that TypeScript can understand and check the Node.js API usage in your code.

* Generate tsconfig.json
* (Terminal) npx tsc –init

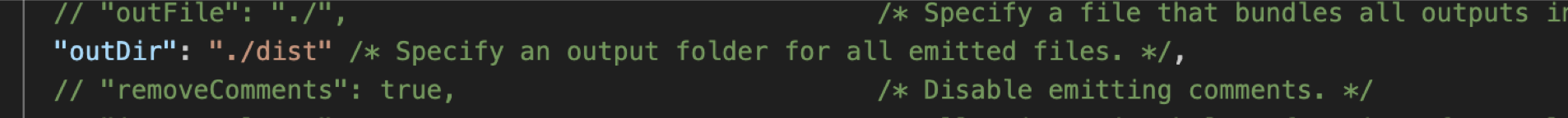
A screenshot of a computer

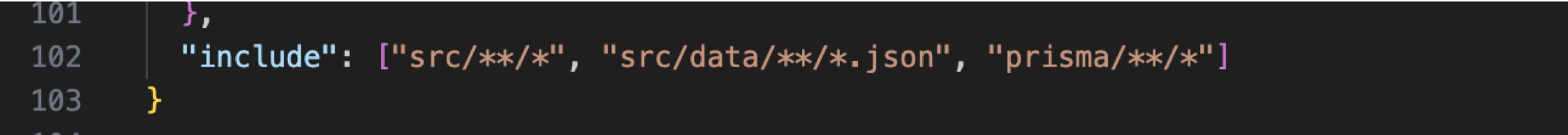
Description automatically generated

* Modify tsconfig.json

A screen shot of a computer program

Description automatically generated





**Prisma Setup**

* (Terminal) npm i prisma @prisma/client
* (Terminal) npx prisma init

A screenshot of a computer

Description automatically generated

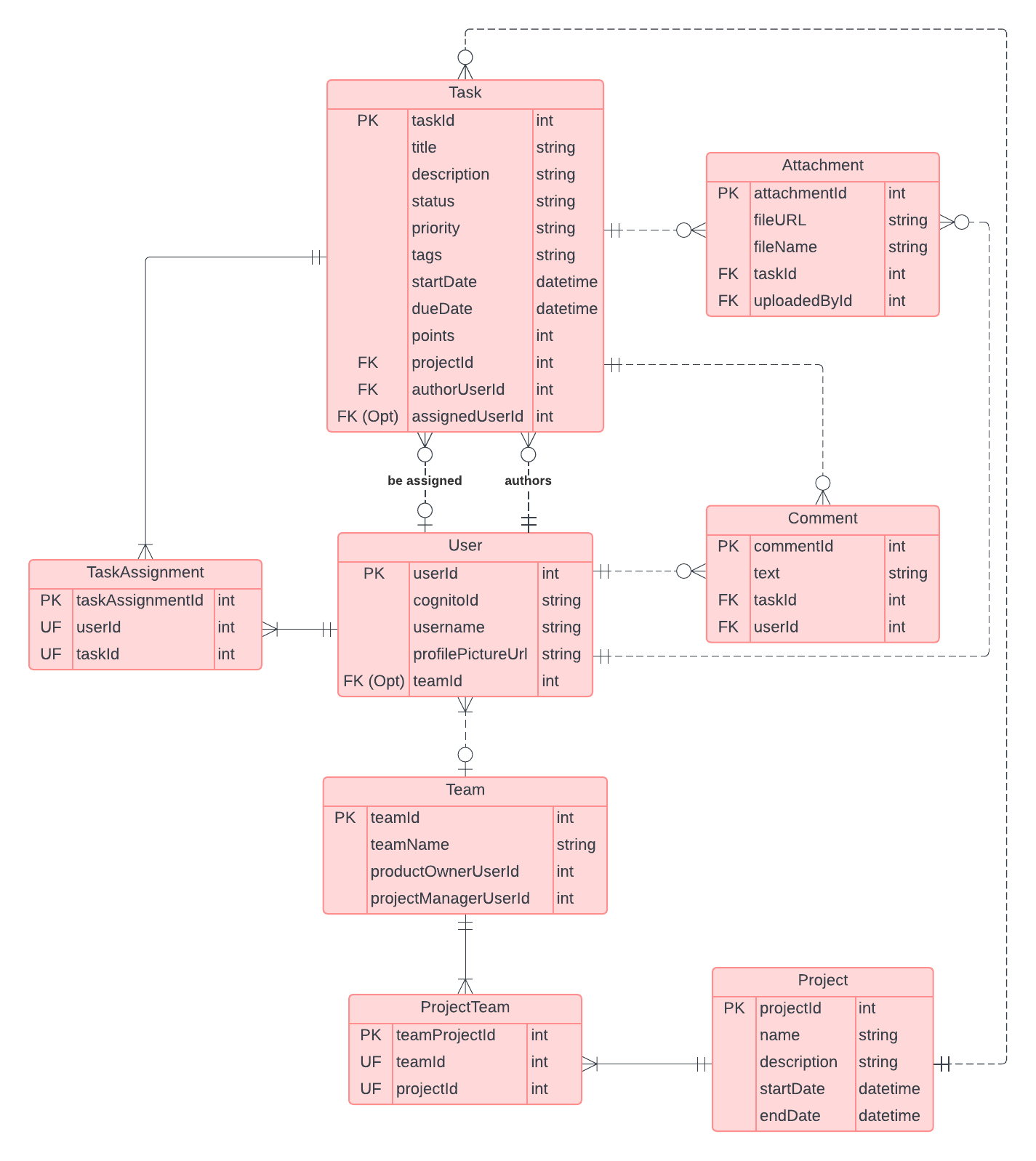
* Create seed.ts file.
* Download prisma extension (to automatically format schema.prisma file).

