# React Lab Report - BlogApp (Lifecycle Methods)

## Introduction

In this lab, I explored component lifecycle methods in React by creating a blog application named 'blogapp'. I implemented class-based components using lifecycle hooks like componentDidMount and componentDidCatch.

## Objectives and What I Learned

* Objectives:
* Explain the need and Benefits of component life cycle
* Identify various life cycle hook methods
* List the sequence of steps in rendering a component
* What I Learned:
* Component lifecycle helps manage mounting, updating, and unmounting logic.
* componentDidMount is useful for fetching data after component is rendered.
* componentDidCatch handles unexpected errors in component rendering.
* The sequence of lifecycle: constructor → render → componentDidMount → updates → componentDidCatch (if error).

## Hands-on Practice Summary

Created a new class component 'Posts' that maintains state with post list. Fetched data from a public API using loadPosts() and displayed post titles and content. Used lifecycle hooks componentDidMount and componentDidCatch.

## Code Snippets

### App.js

import React from "react";

import Posts from "./Posts";

function App() {

  return (

    <div className="App">

      <h1>Welcome to BlogApp</h1>

      <Posts />

    </div>

  );

}

export default App;

### Posts.js

import React, { Component } from "react";

import Post from "./Post";

class Posts extends Component {

  constructor(props) {

    super(props);

    this.state = {

      posts: [],

      hasError: false

    };

  }

  // Step 6: Fetch API call

  loadPosts = () => {

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

      .then((res) => res.json())

      .then((data) => {

        this.setState({ posts: data });

      })

      .catch((error) => {

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

        this.setState({ hasError: true });

      });

  };

  // Step 7: componentDidMount()

  componentDidMount() {

    this.loadPosts();

  }

  // Step 9: componentDidCatch()

  componentDidCatch(error, info) {

    alert("An error occurred: " + error.message);

    this.setState({ hasError: true });

  }

  // Step 8: Render

  render() {

    if (this.state.hasError) {

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

    }

    return (

      <div>

        <h2>Blog Posts</h2>

        {this.state.posts.slice(0, 10).map((post) => (

          <Post key={post.id} title={post.title} body={post.body} />

        ))}

      </div>

    );

  }

}

export default Posts;

### Post.js

import React from "react";

function Post({ title, body }) {

  return (

    <div style={{ border: "1px solid gray", padding: "10px", marginBottom: "10px" }}>

