Paytronix FE Test Questions

Data available in src/data/cars.js

Question 1: Data Display

Task:

Create a React component that fetches the car data from a provided JSON file and displays it in a table. The table should have the following columns:
- Make
- Model
- Year
- Price
- Horsepower
- Fuel Efficiency (City MPG / Highway MPG)

Requirements:

- Use functional components and React hooks.
- Implement pagination to display 10 cars per page.
- Include a search bar to filter cars by make and model.
- Make the car name clickable

Question 2: Sorting and Filtering

Task:

Extend the previous React component to add sorting functionality. Users should be able to click on the table headers to sort the data by the following columns:
- Year
- Price
- Horsepower
- Fuel Efficiency (City MPG / Highway MPG)

Requirements:

- Implement sorting in both ascending and descending order.
- Ensure the sorting feature works in conjunction with the search functionality.

Question 3: Detailed View

Task:
Add a feature to the table where clicking on a car row opens a modal displaying detailed information about the selected car, including its additional features.

Requirements:
- The modal should display all the details of the car, including the make, model, year, price, engine type, horsepower, fuel efficiency, transmission, and additional features.
- Use your creativity and css skills to design and beautify the component

Question 4: Data Visualization

Task:
Create a dashboard that displays the following charts:
- A bar chart showing the number of cars per make.
- A pie chart showing the distribution of cars by fuel type (Electric, I4, V6, etc.).
- A line chart showing the average price of cars per year.

Requirements:
- Use a charting library like Chart.js or D3.js.
- Ensure the charts are responsive and update dynamically as the car data changes.

Question 5: Performance Optimization

Task:
Optimize the React application for performance, especially when dealing with a large dataset.

Requirements:
- Implement lazy loading for the car data.
- Use memoization techniques to prevent unnecessary re-renders.
- Ensure the application remains performant with up to 1000 car entries.

Question 6: API Integration

Task:
Assume the car data is available through an API endpoint. Modify your application to fetch the car data from the API instead of a local JSON file.

Requirements:

- Use `fetch` or `axios` to retrieve the data from the API.
- Handle loading states and display a spinner while the data is being fetched.
- Implement error handling to display an error message if the API request fails.

Bonus Question: Advanced Filtering

Task:

Implement an advanced filtering system that allows users to filter cars by multiple criteria, including make, model, year range, price range, and fuel type.

Requirements:

- Use a form or set of checkboxes/dropdowns for the filtering options.
- Combine the filtering functionality with the existing search and sorting features.
- Ensure the filtered results are displayed dynamically as the user adjusts the filters.