A screenshot of a phone

Description automatically generated

* Write the schema codes

**Fixed ERD**



* Modify server/package.json

A screen shot of a computer

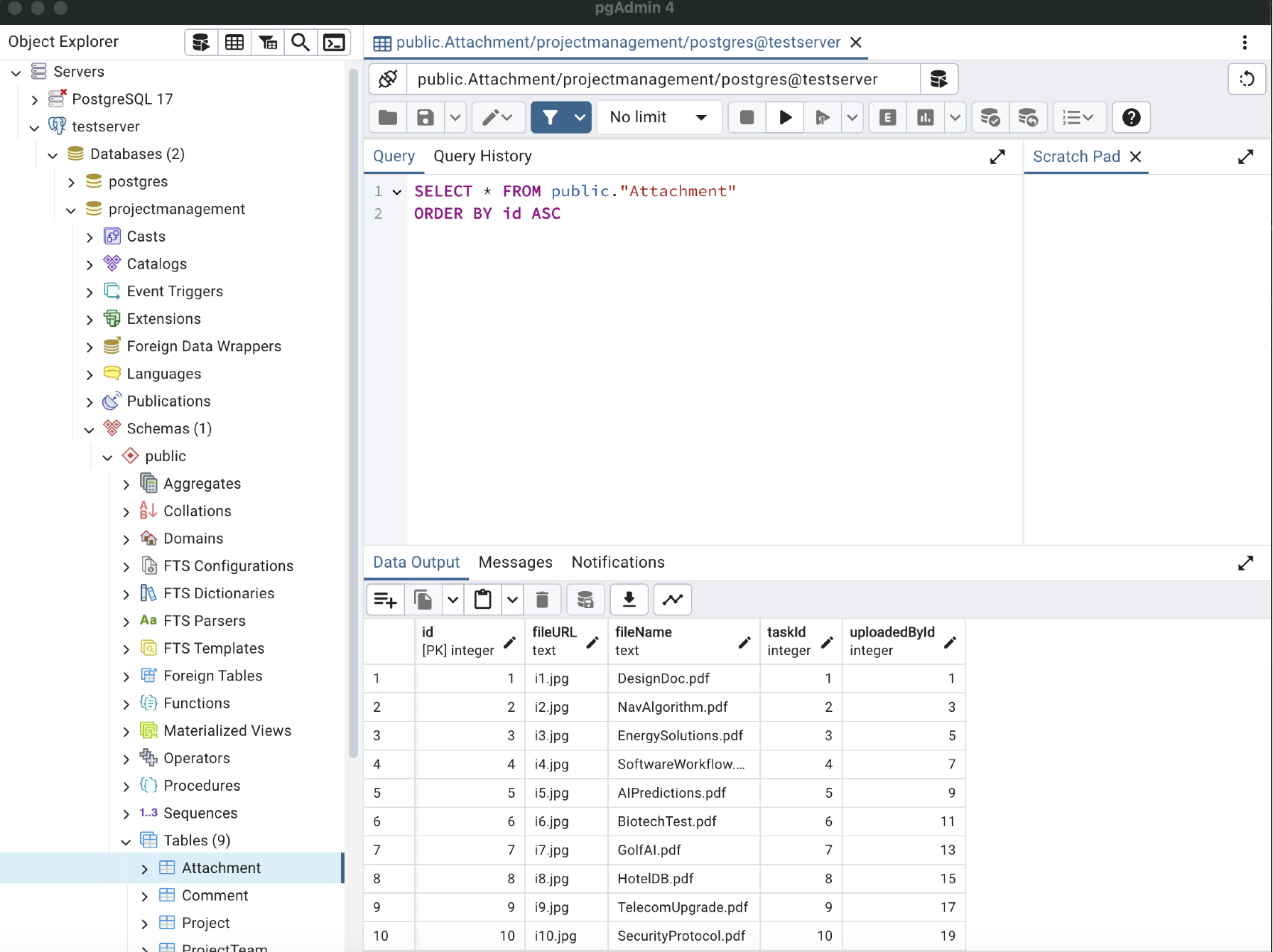
Description automatically generated

* Modify server/.env

A screen shot of a computer

Description automatically generated

* (Terminal) npx prisma generate
* (Terminal) npx prisma migrate dev --name init
* (Terminal) npm run seed
* (Terminal – (if we setup the schema incorrectly) npx prisma migrate reset



**Backend Install (Packages)**

* (Terminal) (Packages) npm i express body-parser cors dotenv helmet morgan

The command installs a set of popular Node.js packages commonly used in Express.js applications for various functionalities. Here’s a breakdown of each package and its purpose:

**1. express**

* **Purpose**: Express is a minimal and flexible web application framework for Node.js that provides robust features for building web and mobile applications.
* **Use**: It simplifies the process of routing, handling requests, and managing middleware, making it much easier to build APIs and web applications on top of Node.js.

**2. body-parser**

* **Purpose**: Body-parser is a middleware for parsing incoming request bodies in a middleware before your handlers, making the req.body property available.
* **Use**: Commonly used for parsing JSON and URL-encoded form data sent by clients in HTTP POST requests. However, Express has built-in body parsing since version 4.16.0, so this package may not be necessary if using newer versions.

**3. cors**

* **Purpose**: CORS (Cross-Origin Resource Sharing) is a middleware that enables controlled access to resources from different origins (domains) than the server’s origin. This is crucial for security in applications that interact with resources on different domains.
* **Use**: By default, browsers block requests that violate the Same-Origin Policy. This package allows you to specify which domains can access your API.

**4. dotenv**

* **Purpose**: Dotenv is a module for loading environment variables from a .env file into process.env.
* **Use**: Useful for managing configuration variables (such as database connection strings, API keys, and other sensitive data) that should remain outside your codebase and vary between development, staging, and production environments.

