Prerequisites to

build a ui code in typescript based on react js that allows a user to perform Crud operations by a rest api on class called person which has attributes such as first name, last name, and phone number. Use axios and postgresql

To build a UI code in TypeScript based on React.js that allows a user to perform CRUD operations on a "Person" resource using a REST API backed by PostgreSQL, you'll need a combination of front-end and back-end technologies. Here's what you'll need:

**Front-End (React.js with TypeScript and Axios):**

1. **Node.js and npm**: Make sure you have Node.js installed, which comes with npm (Node Package Manager).
2. **Create React App**: You can use Create React App to set up your React project with TypeScript:

bashCopy code

npx create-react-app my-crud-app --template typescript cd my-crud-app

1. **Axios**: Install Axios to make HTTP requests to your API:

bashCopy code

npm install axios

1. **React Components**: Create components for CRUD operations (Create, Read, Update, Delete) that interact with your REST API.

**Back-End (Node.js with Express.js and PostgreSQL):**

1. **Node.js and npm**: Ensure you have Node.js installed.
2. **Express.js**: Set up an Express.js server to handle API requests and routes.

bashCopy code

npm install express

1. **PostgreSQL**: Install and set up PostgreSQL for your database. You'll need to create a table for the "Person" resource with attributes such as first name, last name, and phone number.

bashCopy code

npm install pg

1. **Database Connection**: Set up a connection to your PostgreSQL database using a library like **pg**.
2. **API Endpoints**: Create API routes for CRUD operations that interact with the PostgreSQL database.

**Development Environment and Setup:**

1. **IDE or Code Editor**: Use an Integrated Development Environment (IDE) or a code editor like Visual Studio Code.
2. **API Documentation**: Document the API endpoints and their expected request and response formats.
3. **Environment Variables**: Use environment variables to manage sensitive information like database credentials and API keys.
4. **Testing**: Set up testing using tools like Jest for both front-end and back-end components.
5. **Error Handling**: Implement error handling and validation on both the front-end and back-end to ensure a smooth user experience.
6. **Security**: Implement security measures such as authentication and authorization to protect your API endpoints.
7. **Deployment**: Deploy your front-end and back-end components separately to suitable hosting platforms.

Remember that building a full-fledged application involves various aspects, from design to user experience to performance optimization. This overview provides a starting point for building your CRUD application using React.js, TypeScript, Axios, and PostgreSQL. It's recommended to break down the development process into smaller tasks and gradually build and test each component.

build a ui code in typescript based on react js that allows a user to perform Crud operations by a rest api on class called person which has attributes such as first name, last name, and phone number. Use axios and postgresql

Certainly! Here's a basic example of a React UI code using TypeScript, Axios, and PostgreSQL for performing CRUD operations on a **Person** class. This example focuses on the frontend part of the application. You should integrate this with your backend code that handles the REST API and interacts with the PostgreSQL database.

**Step 1: Set Up Your React App**

Make sure you have Node.js and npm installed. Then, you can create a new React app and navigate to the project directory:

bashCopy code

npx create-react-app react-crud-app --template typescript cd react-crud-app

**Step 2: Install Dependencies**

Install Axios for making API requests:

bashCopy code

npm install axios

**Step 3: Create Components**

Replace the content of **src/App.tsx** with the following code:

tsxCopy code

import React, { useState, useEffect } from 'react'; import axios from 'axios'; import './App.css'; interface Person { id: number; first\_name: string; last\_name: string; phone\_number: string; } const App: React.FC = () => { const [persons, setPersons] = useState<Person[]>([]); const [firstName, setFirstName] = useState(''); const [lastName, setLastName] = useState(''); const [phoneNumber, setPhoneNumber] = useState(''); useEffect(() => { fetchPersons(); }, []); const fetchPersons = async () => { try { const response = await axios.get<Person[]>('/api/person'); setPersons(response.data); } catch (error) { console.error('Error fetching persons:', error); } }; const createPerson = async () => { try { const response = await axios.post<Person>('/api/person', { first\_name: firstName, last\_name: lastName, phone\_number: phoneNumber, }); setPersons([...persons, response.data]); setFirstName(''); setLastName(''); setPhoneNumber(''); } catch (error) { console.error('Error creating person:', error); } }; return ( <div className="App"> <h1>CRUD App</h1> <div> <h2>Create Person</h2> <input type="text" placeholder="First Name" value={firstName} onChange={(e) => setFirstName(e.target.value)} /> <input type="text" placeholder="Last Name" value={lastName} onChange={(e) => setLastName(e.target.value)} /> <input type="text" placeholder="Phone Number" value={phoneNumber} onChange={(e) => setPhoneNumber(e.target.value)} /> <button onClick={createPerson}>Create</button> </div> <div> <h2>Persons</h2> <ul> {persons.map((person) => ( <li key={person.id}> {person.first\_name} {person.last\_name} ({person.phone\_number}) </li> ))} </ul> </div> </div> ); }; export default App;

This code sets up a React component that allows you to create and display persons using Axios to interact with the REST API.

**Step 4: Start the App**

Run your React app:

bashCopy code

npm start

Make sure you integrate the frontend with your backend that handles the REST API endpoints for CRUD operations on the **Person** class and interacts with the PostgreSQL database. This example focuses solely on the frontend part.

In package.jsoc replaced following two lines

// "start": "react-scripts start",

    // "build": "react-scripts build",

With

"start": "react-scripts --openssl-legacy-provider start",

"build": "react-scripts --openssl-legacy-provider build",