**React HOL 4 Submission**

**Student Details**

**Name**: *Divyansh Tiwari*

**SuperSet ID:** 6394658

**Assignment Title**: *ReactJS HOL 4 - Component Lifecycle Hooks*

**1. Objectives of the Lab**

In this hands-on lab, I explored:

* The **need for React component lifecycle** methods.
* Usage of **componentDidMount()** to load data after rendering.
* Usage of **componentDidCatch()** to catch and handle errors gracefully.
* Rendering dynamic data from an API.

**2. Steps Performed**

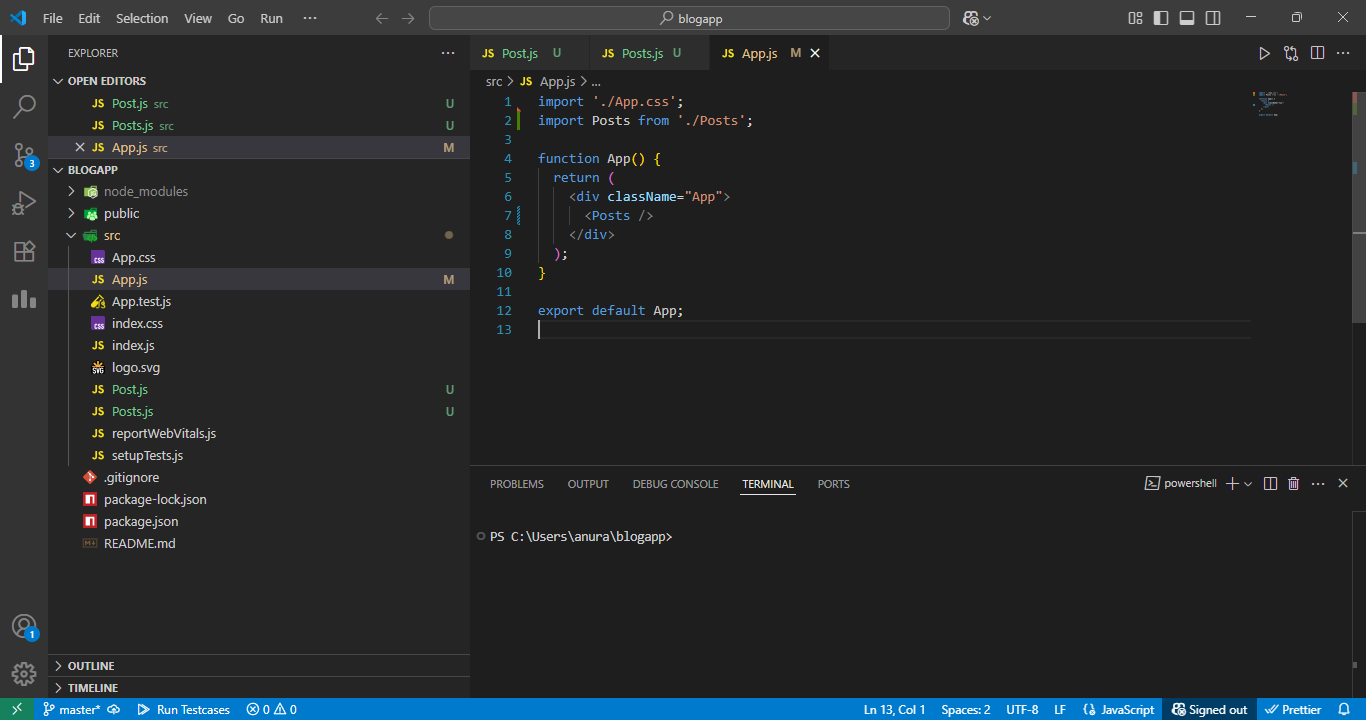
**Step 1: Create a New React Project**

*npx create-react-app blogapp*

**Step 2: Open Project in VS Code**

*cd blogapp*

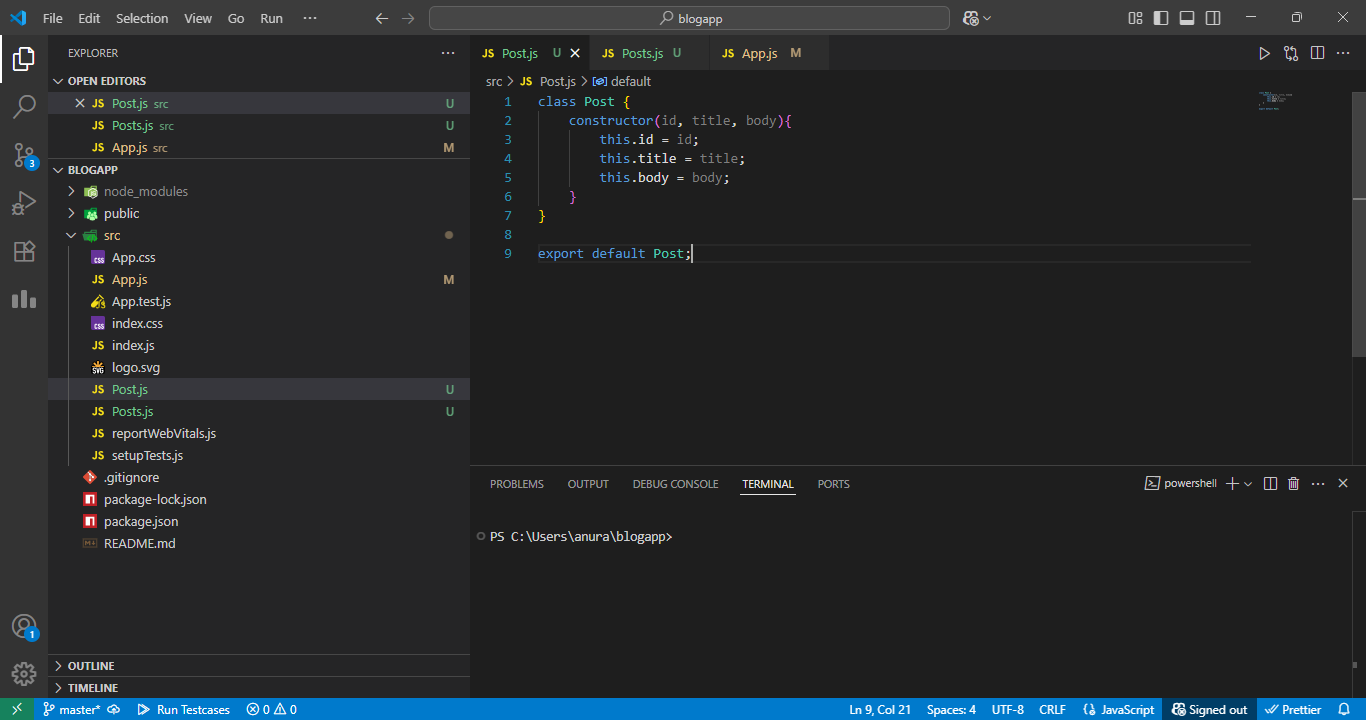
*code .*



*Project opened in VS Code*

**Step 3: Create Post.js**

Path: src/Post.js



*Code for Post component in VS Code*

**Step 4: Create Posts.js with Lifecycle Methods**

Path: src/Posts.js

Importing React and Component base class from the React library

*import React, { Component } from 'react';*

Importing the Post child component to display individual post data

*import Post from './Post';*

Defining a class component named 'Posts'

class Posts extends Component {

Constructor to initialize component state

*constructor(props) {*

*super(props);*

*this.state = {*

*posts: [],*

*hasError: false*

*};*

*}*

Custom method to fetch posts using Fetch API

*loadPosts() {*

*fetch('https://jsonplaceholder.typicode.com/posts') // API call to fetch dummy post data*