**5. helmet**

* **Purpose**: Helmet is a security middleware for Express applications that helps protect your app from common web vulnerabilities.
* **Use**: It sets various HTTP headers to enhance security, including protections against XSS, clickjacking, and MIME-sniffing attacks, making it a go-to for web app security in Node.js.

**6. morgan**

* **Purpose**: Morgan is an HTTP request logger middleware for Node.js, useful for logging requests and responses to the console or a file.
* **Use**: Helps with monitoring incoming requests, logging details like request method, status codes, response time, etc. This is invaluable for debugging and analyzing HTTP traffic.
* (Terminal) (Dependencies) npm i -D rimraf concurrently nodemon @types/cors @types/express @types/morgan @types/node

The command installs several packages as **development dependencies** (indicated by the -D flag). These packages are typically used in development but are not necessary for running the app in production. Here’s a breakdown of each package:

**1. rimraf**

* **Purpose**: rimraf is a cross-platform command-line tool for deleting files and directories, similar to the Unix command rm -rf.
* **Use**: Often used to clean up build artifacts, cache, or temporary files in development environments. It’s commonly configured in scripts (like npm run clean) to remove dist or build folders.

**2. concurrently**

* **Purpose**: concurrently allows you to run multiple commands or scripts in parallel from a single npm script.
* **Use**: Useful in development when you need to run multiple tasks simultaneously, such as starting both a frontend and backend server, or watching files for changes while running a build script. This is particularly handy for full-stack projects with separate client and server setups.

**3. nodemon**

* **Purpose**: nodemon is a utility that automatically restarts the Node.js application when file changes in the directory are detected.
* **Use**: Great for development, as it allows you to see code changes reflected immediately without needing to manually stop and restart the server each time a file is updated.

**Type Definitions (for TypeScript Support)**

The @types/ packages below provide TypeScript type definitions for the corresponding libraries, allowing TypeScript users to use these packages with full type support. This helps the TypeScript compiler understand the shape of the modules’ data and methods, enabling IDEs to provide features like autocompletion and error checking.

**4. @types/cors**

* **Purpose**: Adds type definitions for the cors middleware package.
* **Use**: Provides type information for configuration options and usage of cors in TypeScript applications.

