Internship Report – Frontend Dev

Week 7: React.js Advanced

Name: Zainab
Father Name: Assad Qayyum
Date: 8th Aug, 2025
Internship Domain: Front-end Intern
Task: Mini Project - The Blog Viewer using API

Task Overview: (Day5)

The task for today was to build a Mini Project in React.js that applies all the concepts learned so far in Week 6 (React Basics) and Week 7 (React Advanced). I have selected the project **Blog Viewer** that fetches and displays real blog posts from an API.

Objective:

The objective was:

- To implement concepts of react.js such as routing, API calls etc.
- To understand and use real APIs for fetching live data.
- To practice component-based architecture and state management in React.
- To enhance the project's styling and UI for a better user experience.

Mini-project: Blog Viewer using API

The Blog Viewer is a simple React.js application that fetches real blog posts from the Sling Academy API and displays them in a user-friendly interface.

The home page lists all available blog posts, showing the title, an image (if available), and a short excerpt of the content. Each post has a "Read More" link that navigates to a detailed view page showing the full content of the blog. The project uses React Router for navigation, a custom useFetch hook for API calls, and CSS for styling the interface.

All the Concepts Used

1. React Router

Used for navigating between pages:

- / → Displays the list of blog posts.
- /post/:id → Displays details of a selected post.

2. useEffect for API Calls

Used to fetch real blog data from the Sling Academy API when the component mounts.

3. Custom Hook (useFetch)

Created a reusable hook to fetch data from any given API endpoint and handle loading/error states.

4. React Project Folder Structure

Organized into folders:

- components/ for UI components like PostList and PostDetail.
- hooks/ for reusable logic (useFetch.js).

5. Conditional Styling

Used CSS classes to make posts and details look clean and responsive.

Future Improvements:

While the current version of the Blog Viewer is functional and visually appealing, there are several enhancements that could improve user experience and functionality in future iterations:

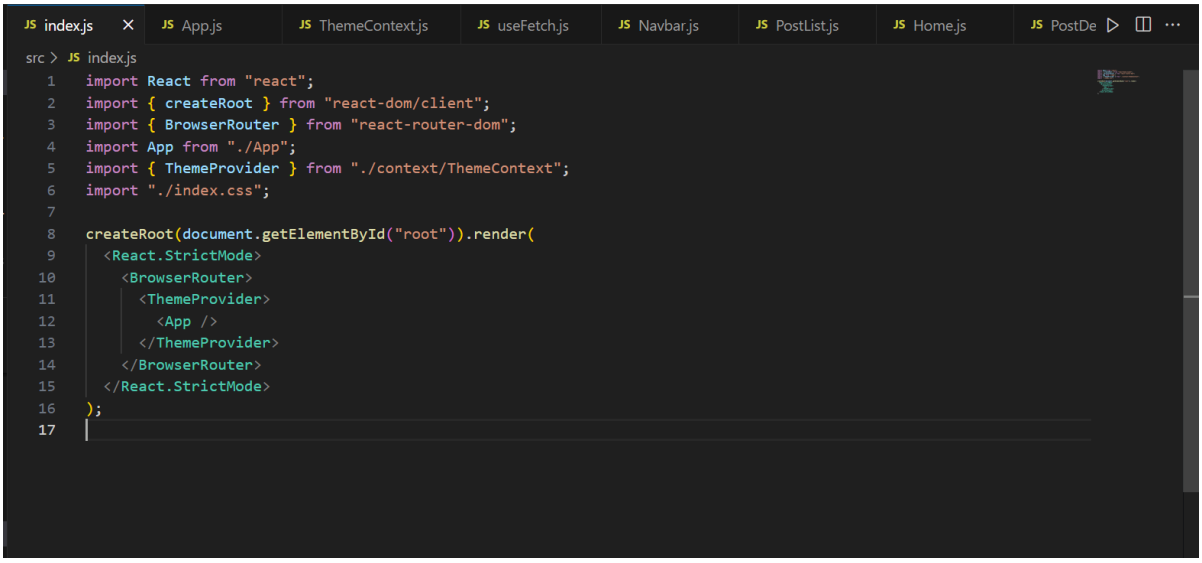
- **Search Bar:** Allow users to search for blog posts by title or keyword to quickly find relevant articles.
- **Pagination or Infinite Scroll:** Instead of loading all posts at once, implement pagination or infinite scroll for better performance and smoother browsing.
- **Dark Mode Toggle:** Add a theme switcher to let users choose between light and dark mode for better accessibility.
- **Category Filters:** Enable filtering of posts based on categories or tags for more organized content navigation.

Practice Code:

This Blog Viewer is a simple React.js app that fetches blog posts from a public API and displays them in a clean, styled interface.

- Fetches and displays blog posts from a public API (JSONPlaceholder).
- Uses React Router for multi-page navigation.
- Implements Context API for light/dark theme toggle.
- Includes a custom hook (useFetch) for API calls.
- Applies conditional styling for different themes.

