**Objectives**

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

In React, components go through a series of stages from creation to removal. This process is called the component life cycle. Understanding the life cycle is important because it gives developers control over what happens at each stage – such as when the component is first added to the page, when it updates, and when it is removed.

The benefits of the component life cycle include:  
**1. Better control over behavior:** You can run specific code at different stages, like fetching data after the component loads.

**2. Efficient resource management:** Clean up tasks like removing timers or canceling network requests can be done before a component unmounts.

**3. Error handling:** You can catch and manage errors gracefully without breaking the entire application.

**4. Performance optimization:** With hooks like shouldComponentUpdate, unnecessary renders can be avoided, improving performance.

**Identify various life cycle hook methods**

React provides several built-in life cycle methods (or “hooks”) in class components that allows us to tap into different stages of the component’s life:

1. constructor(props): Called first, used to initialize state and bind methods.
2. render(): Called during mounting and updating, returns the JSX to display.
3. componentDidMount(): Runs after the component is added to the DOM. Ideal for data fetching.
4. componentDidUpdate(prevProps, prevState): Called after updates, useful for responding to changes.
5. componentWillMount(): Called before the component is removed, used for cleanup.
6. componentDidCatch(error, info): handles error in child components and prevents the app from crashing.

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

1. constructor(): Component is being created. State is initialized here.
2. render(): JSX is returned and virtual DOM is created.
3. componentDidMount(): After the component is inserted into the real DOM, this method runs.
4. (Updates): If props or state change:
5. render() runs again.
6. componentDidMount() is triggered after re-render.
7. (Error handling): If there’s an error during rendering, componentDidCatch() handles it.
8. componentWillUnmount(): When the component is about to be removed, this runs for cleanup.

**In this hands-on lab**

* **Implement componentDidMount() hook**
* **Implementing componentDidCatch() life cycle hook.**

**Codes:**

**Post.js:**

class **Post**{

    constructor(id,title,body){

        this.id=id;

        this.title=title;

        this.body=body;

    }

}

export default **Post**;

**Posts.js:**

import React from 'react';

import Post from './Post';

class **Posts** extends **React**.**Component**{

    constructor(props){

        super(props);

        this.state={

            posts:[]

        };

    }

**loadPosts**(){

**fetch**("https://jsonplaceholder.typicode.com/posts")

        .**then**(res=>res.**json**())

        .**then**(data=>{

            const posts=data.**map**(p=>new **Post**(p.id,p.title,p.body));

            this.**setState**({posts});

        })

        .**catch**(err=>{

            console.**error**("Failed to fetch posts:",err);

        });

    }

**componentDidMount**(){

        this.**loadPosts**();

    }

**componentDidCatch**(error,info){

**alert**("Something went wrong!");

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

    }

**render**(){

        return(

            <div>

                <h1>

                    Posts

                </h1>

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

                    <div *key*={post.id}>

                        <h3>

                            {post.title}

                        </h3>

                        <p>

                            {post.body}

                        </p>

                    </div>

                ))

                }

            </div>

        );

    }

}

export default **Posts**;

**App.js:**

import React from 'react';

import './App.css';

import Posts from './Posts';

