**Frontend Architecture Overview**

Your project is organized in a modular way, which is a best practice for building scalable React applications. Here's what each file does:

**1. main.jsx**

* **Purpose:**
  + This is the entry point of your React application.
  + It uses createRoot from ReactDOM to render your <App /> component inside the element with the id "root".
* **Key Points:**
  + Wrapped in <StrictMode> for highlighting potential issues during development.
  + Imports global styles from ./index.css.

**import** **{** StrictMode **}** from 'react'**;**

**import** **{** createRoot **}** from 'react-dom/client'**;**

**import** './index.css'**;**

**import** App from './App.jsx'**;**

createRoot**(document.**getElementById**(**'root'**)).**render**(**

**<**StrictMode**>**

**<**App **/>**

**</**StrictMode**>**

**);**

**2. App.jsx**

* **Purpose:**
  + Serves as the main component that sets up the React Query provider, React Router, and defines your app routes.
* **Key Points:**
  + Creates a QueryClient for React Query to manage server state (like todos).
  + Uses React Router’s <Routes> and <Route> to map the path / to the <Home /> component.
* **Code:**

**import** "./App.css"**;**

**import** React**,** **{** StrictMode **}** from "react"**;**

**import** **{** QueryClient**,** QueryClientProvider **}** from "@tanstack/react-query"**;**

**import** **{** BrowserRouter as Router**,** Routes**,** Route **}** from "react-router-dom"**;**

**import** Home from "./pages/Home"**;**

**const** queryClient **=** **new** QueryClient**();**

**function** App**()** **{**

**return** **(**

**<**QueryClientProvider client**={**queryClient**}>**

**<**Router**>**

**<**Routes**>**

**<**Route path**=**"/" **element={<**Home **/>}** **/>**

**</**Routes**>**

**</**Router**>**

**</**QueryClientProvider**>**

**);**

**}**

**export** **default** App**;**

**3. ./services/api.jsx**

* **Purpose:**
  + Contains functions for making API calls using Axios.
  + Each function corresponds to a REST endpoint for getting, adding, updating, or deleting todos.
* **Key Points:**
  + API\_BASE\_URL is defined for ease of use.
  + Uses Axios to simplify HTTP requests and error handling.
* **Code:**

**import** axios from "axios"**;**

**const** API\_BASE\_URL **=** "http://localhost:5000/v1/todos"**;**

**export** **const** getTodos **=** **async** **()** **=>** **{**

**const** response **=** **await** axios**.**get**(**API\_BASE\_URL**);**

**return** response**.**data**;**

**};**

**export** **const** addTodo **=** **async** **(**description**)** **=>** **{**

**const** response **=** **await** axios**.**post**(**API\_BASE\_URL**,** **{** description **});**

**return** response**.**data**;**

**};**

**export** **const** updateTodo **=** **async** **(**id**,** description**)** **=>** **{**

**const** response **=** **await** axios**.**put**(**`**${**API\_BASE\_URL**}**/**${**id**}**`**,** **{** description **});**

**return** response**.**data**;**

**};**

**export** **const** deleteTodo **=** **async** **(**id**)** **=>** **{**

**return** axios**.delete(**`**${**API\_BASE\_URL**}**/**${**id**}**`**);**

**};**

**4. ./pages/Home.jsx**

* **Purpose:**
  + Acts as the landing page of your application.
  + It imports and displays the input component (for adding a new todo) and the list component (for displaying all todos).
* **Key Points:**
  + Wrapped in a container for layout consistency.
* **Code:**

**import** React from "react"**;**

**import** InputTodo from "../components/InputTodo"**;**

**import** ListTodos from "../components/ListTodos"**;**

**const** Home **=** **()** **=>** **{**

**return** **(**

**<**div className**=**"container"**>**

**<**InputTodo **/>**

**<**ListTodos **/>**

**</**div**>**

**);**

**};**

**export** **default** Home**;**

**5. ./hooks/useTodos.jsx**

* **Purpose:**
  + A custom hook that encapsulates all logic related to fetching and mutating todos using React Query.
* **Key Points:**
  + Uses useQuery to fetch todos.
  + Uses useMutation for add, update, and delete operations, with automatic query invalidation for refreshing the todos list.
* **Code:**

