# 4. ReactJS-HOL

# Objectives

• Explain the need and benefits of component life cycle

• Identify various lifecycle hook methods

• List the sequence of steps in rendering a component

## What is a Component Lifecycle?

In React, each component goes through a lifecycle – from creation (mounting), update (rendering), and finally unmounting (removal). Understanding this lifecycle helps developers:

• Execute code at specific points (e.g., fetching data after component is loaded)

• Manage side effects and cleanup

• Catch and handle errors effectively

## Benefits of Component Lifecycle

• Better control over rendering and performance

• Integration of external systems or APIs

• Clean-up memory (unsubscribe listeners, timers)

• Error handling during rendering or API calls

## Lifecycle Hook Methods (Class Components)

• constructor() – Initializes state and binds methods

• componentDidMount() – Called after component is rendered in the DOM (ideal for API calls)

• componentDidUpdate() – Called after component updates due to props/state changes

• componentWillUnmount() – Used for clean-up before the component is removed

• componentDidCatch() – Used for catching rendering errors and displaying fallback UI

## Component Rendering Steps

• constructor()

• render()

• componentDidMount()

• On state/prop change → render() → componentDidUpdate()

• When component removed → componentWillUnmount()

# Hands-On Lab: Working with Lifecycle Hooks

## Steps

### Step 1: Create a New React App

npx create-react-app blogapp  
cd blogapp

### Step 2: Create Post.jsx in src/ Folder

import React from 'react';  
  
class Post extends React.Component {  
 render() {  
 return (  
 <div>  
 <h2>{this.props.title}</h2>  
 <p>{this.props.body}</p>  
 </div>  
 );  
 }  
}  
  
export default Post;

### Step 3: Create Posts.jsx as a Class Component

import React, { Component } from 'react';  
import Post from './Post';  
  
class Posts extends Component {  
 constructor(props) {  
 super(props);  
 this.state = {  
 posts: [],  
 };  
 }  
  
 loadPosts = () => {  
 fetch('https://jsonplaceholder.typicode.com/posts')  
 .then(response => response.json())  
 .then(data => this.setState({ posts: data }))  
 .catch(error => {  
 throw error;  
 });  
 };  
  
 componentDidMount() {  
 this.loadPosts();  
 }  
  
 componentDidCatch(error, info) {  
 alert("An error occurred: " + error.toString());  
 }  
  
 render() {  
 return (  
 <div>  
 <h1>Blog Posts</h1>  
 {this.state.posts.map(post => (  
 <Post key={post.id} title={post.title} body={post.body} />  
 ))}  
 </div>  
 );  
 }  
}  
  
export default Posts;

### Step 4: Add Posts to the App.jsx

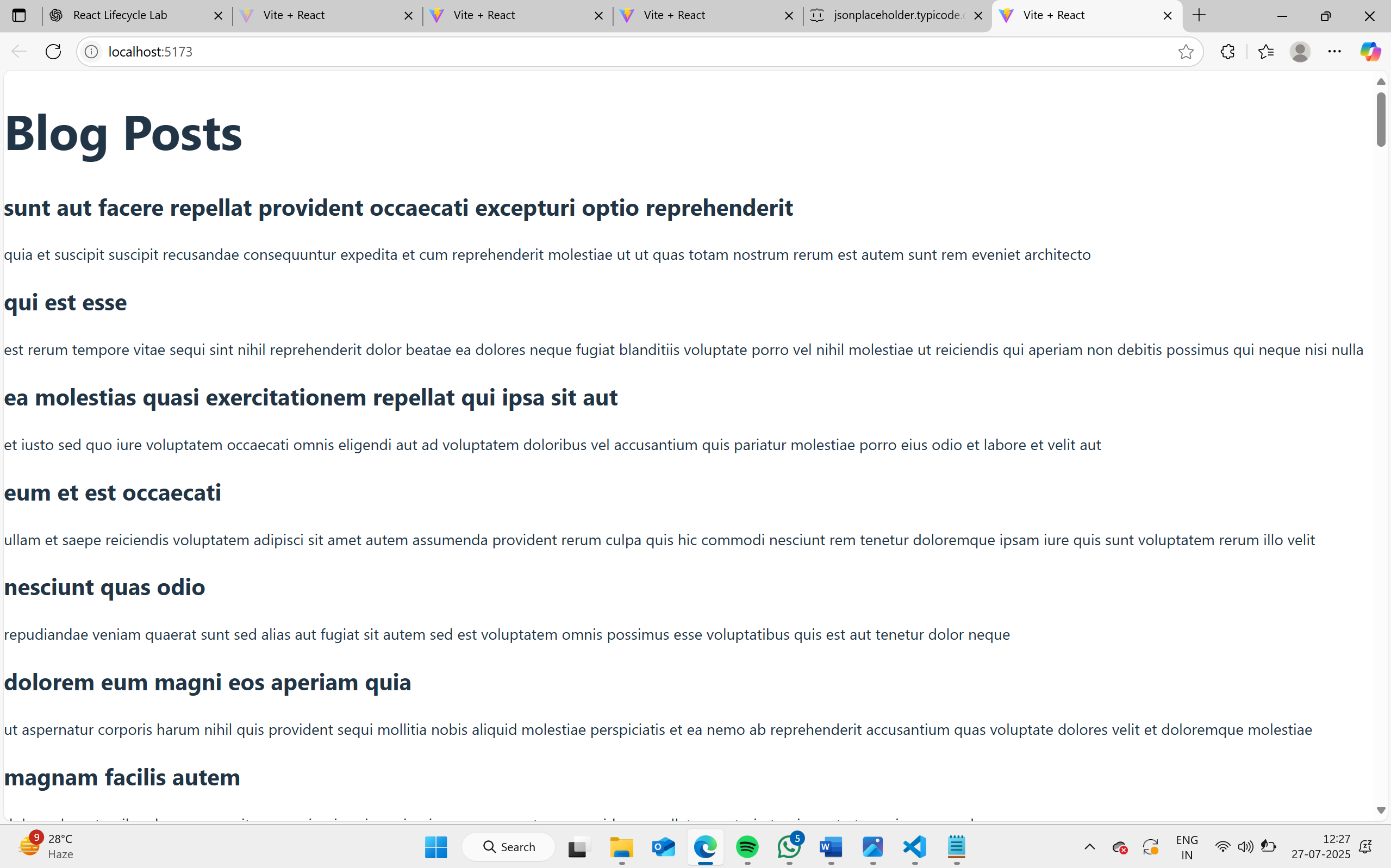
import React from 'react';  
import Posts from './Posts';  
  
function App() {  
 return (  
 <div className="App">  
 <Posts />  
 </div>  
 );  
}  
  
export default App;

### Step 5: Run the Application

npm start

## Output

- On successful fetch, posts will display on the screen.

  
- If there's an error, an alert message will be shown using componentDidCatch().