Full Stack To-Do List App Setup Guide

# 1. Project Setup

1.1. Create the Project Directory:  
 mkdir typescript-todo-app  
 cd typescript-todo-app  
  
1.2. Set up the Server (Back-End):  
 Create the server folder inside the project:  
 mkdir server  
 cd server  
  
 Initialize a Node.js project:  
 npm init -y  
  
 Install required packages for back-end (Express, TypeScript, etc.):  
 npm install express  
 npm install typescript ts-node @types/express @types/node --save-dev  
  
 Generate tsconfig.json by running:  
 npx tsc --init  
  
 Update tsconfig.json to set the target and module for compatibility:  
  
 {  
 "compilerOptions": {  
 "target": "ES6",  
 "module": "commonjs",  
 "strict": true,  
 "esModuleInterop": true  
 }  
 }  
  
 Create server.ts to handle basic routes for adding and fetching tasks:  
  
 import express, { Request, Response } from "express";  
 const app = express();  
 const PORT = 3000;  
 app.use(express.json());  
  
 let tasks: string[] = [];  
  
 // Endpoint to get all tasks  
 app.get("/tasks", (req: Request, res: Response) => {  
 res.json(tasks);  
 });  
  
 // Endpoint to add a new task  
 app.post("/tasks", (req: Request, res: Response) => {  
 const { task } = req.body;  
 tasks.push(task);  
 res.json({ message: "Task added!" });  
 });  
  
 app.listen(PORT, () => {  
 console.log(`Server is running on http://localhost:${PORT}`);  
 });  
  
 Start the back-end server by running:  
 npm start

# 2. Create the Client (Front-End)

2.1. Set up the React Front-End:  
 Create the client folder inside the main project directory:  
 cd ..  
 mkdir client  
  
 Use create-react-app with TypeScript to set up React:  
 npx create-react-app client --template typescript  
  
 Navigate to the client folder and start the React app:  
 cd client  
 npm start  
  
 Resolve the port conflict by choosing Y to run React on a different port (3001).

# 3. Connect Front-End to Back-End

Modify the React Component (App.tsx):  
Replace the default React code with our custom To-Do List functionality:  
  
import React, { useState, useEffect } from "react";  
  
const App: React.FC = () => {  
 const [tasks, setTasks] = useState<string[]>([]);  
 const [newTask, setNewTask] = useState<string>("");  
  
 // Fetch tasks from the back-end  
 const fetchTasks = async () => {  
 const response = await fetch("http://localhost:3000/tasks");  
 const data = await response.json();  
 setTasks(data);  
 };  
  
 // Add a new task to the back-end  
 const addTask = async () => {  
 if (newTask.trim()) {  
 await fetch("http://localhost:3000/tasks", {  
 method: "POST",  
 headers: {  
 "Content-Type": "application/json",  
 },  
 body: JSON.stringify({ task: newTask }),  
 });  
 setNewTask("");  
 fetchTasks(); // Refresh the list of tasks  
 }  
 };  
  
 useEffect(() => {  
 fetchTasks();  
 }, []);  
  
 return (  
 <div className="App">  
 <h1>To-Do List</h1>  
 <input  
 type="text"  
 value={newTask}  
 onChange={(e) => setNewTask(e.target.value)}  
 placeholder="Add a new task"  
 />  
 <button onClick={addTask}>Add Task</button>  
  
 <ul>  
 {tasks.map((task, index) => (  
 <li key={index}>{task}</li>  
 ))}  
 </ul>  
 </div>  
 );  
};  
  
export default App;

# 4. Resolving CORS Issues

4.1. Installing CORS in the Back-End:  
 We installed the cors package:  
 npm install cors  
  
 Updated server.ts to use CORS:  
 import cors from 'cors';  
 app.use(cors());  
  
 Restarted the back-end server to apply the changes:  
 npm start

# 5. Test and View the Final App

Test the app by adding tasks, which are displayed in the front-end and saved in the back-end at:  
http://localhost:3000/tasks (back-end)  
http://localhost:3001 (front-end)  
  
Now you have a fully functional full-stack To-Do List application!