**import** **{** useQuery**,** useMutation**,** useQueryClient **}** from "@tanstack/react-query"**;**

**import** **{** getTodos**,** addTodo**,** updateTodo**,** deleteTodo **}** from "../services/api"**;**

**export** **const** useTodos **=** **()** **=>** **{**

**const** queryClient **=** useQueryClient**();**

**const** **{** data**:** todos**,** isLoading**,** error **}** **=** useQuery**({**

queryKey**:** **[**"todos"**],**

queryFn**:** getTodos**,**

**});**

**const** addTodoMutation **=** useMutation**({**

mutationFn**:** addTodo**,**

onSuccess**:** **()** **=>** queryClient**.**invalidateQueries**([**"todos"**]),**

**});**

**const** updateTodoMutation **=** useMutation**({**

mutationFn**:** **({** id**,** description **})** **=>** updateTodo**(**id**,** description**),**

onSuccess**:** **()** **=>** queryClient**.**invalidateQueries**([**"todos"**]),**

**});**

**const** deleteTodoMutation **=** useMutation**({**

mutationFn**:** deleteTodo**,**

onSuccess**:** **()** **=>** queryClient**.**invalidateQueries**([**"todos"**]),**

**});**

**return** **{**

todos**,**

isLoading**,**

error**,**

addTodoMutation**,**

updateTodoMutation**,**

deleteTodoMutation**,**

**};**

**};**

**6. ./components/EditTodo.jsx**

* **Purpose:**
  + A component for editing an existing todo.
  + It uses React state to control the modal display and form input.
* **Key Points:**
  + Uses React Query’s updateTodoMutation to send the update.
  + Uses local state (isOpen) to control modal visibility without relying on Bootstrap’s JavaScript.
* **Code:**

**import** React**,** **{** useState **}** from "react"**;**

**import** **{** useTodos **}** from "../hooks/useTodos"**;**

**const** EditTodo **=** **({** todo **})** **=>** **{**

**const** **[**description**,** setDescription**]** **=** useState**(**todo**.**description**);**

**const** **[**isOpen**,** setIsOpen**]** **=** useState**(false);** // React state for modal

**const** **{** updateTodoMutation **}** **=** useTodos**();**

**const** updateDescription **=** **async** **(**e**)** **=>** **{**

e**.**preventDefault**();**

**try** **{**

**await** updateTodoMutation**.**mutateAsync**({** id**:** todo**.**todo\_id**,** description **});**

setIsOpen**(false);** // Close the modal after update

**}** **catch** **(**err**)** **{**

console**.**error**(**"Update failed:"**,** err**.**message**);**

**}**

**};**

**return** **(**

**<>**

**{**/\* Open modal button \*/**}**

**<button** className**=**"btn btn-warning" onClick**={()** **=>** setIsOpen**(true)}>**

Edit

**</button>**

**{**/\* Edit Todo Modal \*/**}**

**{**isOpen **&&** **(**

**<**div className**=**"modal show d-block" tabIndex**=**"-1"**>**

**<**div className**=**"modal-dialog"**>**

**<**div className**=**"modal-content"**>**

**<**div className**=**"modal-header"**>**

**<**h4 className**=**"modal-title"**>**Edit Todo**</**h4**>**

**<button** type**=**"button" className**=**"close" onClick**={()** **=>** setIsOpen**(false)}>**

**&**times**;**

**</button>**

**</**div**>**

**<**div className**=**"modal-body"**>**

**<**input

type**=**"text"

className**=**"form-control"

value**={**description**}**

onChange**={(**e**)** **=>** setDescription**(**e**.**target**.**value**)}**

**/>**

**</**div**>**

**<**div className**=**"modal-footer"**>**

**<button** className**=**"btn btn-warning" onClick**={**updateDescription**}>**

Save

**</button>**

**<button** className**=**"btn btn-danger" onClick**={()** **=>** setIsOpen**(false)}>**

Close

**</button>**

**</**div**>**

**</**div**>**

**</**div**>**

**</**div**>**

**)}**

**</>**

**);**

**};**

**export** **default** EditTodo**;**

**7. ./components/InputTodo.jsx**

* **Purpose:**
  + Provides an input form for adding new todos.
* **Key Points:**
  + Uses React Query’s addTodoMutation to send the new todo.
  + Resets the input after a successful addition.
* **Code:**