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

*.then(data => this.setState({ posts: data*

*.catch(error => {*

*console.error('Error loading posts:', error);*

*this.setState({ hasError: true });*

*});*

*}*

Lifecycle method: called after the component is mounted

*componentDidMount() {*

*this.loadPosts(); // Automatically fetch posts after component loads*

*}*

Lifecycle method: catches errors in child components during rendering

*componentDidCatch(error, info) {*

*alert("An error occurred while rendering posts.");*

*console.error("Caught error:", error, info);*

*this.setState({ hasError: true });*

*}*

Renders the component UI

*render() {*

*if (this.state.hasError) {*

*return <h2>Something went wrong.</h2>;*

*}*

If no error, display the list of posts

*return (*

*<div>*

*<h2>Blog Posts</h2>*

*{*

*this.state.posts.slice(0, 5).map(post => (*

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

*))*

*}*

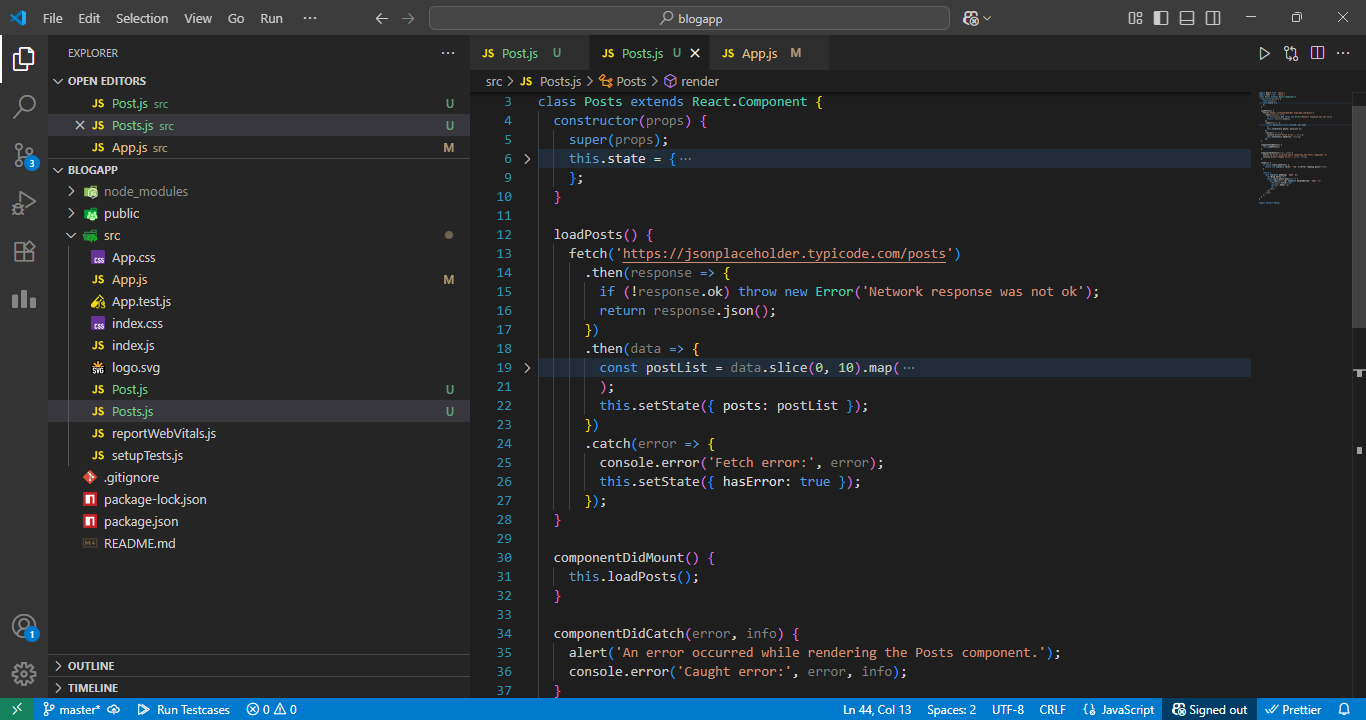
*</div>*

*);*

*}*

*}*

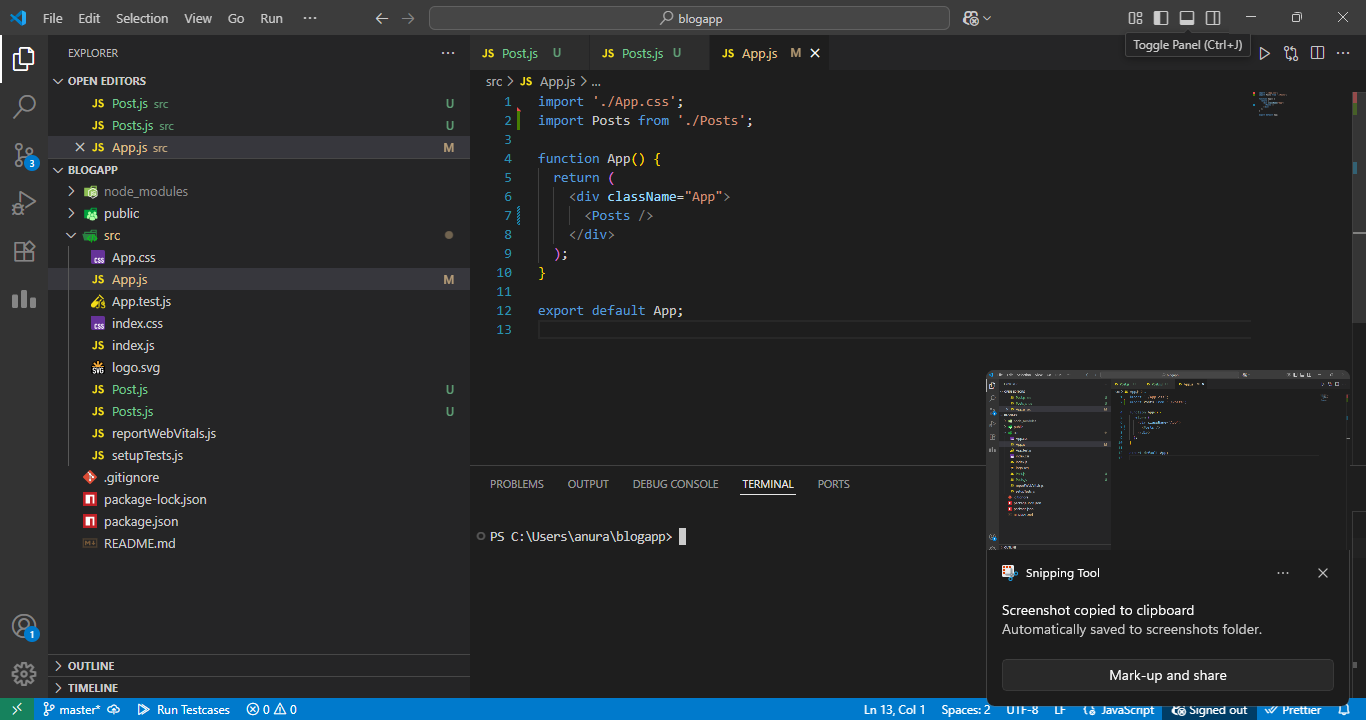
export default Posts;