function **App**() {

  return (

    <div *className*="App">

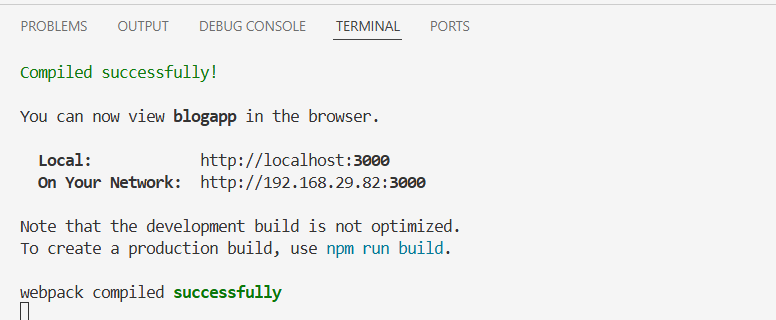
      <**Posts**/>

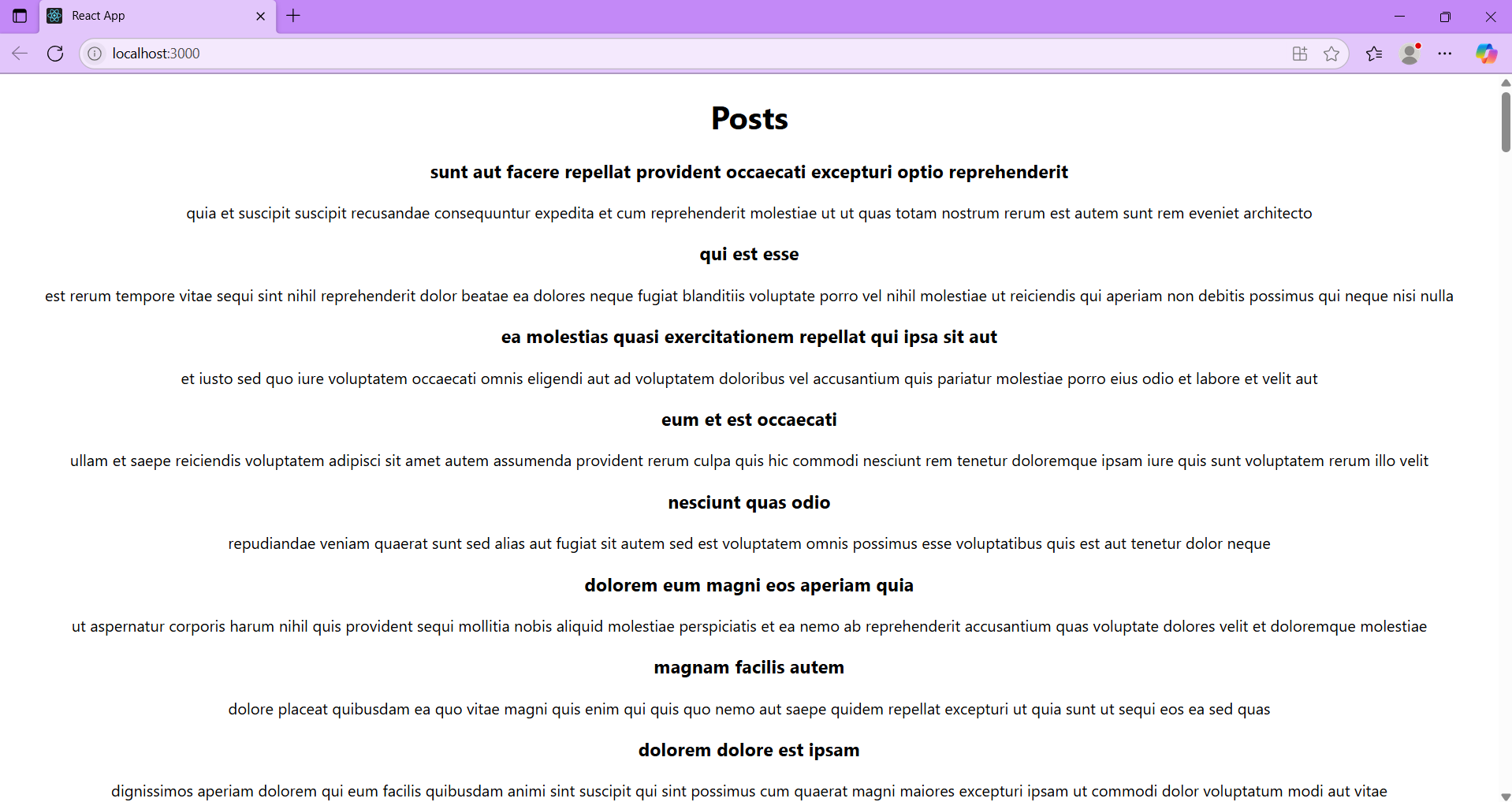
    </div>

  );

}

export default **App**;

**Output:  
**

****

****

****