**import** React**,** **{** useState **}** from "react"**;**

**import** **{** useTodos **}** from "../hooks/useTodos"**;**

**const** InputTodo **=** **()** **=>** **{**

**const** **[**description**,** setDescription**]** **=** useState**(**""**);**

**const** **{** addTodoMutation **}** **=** useTodos**();**

**const** onSubmitForm **=** **async** **(**e**)** **=>** **{**

e**.**preventDefault**();**

addTodoMutation**.**mutate**(**description**,** **{**

onSuccess**:** **()** **=>** setDescription**(**""**),**

**});**

**};**

**return** **(**

**<**div**>**

**<**h1 className**=**"text-center mt-5"**>**PERN Todo List**</**h1**>**

**<form** className**=**"d-flex mt-5" onSubmit**={**onSubmitForm**}>**

**<**input

type**=**"text"

className**=**"form-control"

value**={**description**}**

onChange**={(**e**)** **=>** setDescription**(**e**.**target**.**value**)}**

**/>**

**<button** className**=**"btn btn-success"**>**Add**</button>**

**</form>**

**</**div**>**

**);**

**};**

**export** **default** InputTodo**;**

**8. ./components/ListTodos.jsx**

* **Purpose:**
  + Displays a table of all todos.
  + Provides options to edit or delete each todo.
* **Key Points:**
  + Uses React Query’s deleteTodoMutation for deletion.
  + Maps over todos and renders each row with corresponding edit and delete buttons.
* **Code:**

**import** React from "react"**;**

**import** **{** useTodos **}** from "../hooks/useTodos"**;**

**import** EditTodo from "./EditTodo"**;**

**const** ListTodos **=** **()** **=>** **{**

**const** **{** todos**,** isLoading**,** error**,** deleteTodoMutation **}** **=** useTodos**();**

**if** **(**isLoading**)** **return** **<**p**>**Loading**...</**p**>;**

**if** **(**error**)** **return** **<**p**>**Error loading todos**</**p**>;**

**return** **(**

**<**table className**=**"table mt-5 text-center"**>**

**<**thead**>**

**<**tr**>**

**<**th**>**Description**</**th**>**

**<**th**>**Edit**</**th**>**

**<**th**>**Delete**</**th**>**

**</**tr**>**

**</**thead**>**

**<**tbody**>**

**{**todos**?.**map**((**todo**)** **=>** **(**

**<**tr key**={**todo**.**todo\_id**}>**

**<**td**>{**todo**.**description**}</**td**>**

**<**td**>**

**<**EditTodo todo**={**todo**}** **/>**

**</**td**>**

**<**td**>**

**<button**

className**=**"btn btn-danger"

onClick**={()** **=>**

deleteTodoMutation**.**mutate**(**todo**.**todo\_id**,** **{**

onError**:** **(**err**)** **=>**

**alert(**"Error deleting todo: " **+** err**.**message**),**

**})**

**}**

**>**

Delete

**</button>**

**</**td**>**

**</**tr**>**

**))}**

**</**tbody**>**

**</**table**>**

**);**

**};**

**export** **default** ListTodos**;**

**Detailed Explanation**

* **Modular Structure:**  
  Each file has a single responsibility—services handle API calls, hooks manage data fetching with React Query, components manage UI and interactions, and pages combine these components to form views.
* **React Query Integration:**  
  Using React Query simplifies asynchronous data fetching and caching, making your UI automatically update after mutations (add, update, delete).
* **Routing:**  
  React Router is used in App.jsx to manage navigation. The <Home /> page is rendered at the root route.
* **State Management in Modals:**  
  Instead of relying on Bootstrap’s data attributes, the modal for editing todos is controlled using React state (isOpen). This provides more predictable behavior in a React environment.

**Backend Architecture Overview**

Your backend is built with **Node.js** using the **Express** framework. It connects to a **PostgreSQL** database through the pg library and uses environment variables to keep sensitive configuration separate. The project is organized into several folders and files:

**1. server.js**

* **Purpose:**  
  This is the main entry point for your backend application. It sets up the Express server, loads environment variables with dotenv, configures middleware (like CORS and JSON parsing), checks the database connection, mounts the API routes, and starts the server.
* **Key Points:**
  + **Environment Variables:**  
    Loaded using dotenv.config(), which reads settings from a .env file (such as database credentials and port).
  + **Middleware:**
    - cors(): Enables Cross-Origin Resource Sharing, allowing your frontend to access the API.
    - express.json(): Parses incoming JSON payloads.
  + **Database Connection Test:**  
    Runs a simple query (SELECT NOW()) to verify that the connection to PostgreSQL is working.
  + **Routing:**  
    The route /v1/todos is mounted to handle all todo-related operations.
  + **Error Handling:**  
    A catch-all 404 handler returns an error JSON if a route is not found.
  + **Server Listening:**  
    The server listens on the port defined in your environment variables (or 5000 by default).

