**WEEK – 6(HandsOn)**

**React**

**4. ReactJS-HOL**

**Explain the need and Benefits of component life cycle:**

* **Need:**React components go through stages: creation, updating, and destruction. Lifecycle methods allow you to run custom logic at specific times in this process, such as fetching data after a component mounts, cleaning up timers before a component unmounts, or responding to props/state changes**.**
* **Benefits:**
  + Controlled Side Effects: Run code (e.g., API calls or subscriptions) at the right time.
  + Resource Management: Clean up resources (timers, listeners) to prevent memory leaks.
  + Performance Optimization: Decide when updates should happen (e.g., by using shouldComponentUpdate) to avoid unnecessary rendering.
  + Error Handling: Catch errors in child components using methods like componentDidCatch to avoid app crashes.

**Identify various life cycle hook methods**

Lifecycle methods (for class components) are grouped by when they run:

* **Mounting (when component is created):**
  + constructor()
  + static getDerivedStateFromProps()
  + render()
  + componentDidMount()
* **Updating (when state or props change):**
  + static getDerivedStateFromProps()
  + shouldComponentUpdate()
  + render()
  + getSnapshotBeforeUpdate()
  + componentDidUpdate()
* **Unmounting (when component is removed):**
  + componentWillUnmount()
* **Error Handling:**
  + componentDidCatch()

**List the sequence of steps in rendering a component:**

1. **Mounting Phase**
   * constructor() (Initialize state and bind methods)
   * static getDerivedStateFromProps() (Set state based on props, if needed)
   * render() (Returns JSX to display UI)
   * componentDidMount() (Runs after the component is added to the DOM)
2. **Updating Phase (happens when props or state change)**
   * static getDerivedStateFromProps() (Again, before every re-render)
   * shouldComponentUpdate() (Decide if render should proceed)
   * render() (Re-render with new data)
   * getSnapshotBeforeUpdate() (Capture current values, e.g., scroll position, before DOM updates)
   * componentDidUpdate() (Runs after changes are flushed to the DOM)
3. **Unmounting Phase**
   * componentWillUnmount() (Cleanup before component removal)
4. **Error Handling Phase**
   * componentDidCatch() (If any child component throws an error during rendering or lifecycle)

**CODE:**

**src/Post.js:**

class Post {

  constructor(id, title, body) {

    this.id = id;

    this.title = title;

    this.body = body;

  }

}

export default Post;

**src/Posts.js:**

import React, { Component } from 'react';

import Post from './Post';

class Posts extends Component {

  constructor(props) {

    super(props);

    this.state = {

      posts: [],

      hasError: false

    };

  }

  // Method to fetch posts from API

  loadPosts = () => {

    fetch('https://jsonplaceholder.typicode.com/posts')

      .then(response => response.json())

      .then(data => {

        const postList = data.map(post => new Post(post.id, post.title, post.body));

        this.setState({ posts: postList });

      })

      .catch(error => {

        console.error("Error while fetching posts:", error);

        this.setState({ hasError: true });

      });

  };

  // Lifecycle hook: runs after component is mounted

  componentDidMount() {

    this.loadPosts();

  }

  // Lifecycle hook: catches rendering errors

  componentDidCatch(error, info) {

    console.error("Error caught in componentDidCatch:", error, info);

    alert("An error occurred while displaying the posts.");

    this.setState({ hasError: true });

  }

  // Renders the posts to the screen

  render() {

    if (this.state.hasError) {

      return <h2>Something went wrong while displaying the posts.</h2>;

    }

    return (

      <div style={{ padding: '20px' }}>

        <h1>Blog Posts</h1>

        {this.state.posts.map(post => (

          <div key={post.id} style={{ marginBottom: '20px' }}>

            <h3>{post.title}</h3>

            <p>{post.body}</p>

          </div>

        ))}

      </div>

    );

  }

}

export default Posts;