**5. @types/express**

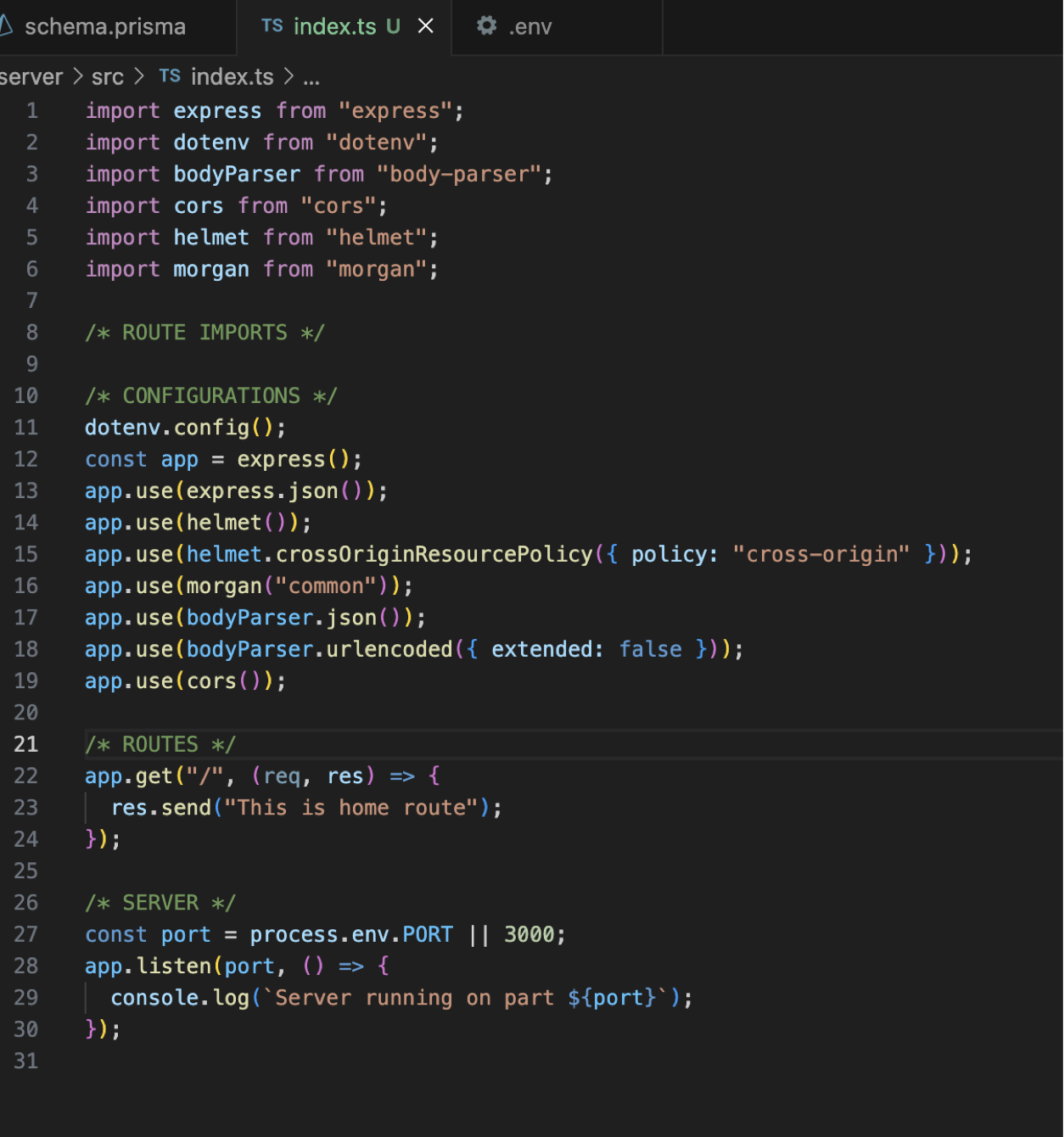
* **Purpose**: Adds type definitions for the express framework.
* **Use**: Essential for TypeScript projects that use Express, as it defines types for Express functions, middleware, requests, and responses.

**6. @types/morgan**

* **Purpose**: Adds type definitions for the morgan HTTP request logger middleware.
* **Use**: Defines the type signatures for morgan configuration options and setup, useful when using it in a TypeScript-based Express application.

**7. @types/node**

* **Purpose**: Adds type definitions for core Node.js modules and global variables.
* **Use**: This package is essential for any TypeScript project running on Node.js, as it provides type information for core modules like fs, path, http, and process.
* Create server/src/index.ts



