**LAB10**

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.

App.js

import React, { Component } from 'react';const API\_URL = 'https://jsonplaceholder.typicode.com/users';class DataFetcher extends Component {

  constructor(props) {

  super(props);

  this.state = {

  data: [],

  filteredData: [],

  searchQuery: '',

  error: null,

  loading: false,

  };

  }componentDidMount() {

  this.fetchData();

  }fetchData = async () => {

  this.setState({ loading: true, error: null });

  try {

  const response = await fetch(API\_URL);

  if (!response.ok) {

  throw new Error('Failed to fetch data');

  }

  const data = await response.json();

  this.setState({ data, filteredData: data, loading: false });

  } catch (error) {

  this.setState({ error: error.message, loading: false });

  }

  };componentDidUpdate(prevProps, prevState) {

  if (prevState.searchQuery !== this.state.searchQuery) {

  this.filterData();

  }

  }handleSearchChange = (event) => {

  this.setState({ searchQuery: event.target.value });

  };filterData = () => {

  const { data, searchQuery } = this.state;

  if (searchQuery.trim() === '') {

  this.setState({ filteredData: data });

  } else {

  const filteredData = data.filter((item) =>

  item.name.toLowerCase().includes(searchQuery.toLowerCase())

  );

  this.setState({ filteredData });

  }

  };renderError = () => {

  const { error } = this.state;

  return error ? <div className="error">{`Error: ${error}`}</div> : null;

  };render() {

  const { filteredData, searchQuery, loading } = this.state;return (

  <div className="data-fetcher">

  <h1>User Data</h1>{this.renderError()}<div className="search-bar">

  <input

  type="text"

  value={searchQuery}

  onChange={this.handleSearchChange}

  placeholder="Search by name"

  />

  </div>{loading ? (

  <div>Loading...</div>

  ) : (

  <table>

  <thead>

  <tr>

  <th>Name</th>

  <th>Email</th>

  <th>City</th>

  </tr>

  </thead>

  <tbody>

  {filteredData.length > 0 ? (

  filteredData.map((item) => (

  <tr key={item.id}>

  <td>{item.name}</td>

  <td>{item.email}</td>

  <td>{item.address.city}</td>

  </tr>

  ))

  ) : (

  <tr>

  <td colSpan="3">No results found.</td>

  </tr>

  )}

  </tbody>

  </table>

  )}<button onClick={this.fetchData}>Refresh Data</button>

  </div>

  );

  }

  }export default DataFetcher;

Index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import './App.css';

import DataFetcher from './App';const root = ReactDOM.createRoot(document.getElementById('root'));root.render(

<React.StrictMode>

<DataFetcher />

</React.StrictMode>

);