**src/App.js:**

import React from 'react';

import './App.css';

import Posts from './Posts';

function App() {

  return (

    <div className="App">

      <Posts />

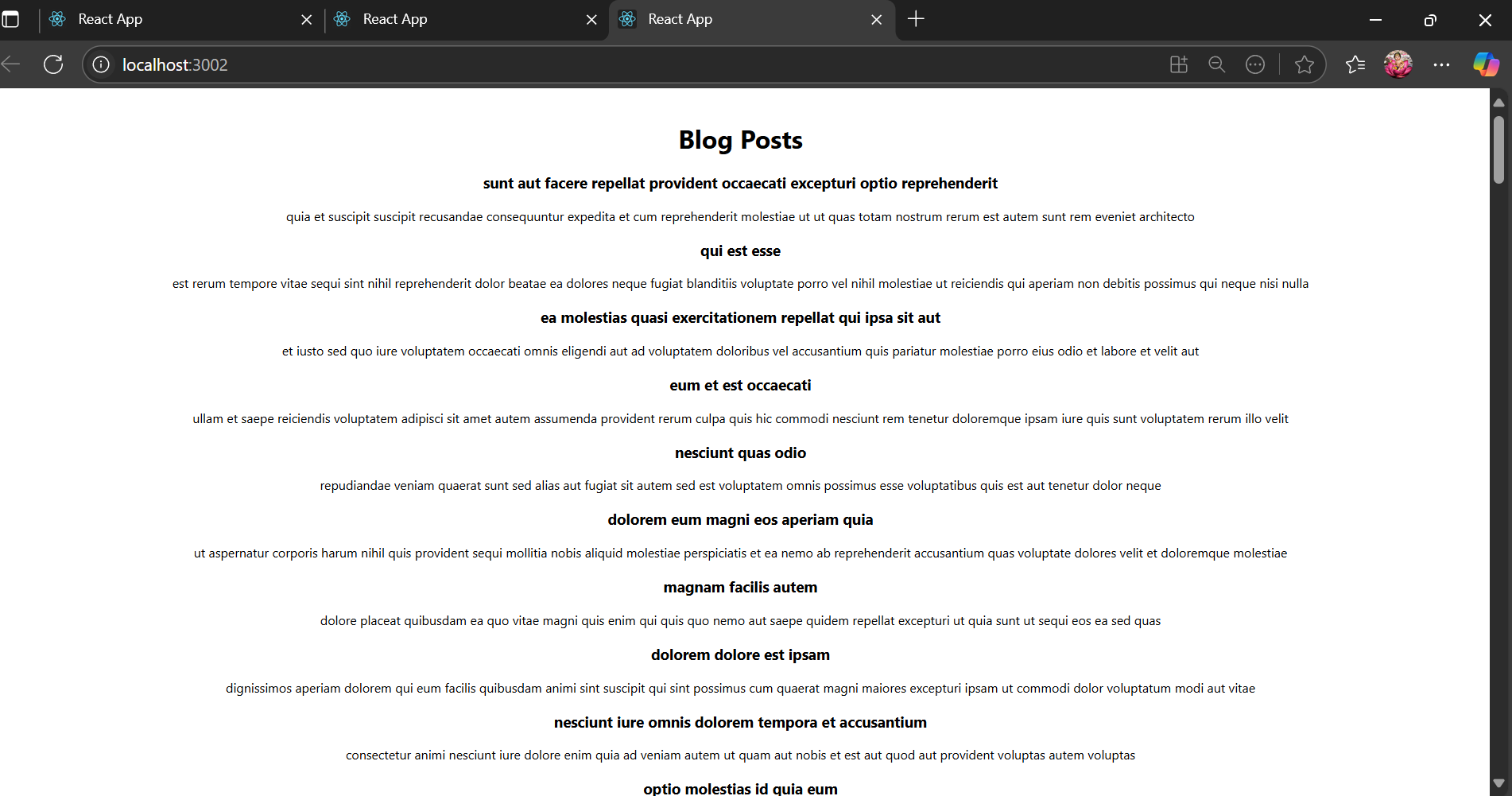
    </div>

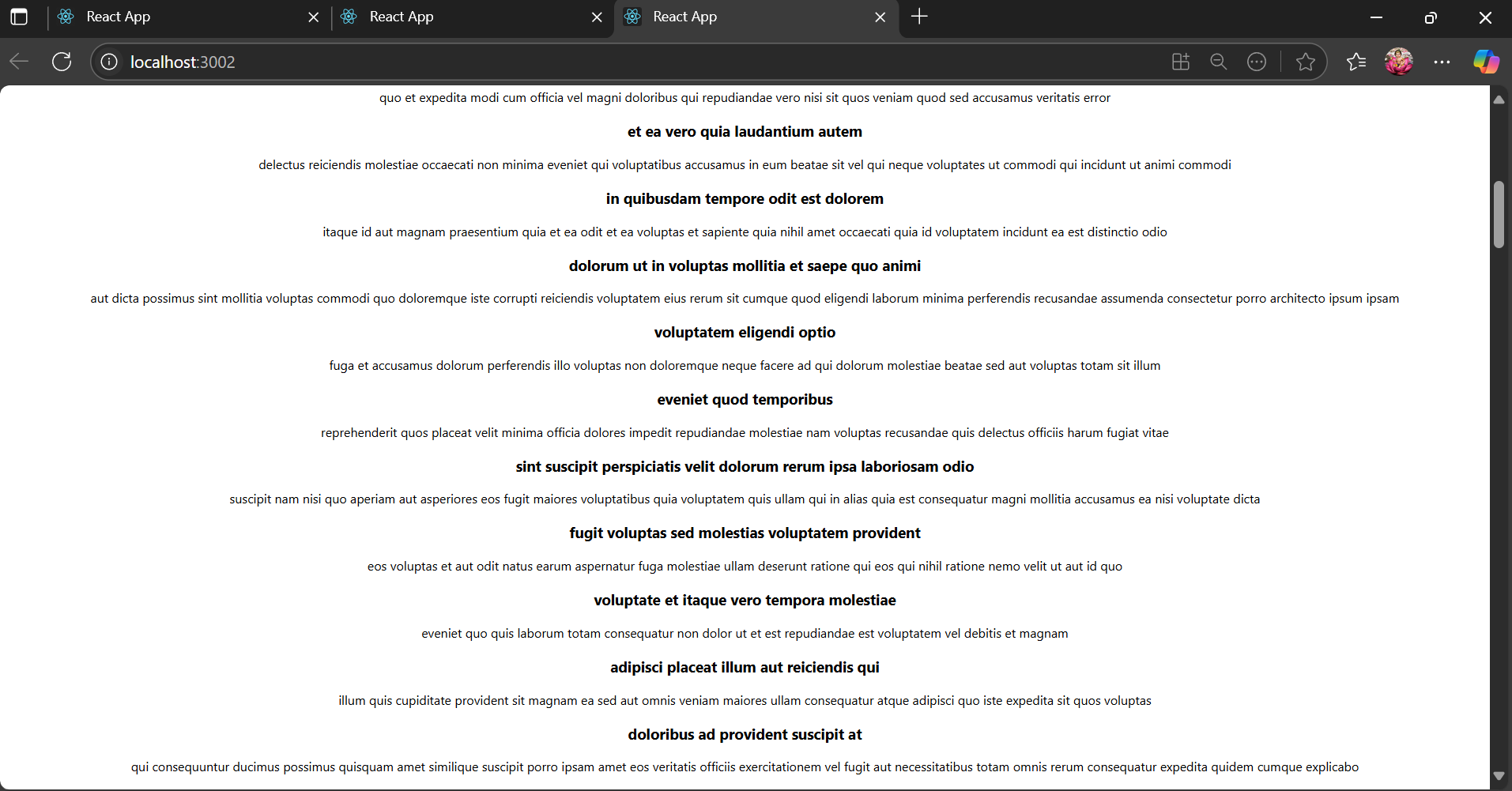
  );