* Modify server/.env

A screenshot of a computer

Description automatically generated

* Modify server/package.json

A screen shot of a computer

Description automatically generated

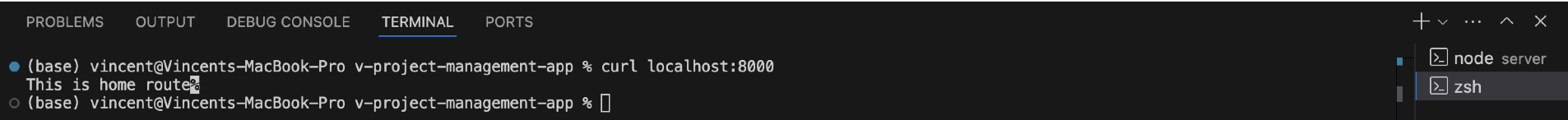
**Test the server**

* (Terminal) npm run dev

A screen shot of a computer program

Description automatically generated

* (Open a new terminal) curl localhost:8000



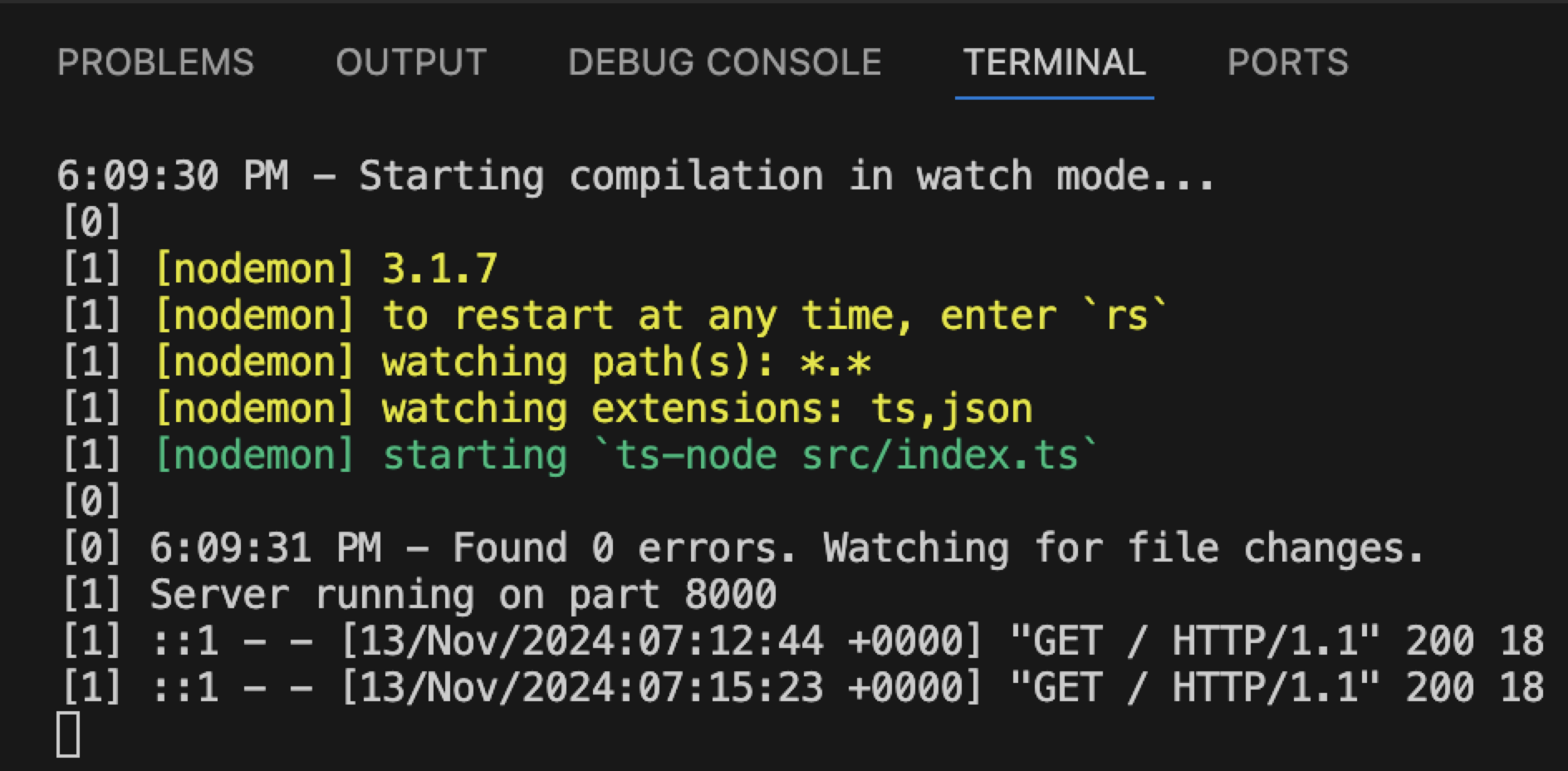
A screen shot of a computer program

Description automatically generated

* (Postman) localhost:8000

A screenshot of a computer

Description automatically generated



**Projects Section Backend**