1. Setting up REST API Functions

a. GET (Fetch All Contacts)

1. **Create a service function**:
    - Define a function `ContactListJson` that uses `axios.get` to make a GET request to fetch all contacts.
2. **Handle Response in Component**:
    - In the `ContactsList` component, use the `useEffect` hook to call `getAllContacts` when the component mounts.
    - Define `getAllContacts` to call `ContactListJson`.
    - Use `then` to handle the response and update the component state (`contacts`) with the fetched data.
    - Use `catch` to handle any errors that occur during the fetch.

b. POST (Add a New Contact)

1. **Create a service function**:
    - Define a function `AddContactJson` that uses `axios.post` to make a POST request to add a new contact.
2. **Handle Request in Form**:
    - In the `ContactForm` component, initialize state for the form fields (name, email, phone).
    - Define a `handleSubmit` function that:
        - Prevents the default form submission behavior.
        - Calls `AddContactJson` with the new contact data.
        - Uses `then` to handle the response and update the parent component state with the new contact.
        - Uses `catch` to handle any errors.
    - Pass this `handleSubmit` function to the form's `onSubmit` event.
3. **Update Parent Component**:

- In the `ContactsList` component, define a function `handleAddContact` to update the state with the new contact.
- Pass this function as a prop to `ContactForm`.

c. PUT (Update an Existing Contact)

1. **Create a service function**:
   - Define a function `UpdateContactJson` that uses `axios.put` to make a PUT request to update an existing contact.
2. **Handle Request in Form**:
   - Modify the `ContactForm` to accept a `contact` prop for pre-filling form fields when editing.
   - Update the `handleSubmit` function to check if a contact is being edited:
     - If yes, call `UpdateContactJson` with the contact ID and updated data.
     - If no, call `AddContactJson`.
   - Use `then` to handle the response and update the parent component state.
   - Use `catch` to handle any errors.
3. **Update Parent Component**:
   - In the `ContactsList` component, define a function `handleEditContact` to set the contact being edited and open the form.
   - Define a function `handleUpdateContact` to update the state with the edited contact data.
   - Pass these functions as props to `ContactForm`.

d. DELETE (Remove a Contact)

1. **Create a service function**:
   - Define a function `DeleteContactJson` that uses `axios.delete` to make a DELETE request to remove a contact by ID.
2. **Handle Request in Component**:
   - In the `ContactsList` component, define a function `handleDeleteContact` to set the contact ID to be deleted and open a confirmation popup.

    - Define a function `ConfirmDeleteContact` that:
        - Calls `DeleteContactJson` with the contact ID.
        - Uses `then` to handle the response and update the state to remove the deleted contact.
        - Uses `catch` to handle any errors.
3. **Confirmation Popup**:
    - Add a confirmation popup component that calls `ConfirmDeleteContact` if the user confirms the deletion.
    - Update the state to close the popup and clear the contact ID.

2. Component Interactions

a. Display Contacts

- Use the `DataGrid` component to display the list of contacts fetched via the GET request.
- Define columns and specify action buttons for edit and delete.

b. Add/Edit Contact

- Open a popup with the `ContactForm` for adding or editing a contact.
- Pre-fill form fields with existing data if editing.
- Submit the form to either add a new contact (POST) or update an existing contact (PUT).

c. Delete Contact

- Open a confirmation popup when the delete button is clicked.
- Confirm the deletion to remove the contact (DELETE).

3. State Management

a. State Initialization and Updates

- Use `useState` hooks to manage the state of contacts, form visibility, and the contact to be deleted or edited.

- Update state based on user actions and API responses to ensure the UI reflects the current state of data.