* *Full Posts.js implementation including lifecycle hooks*

**Step 5: Modify App.js to Include Posts**

Path: src/App.js

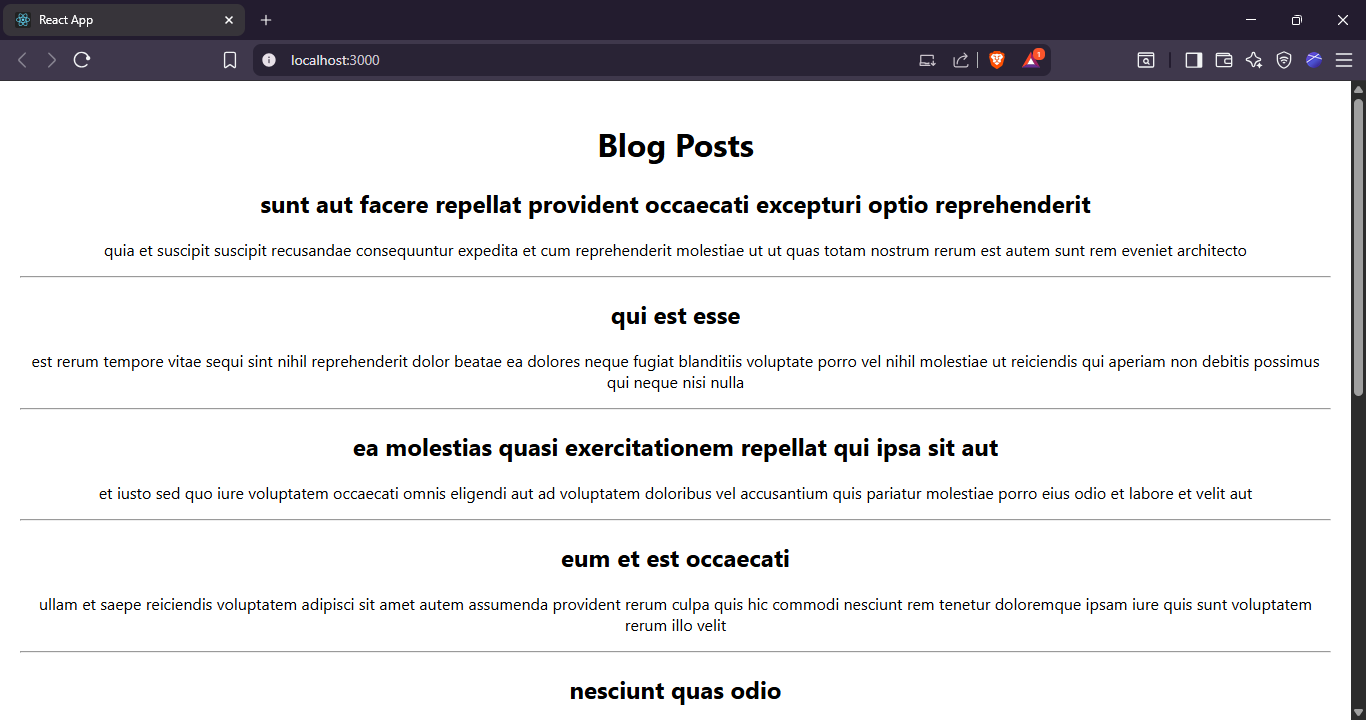


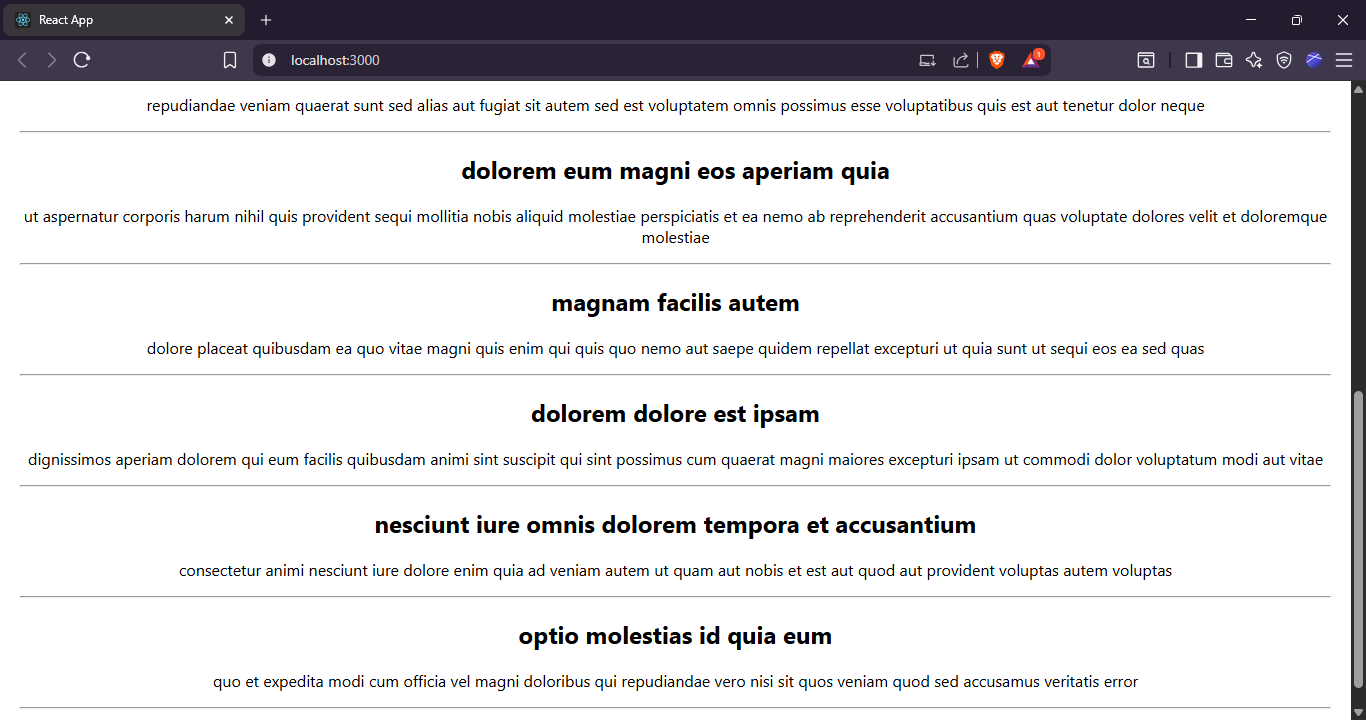
*App.js rendering Posts component*

**Step 6: Run the Application**

*npm start*

* Navigate to http://localhost:3000 in the browser





* *Browser showing list of blog post titles and content*