Project Title:

PrimeReact DataTable with Multi-Page Row Selection

Project Overview

This project is a React-based application using the **PrimeReact** library to display and manage artwork data fetched from the Art Institute of Chicago API. The application implements a DataTable with **server-side pagination**, row selection features, and interactive user actions such as multi-page row selection, double-click row selection, and selection clearing.

The application demonstrates efficient use of React components, API integration, state management, and user-friendly UI to provide seamless navigation and functionality.

Technologies Used

- **React**: For building the user interface.
- **TypeScript**: For type safety and clean coding practices.
- **PrimeReact**: For components like DataTable, Column, Button, and Dialog.
- **CSS**: For custom styles and UI improvements.
- **Fetch API**: To retrieve data from the Art Institute of Chicago API.

Features

1. Server-Side Pagination

- Data is fetched in chunks using pagination, ensuring performance optimization when handling large datasets.
- The user can navigate between pages using the paginator at the bottom of the DataTable.

---

2. Row Selection

a. Single Page Selection

- Users can **manually select rows** on the current page.
- The selection persists only for the rows available on the displayed page.

b. Multi-Page Row Selection

- A custom feature allows users to select rows across **multiple pages**.
- Users can input the desired number of rows to be selected across pages using a **dialog box**.

Implementation:
The application iterates over the API's paginated data, dynamically fetching records until the specified number of rows are selected.

---

3. Clear All Selections

- A **Clear Selection** button is provided to clear all selected rows.
- This is particularly useful when users want to start over and deselect all items with a single click.

---

4. Double-Click to Select a Row

- Users can quickly **double-click** on any row to select it.
- This clears any previous selections and only selects the row the user interacted with.

Use Case:
When the user wants to select a single row without manually clearing previous selections.

---

5. Responsive Design

- The DataTable layout is responsive, ensuring the application works smoothly on various screen sizes.
- The table adapts dynamically to display rows and columns in a scrollable format when space is constrained.

---

6. Data Display

- The application displays key details about artworks fetched from the API:
    - **ID**: Unique identifier of the artwork.
    - **Title**: Title of the artwork.
    - **Place of Origin**: Where the artwork originated.
    - **Artist**: Artist's name or details.
    - **Inscriptions**: Any inscriptions on the artwork.
    - **Start Date & End Date**: The timeline of the artwork.

- **Image**: A thumbnail image fetched dynamically via the Art Institute API.

---

7. Error Handling

- Proper error handling is implemented for failed API requests.
- If data fetching fails, the user is alerted with an appropriate error message.

Problems Faced and Solutions

1. Row Selection Across Pages

Problem:
By default, the PrimeReact DataTable only supports row selection on the current page. When navigating between pages, selections would reset.

Solution:

- A custom logic was implemented to store selected rows **across pages**.
- The solution dynamically fetches data across multiple pages and merges selected rows into a single array, ensuring previous selections persist.

---

2. Handling Large Data Sets

Problem:
The Art Institute of Chicago API contains a large number of records, which could lead to performance issues if all data is fetched at once.

Solution:

- **Server-side pagination** was used to fetch small chunks of data (12 rows per page).
- This improves performance by reducing the amount of data loaded at any given time.

---

3. Type Safety with TypeScript

Problem:
When mapping fetched data to components, inconsistent API data could lead to runtime errors.

Solution:

- An Artwork interface was created to define the structure of the data retrieved from the API.
- This ensures type safety, helping identify potential data-related issues at compile time.

---

4. UI Responsiveness

Problem:
The DataTable could overflow on smaller screens, affecting usability.

Solution:

- Used PrimeReact's **responsiveLayout** property and custom styles to ensure the table adapts to smaller screen sizes.
- Images and text were adjusted to scroll when space is limited.

---

5. Improving User Experience

Challenges:

- Providing an intuitive way to select rows across multiple pages.
- Allowing quick single-row selection with a double-click.

Solutions:

- A **dialog box** allows users to input the number of rows they want to select across pages.
- The **double-click** event handler ensures a single row is selected immediately without navigating through menus.

How the Project Works

1. **Data Fetching**:
    - The application fetches paginated artwork data from the API when the user navigates between pages.
2. **Row Selection**:
    - Rows can be selected manually, across multiple pages, or using double-click functionality.
3. **Clear Selection**:
    - A clear selection button is available to reset all selected rows.
4. **User Dialog**:
    - The user can input how many rows they want to select across pages. The application will dynamically fetch and select rows.
5. **Data Rendering**:
    - Each artwork's details, including an image, are displayed in the table.

Conclusion

This project demonstrates a robust implementation of a paginated DataTable with advanced row selection features. It successfully overcomes limitations like single-page row selection and unresponsive UI to deliver an intuitive and high-performing user experience.

The use of **PrimeReact**, **React hooks**, and API integration showcases strong front-end development skills and problem-solving abilities.

Future Enhancements

1. Implement search and filter functionality for artworks.
2. Add sorting options for table columns.
3. Allow the user to export selected rows as a PDF or Excel file.
4. Enhance image display with a gallery or modal view.

Final Notes

This project combines React, TypeScript, and PrimeReact to demonstrate interactive, real-world table functionality. Overcoming challenges like multi-page row selection highlights the problem-solving and logical approach taken to ensure a seamless user experience.