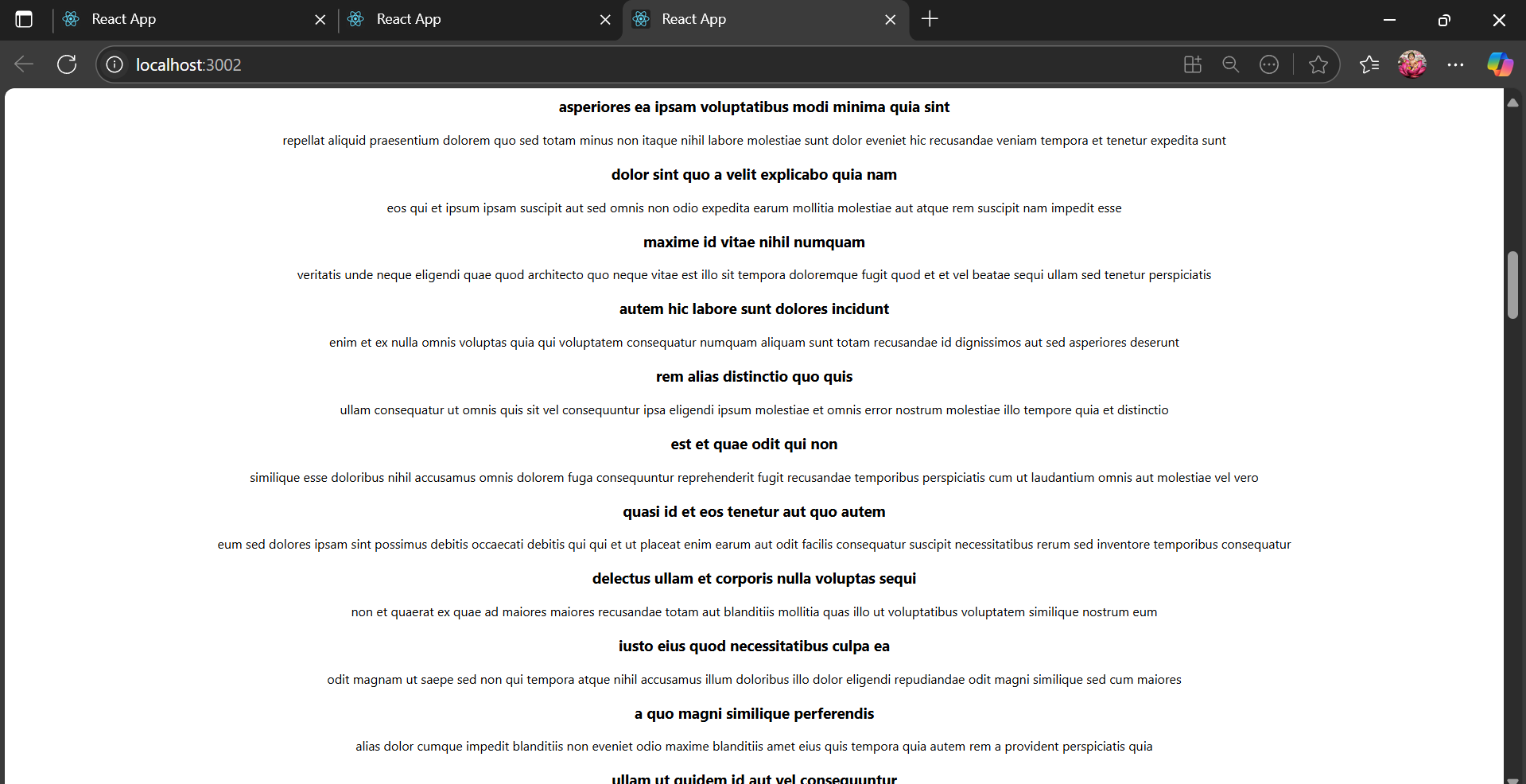
}

export default App;

**OUTPUT:**

****

****

****