**Excerpt from server.js:**

**import** express from "express"**;**

**import** cors from "cors"**;**

**import** dotenv from "dotenv"**;**

**import** pool from "./db/db.js"**;**

**import** todoRoutes from "./routes/todos.js"**;**

dotenv**.**config**();** // Load environment variables

**const** app **=** express**();**

**const** PORT **=** process**.**env**.**PORT **||** 5000**;**

app**.**use**(**cors**());**

app**.**use**(**express**.**json**());**

// Database Connection Test

**const** checkDatabaseConnection **=** **async** **()** **=>** **{**

**try** **{**

**const** res **=** **await** pool**.**query**(**"SELECT NOW()"**);**

console**.**log**(**"✅ Connected to PostgreSQL at:"**,** res**.**rows**[**0**].**now**);**

**}** **catch** **(**err**)** **{**

console**.**error**(**"❌ Database connection error:"**,** err**.**message**);**

process**.**exit**(**1**);**

**}**

**};**

checkDatabaseConnection**();**

// Mount routes

app**.**use**(**"/v1/todos"**,** todoRoutes**);**

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

res**.**send**(**"🚀 Welcome to the Express Server!"**);**

**});**

app**.**use**((**req**,** res**)** **=>** **{**

res**.status(**404**).**json**({** error**:** "Route not found" **});**

**});**

app**.**listen**(**PORT**,** **()** **=>** **{**

console**.**log**(**`🚀 Server running on http://localhost:**${**PORT**}**`**);**

**});**

;

**2. ./routes/todos.js**

* **Purpose:**  
  This file defines the API endpoints related to todos. It uses Express Router to map HTTP methods (GET, POST, PUT, DELETE) to controller functions.
* **Key Points:**
  + **Routes:**
    - POST / calls createTodo to add a new todo.
    - GET / calls getTodos to retrieve all todos.
    - GET /:id calls getTodoById to get a specific todo.
    - PUT /:id calls updateTodo to update a todo.
    - DELETE /:id calls deleteTodo to remove a todo.

**Excerpt from ./routes/todos.js:**

**import** express from "express"**;**

**import** **{**

createTodo**,**

getTodos**,**

getTodoById**,**

updateTodo**,**

deleteTodo**,**

**}** from "../controllers/todosController.js"**;**

**const** router **=** express**.**Router**();**

router**.**post**(**"/"**,** createTodo**);**

router**.**get**(**"/"**,** getTodos**);**

router**.**get**(**"/:id"**,** getTodoById**);**

router**.**put**(**"/:id"**,** updateTodo**);**

router**.delete(**"/:id"**,** deleteTodo**);**

**export** **default** router**;**

**3. ./db/db.js**

* **Purpose:**  
  This file sets up the connection to your PostgreSQL database using the pg package. It uses environment variables for configuration so that sensitive information (like username, password, host, and database name) is not hardcoded.
* **Key Points:**
  + **Environment Variables:**  
    Loaded via dotenv.config() before using them in the connection configuration.
  + **Pool:**  
    Creates a new Pool instance which manages a set of client connections to PostgreSQL, allowing your app to perform queries.

**Excerpt from ./db/db.js:**

**import** pkg from "pg"**;** // ESM import for 'pg'

**import** dotenv from "dotenv"**;**

dotenv**.**config**();**

**const** **{** Pool **}** **=** pkg**;**

**const** pool **=** **new** Pool**({**

user**:** process**.**env**.**PG\_USER**,**

host**:** process**.**env**.**PG\_HOST**,**

database**:** process**.**env**.**PG\_DATABASE**,**

**password:** process**.**env**.**PG\_PASSWORD**,**

port**:** process**.**env**.**PG\_PORT**,**

**});**

**export** **default** pool**;**

**4. database.sql**

* **Purpose:**  
  This file contains the SQL commands used to set up your PostgreSQL database schema.
* **Key Points:**
  + **CREATE DATABASE:**  
    Creates the database named perntodo.
  + **CREATE TABLE:**  
    Creates the todo table with two columns:
    - todo\_id: A serial primary key.
    - description: A variable character field with a maximum of 255 characters.