      <h3>{title}</h3>

      <p>{body}</p>

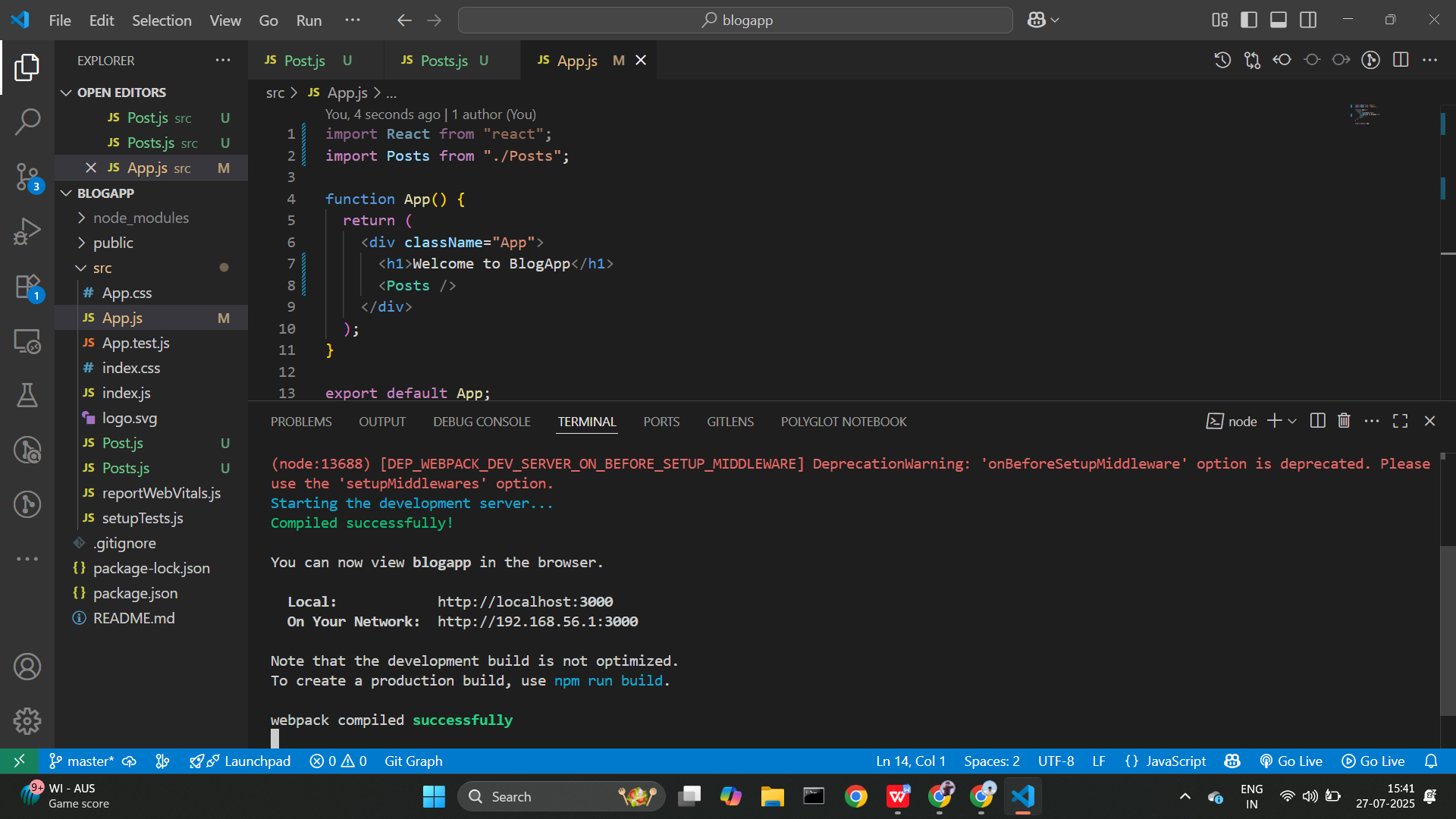
    </div>

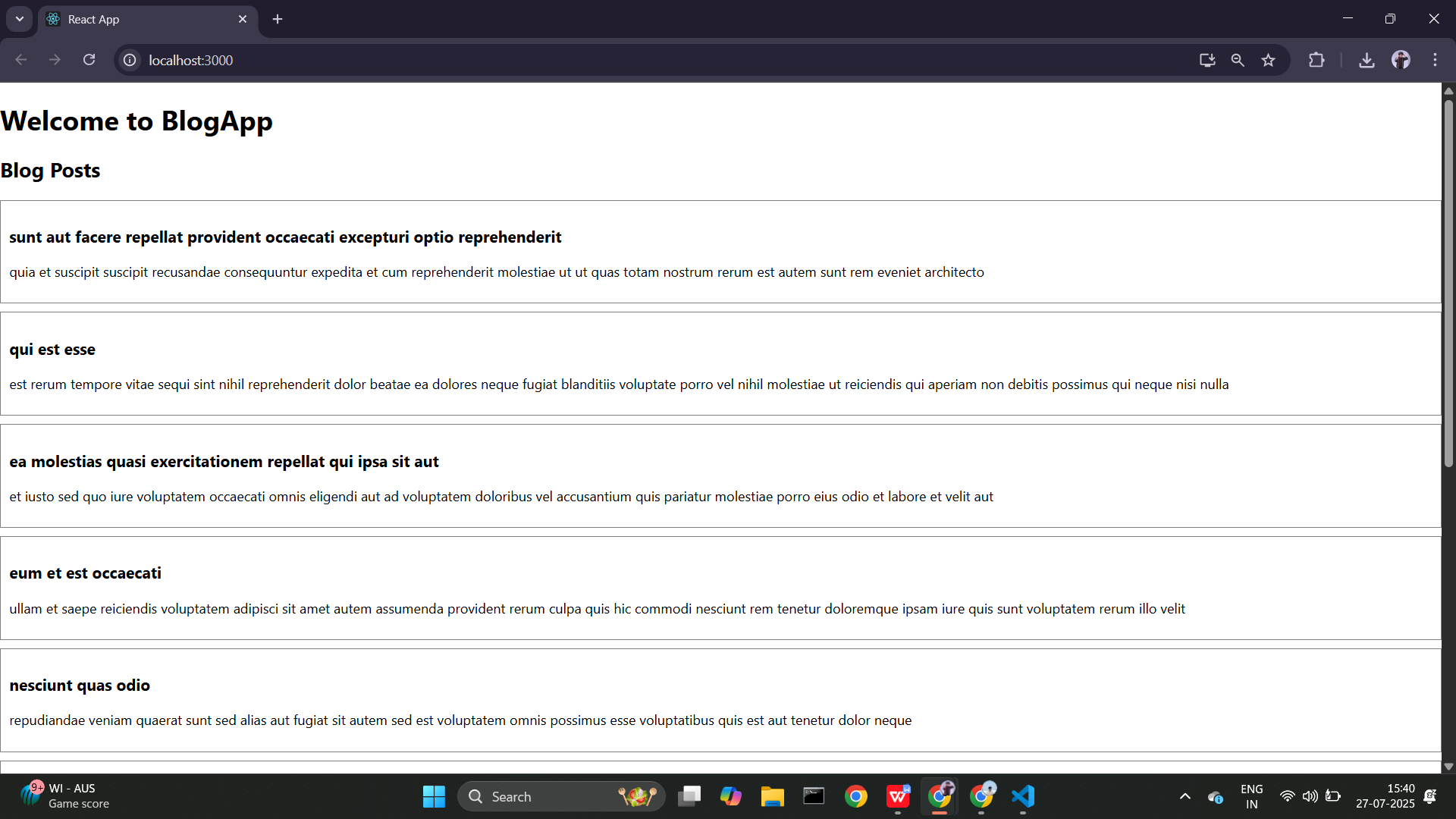
  );

}

export default Post;

## Screenshot





## Conclusion

This lab gave me practical experience using React class components and lifecycle hooks. I learned how to fetch and render data dynamically and handle errors gracefully.