INNOMATICS
RESEARCH LABS

Student Management Portal Project

Overview:

In this project, you will build a fully functional Student Management Portal using React. The goal is to develop a modern, responsive front-end application with multiple pages that allows for managing student information, such as viewing student details, registering new students, updating existing records, and deleting students.

You will be provided with JSON data representing student information, which will serve as your mock data.

Key Requirements:

- Utilize **React** for creating a component-based application.
- Implement **React Router** for navigation across different pages.
- Use **State Management** with **Context API** or **Redux Toolkit**.
- Develop a **responsive** and **user-friendly** interface using **HTML/CSS.**
- Integrate with a mock **Json Data** or **API** for data management.
- Implement **search**, **filtering**, and **pagination** functionalities.

Tasks to Implement:

Task 1: Student Registration Page

Create a form to allow users to register new students.

- **Fields:** Student Name, Email, Age, Class, Address, Phone Number.
- **Validation**: Ensure that all required fields are filled out and validate inputs (e.g., valid email, phone number format).
- **Functionality:** On form submission, the student should be added to the list of students.

Hints:

- Use controlled components in React for handling form inputs.
- Validate form data before adding the student.

Task 2: Student List Page

Create a page to display a list of all registered students.

- **Display:** Show a table with student details like Name, Email, Class, and Actions (View/Edit/Delete).
- **Search Bar:** Implement a search bar to filter students based on their name.
- **Pagination:** If there are many students, paginate the results (optional but encouraged).

Hints:

- Fetch the student data from your mock API or JSON file.
- Implement state management for handling student data.

Task 3: Student Details Page

Create a page to display detailed information about a specific student.

- **Routing:** Implement dynamic routing to open the details page for a specific student when clicked from the student list.
- **Data Display:** Show all the details of the selected student, including address and phone number.

Hints:

- Use React Router to navigate to the detailed student page.
- Pass the student ID or index in the URL to fetch the appropriate details.

Task 4: Update Student Information

Create a feature that allows users to edit/update student details.

- **Edit Form:** Prepopulate the existing student details in a form.
- **Validation:** Ensure the updated details are valid before submitting.
- **Save Changes:** Update the student details upon form submission.

Hints:

- Use controlled components to handle form inputs.
- You can reuse the form component from the registration page with pre-filled data.

Task 5: Delete Student

Allow users to delete a student from the list.

- **Confirmation:** Before deleting, confirm the action with the user.
- **Update List:** After deletion, remove the student from the list and update the state.

Hints:

- Implement a delete button on each student row in the student list.
- Update the state and the UI dynamically after the deletion.

Task 6: Dashboard and Navigation

Create a simple dashboard that gives an overview of the total number of students and other relevant statistics.

- **Dashboard:** Display the total number of students, a chart showing students per class (optional), and other statistics.
- **Navigation Bar:** Implement a navigation bar that allows users to navigate between different pages (e.g., Dashboard, Student List, Register Student).

Hints:

- Use React Router for handling page navigation.
- Use a chart library like Chart.js or Recharts for displaying statistics (optional).

Task 7: Search and Filter Functionality

Enhance the Student List Page by adding:

- **Search Functionality:** Users should be able to search for students by their name.
- **Filter by Class:** Provide a dropdown filter to let users filter students based on their class (e.g., 10th Grade, 11th Grade, etc.).
- **Sorting:** Add sorting options to allow sorting of students based on name or class.

Hints:

- Use the `Array.filter()` method to implement search and filtering.
- Use the `Array.sort()` method to implement sorting.

Task 8: Pagination for Student List

Implement pagination to manage large datasets.

- **Pagination Controls:** Add buttons or controls to paginate through the student list (e.g., showing 10 students per page).
- **Functionality:** When the user navigates to a different page, the displayed students should update accordingly.

Hints:

- Maintain the current page number in the state and use it to calculate which students to display.
- Use array slicing to display only the students for the current page.

Additional Features:

- **Responsive Design:** Ensure that the portal is mobile-friendly and adapts to different screen sizes.
- **Sorting and Filtering:** Implement sorting (e.g., by name or class) and filtering options on the **Student List Page**.

Technologies to Use:

- **React:** For building the UI components.
- **React Router:** For routing and navigation between pages.
- **CSS/SCSS or a CSS Framework (like Bootstrap):** For styling and responsiveness.
- **State Management (Context API or Redux Toolkit):** For managing the state of the student data.
- **HTML5/CSS3:** For structuring and designing the pages.

Mock API / Data:

You can use the following **JSON file as the mock data** for your project:
Json Data : https://dpaste.com/DBQS4CGRZ

Mock JSON File is attached in the discord message as well as in LMS

Alternatively, you can simulate API requests using JSON Placeholder or a similar service.

Final Deliverables:

1. A fully functional **Student Management Portal** with all features mentioned.
2. Clean, well-organized, and readable code.
3. A responsive design that works well on both desktop and mobile devices.

Good luck, and focus on building a scalable and user-friendly application!