**Excerpt from database.sql:**

CREATE DATABASE perntodo;

CREATE TABLE todo(

todo\_id SERIAL PRIMARY KEY,

description VARCHAR(255)

);

**5. ./controllers/todosController.js**

* **Purpose:**  
  Contains the business logic for handling CRUD operations on todos. Each function corresponds to one of the endpoints defined in the routes file.
* **Key Points:**
  + **createTodo:**  
    Inserts a new todo into the database.
  + **getTodos:**  
    Retrieves all todos.
  + **getTodoById:**  
    Retrieves a specific todo by its ID.
  + **updateTodo:**  
    Updates the description of an existing todo, returning an error if the description is missing or the todo doesn't exist.
  + **deleteTodo:**  
    Deletes a todo by its ID.
  + **Error Handling:**  
    Each controller function catches errors and sends a 500 status code if something goes wrong.

**Excerpt from ./controllers/todosController.js:**

**import** pool from "../db/db.js"**;**

**export** **const** createTodo **=** **async** **(**req**,** res**)** **=>** **{**

**try** **{**

**const** **{** description **}** **=** req**.**body**;**

**const** newTodo **=** **await** pool**.**query**(**

"INSERT INTO todo (description) VALUES($1) RETURNING \*"**,**

**[**description**]**

**);**

res**.**json**(**newTodo**.**rows**[**0**]);**

**}** **catch** **(**err**)** **{**

console**.**error**(**err**.**message**);**

res**.status(**500**).**json**({** error**:** "Server Error" **});**

**}**

**};**

**export** **const** getTodos **=** **async** **(**req**,** res**)** **=>** **{**

**try** **{**

**const** allTodos **=** **await** pool**.**query**(**"SELECT \* FROM todo"**);**

res**.**json**(**allTodos**.**rows**);**

**}** **catch** **(**err**)** **{**

console**.**error**(**err**.**message**);**

res**.status(**500**).**json**({** error**:** "Server Error" **});**

**}**

**};**

**export** **const** getTodoById **=** **async** **(**req**,** res**)** **=>** **{**

**try** **{**

**const** **{** id **}** **=** req**.**params**;**

**const** todo **=** **await** pool**.**query**(**"SELECT \* FROM todo WHERE todo\_id = $1"**,** **[**id**]);**

res**.**json**(**todo**.**rows**[**0**]);**

**}** **catch** **(**err**)** **{**

console**.**error**(**err**.**message**);**

res**.status(**500**).**json**({** error**:** "Server Error" **});**

**}**

**};**

**export** **const** updateTodo **=** **async** **(**req**,** res**)** **=>** **{**

**try** **{**

**const** **{** id **}** **=** req**.**params**;**

**const** **{** description **}** **=** req**.**body**;**

**if** **(!**description**)** **{**

**return** res**.status(**400**).**json**({** error**:** "Description is required" **});**

**}**

**const** updatedTodo **=** **await** pool**.**query**(**

"UPDATE todo SET description = $1 WHERE todo\_id = $2 RETURNING \*"**,**

**[**description**,** id**]**

**);**

**if** **(**updatedTodo**.**rowCount **===** 0**)** **{**

**return** res**.status(**404**).**json**({** error**:** "Todo not found" **});**

**}**

res**.**json**({**

message**:** "Todo updated successfully!"**,**

todo**:** updatedTodo**.**rows**[**0**],**

**});**

**}** **catch** **(**err**)** **{**

console**.**error**(**err**.**message**);**

res**.status(**500**).**json**({** error**:** "Server Error" **});**

**}**

**};**

**export** **const** deleteTodo **=** **async** **(**req**,** res**)** **=>** **{**

**try** **{**

**const** **{** id **}** **=** req**.**params**;**

**await** pool**.**query**(**"DELETE FROM todo WHERE todo\_id = $1"**,** **[**id**]);**

res**.**json**({** message**:** "Todo was deleted!" **});**

**}** **catch** **(**err**)** **{**

console**.**error**(**err**.**message**);**

res**.status(**500**).**json**({** error**:** "Server Error" **});**

**}**

**};**