Experiment 10

Design a React application featuring a class-based component that demonstrates the use of lifecycle methods to interact with an external API. The component should fetch and update data dynamically based on user interactions or state changes. Use the componentDidMount lifecycle method to fetch data from an API when the component is initially rendered. Display the fetched data in a structured format, such as a table or list. Use the componentDidUpdate lifecycle method to detect changes in the component's state or props. Trigger additional API calls to update the displayed data based on user input or actions (e.g., filtering, searching, or pagination). Implement error handling to manage issues such as failed API requests or empty data responses. Display appropriate error messages to the user when necessary. Allow users to perform actions like filtering, searching, or refreshing the data. Reflect changes in the displayed data based on these interactions.

ApiDataFetcher.js

```
import React, { Component } from 'react';
class ApiDataFetcher extends Component {
constructor(props) {
super(props);
this.state = {
data: [],
filteredData: [],
loading: true,
error: null,
searchQuery: '',
};
}
// Fetch data when the component mounts
componentDidMount() {
this.fetchData();
}
// Fetch data from the API
fetchData = () => {
this.setState({ loading: true });
fetch('https://jsonplaceholder.typicode.com/posts') // Replace with
your API URL
.then((response) => response.json())
.then((data) => {
this.setState({ data, filteredData: data, loading: false });
})
.catch((error) => {
this.setState({ error: 'Error fetching data', loading: false });
});
}; // Trigger an API call when search query or filters change
componentDidUpdate(prevProps, prevState) {
if (prevState.searchQuery !== this.state.searchQuery) {
this.filterData();
}
```

```
}
// Filter data based on the search query
filterData = () => {
const { data, searchQuery } = this.state;
const filteredData = data.filter((item) =>
item.title.toLowerCase().includes(searchQuery.toLowerCase())
);
this.setState({ filteredData });
};
// Handle search input change
handleSearchChange = (event) => {
this.setState({ searchQuery: event.target.value });
};
// Handle refresh button click
handleRefresh = () => {
this.fetchData();
};
render() {
const { filteredData, loading, error, searchQuery } = this.state;
return (
<div className="api-data-fetcher">
<h1>API Data Fetcher</h1>
{loading && <p>Loading...</p>}
{error && <p>{error}</p>}
```

```
{/* Search Input */}
<input
type="text"
placeholder="Search by title"
value={searchQuery}
onChange={this.handleSearchChange}
/>
{/* Refresh Button */}
<button onClick={this.handleRefresh}>Refresh Data</button>
{/* Data Table */}
{filteredData.length > 0 && (
<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Body</th>
</tr>
</thead>
<tbody>
{filteredData.map((item) => (
<tr key={item.id}>
<td>{item.id}</td>
<td>{item.title}</td>
<td>{item.body}</td>
</tr>
))}
</tbody>
</table>
)}
{/* Message when no results are found */}
{filteredData.length === 0 && !loading && !error && (
<p>No results found.</p>
)}
</div>
);
}
}
export default ApiDataFetcher;
```