Index.js:



App.js:

```
src > App.js > ...
1  import React from "react";
2  import { Routes, Route } from "react-router-dom";
3  import Navbar from "./components/Navbar";
4  import Home from "./pages/Home";
5  import PostDetail from "./pages/PostDetail";
6
7  function App() {
8    return (
9      <div>
10       <Navbar />
11       <Routes>
12         <Route path="/" element={<Home />} />
13         <Route path="/posts/:id" element={<PostDetail />} />
14       </Routes>
15     </div>
16   );
17 }
18
19 export default App;
20
```

ThemeContext.js:

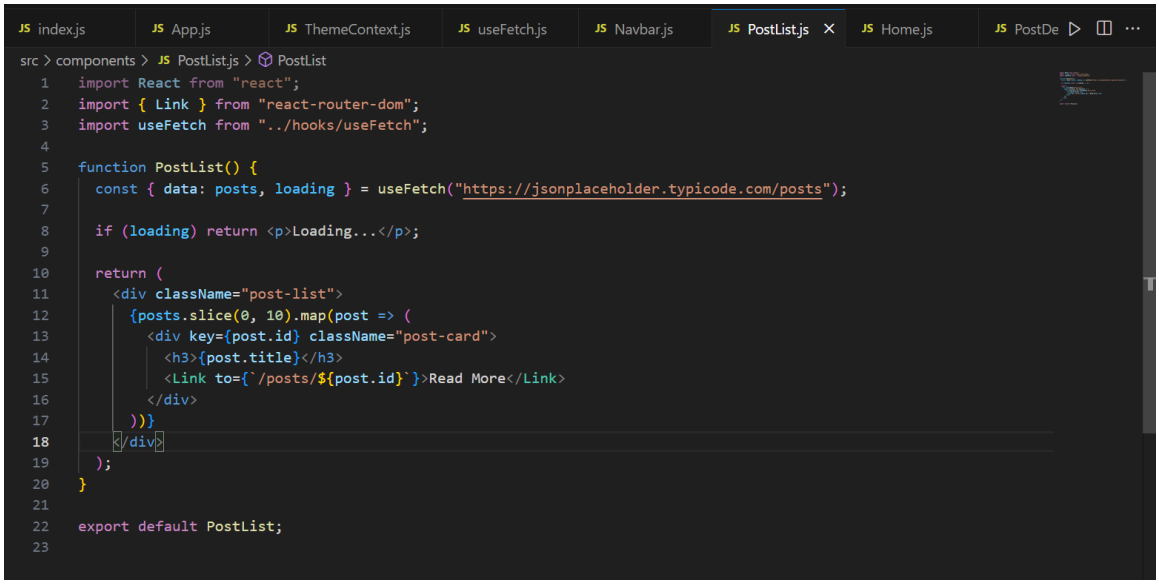
```
src > context > ThemeContext.js > ...
1  import React, { createContext, useState } from "react";
2
3  export const ThemeContext = createContext();
4
5  export function ThemeProvider({ children }) {
6    const [theme, setTheme] = useState("light");
7
8    const toggleTheme = () => {
9      setTheme(prev => prev === "light" ? "dark" : "light");
10   };
11
12   return (
13     <ThemeContext.Provider value={{ theme, toggleTheme }}>
14       <div className={theme === "light" ? "light-theme" : "dark-theme"}>
15         {children}
16       </div>
17     </ThemeContext.Provider>
18   );
19 }
20
```

useFetch.js:

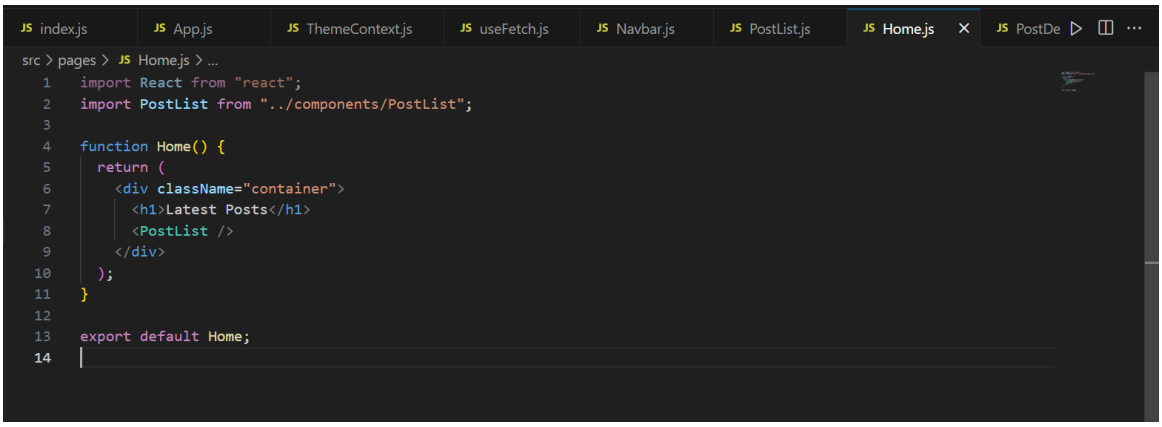
```
src > hooks > useFetch.js > ...
1  import { useState, useEffect } from "react";
2
3  export default function useFetch(url) {
4    const [data, setData] = useState(null);
5    const [loading, setLoading] = useState(true);
6
7    useEffect(() => {
8      fetch(url)
9        .then(res => res.json())
10       .then(data => {
11         setData(data);
12         setLoading(false);
13       })
14       .catch(() => setLoading(false));
15   }, [url]);
16
17   return { data, loading };
18 }
19
```

Navbar.js:

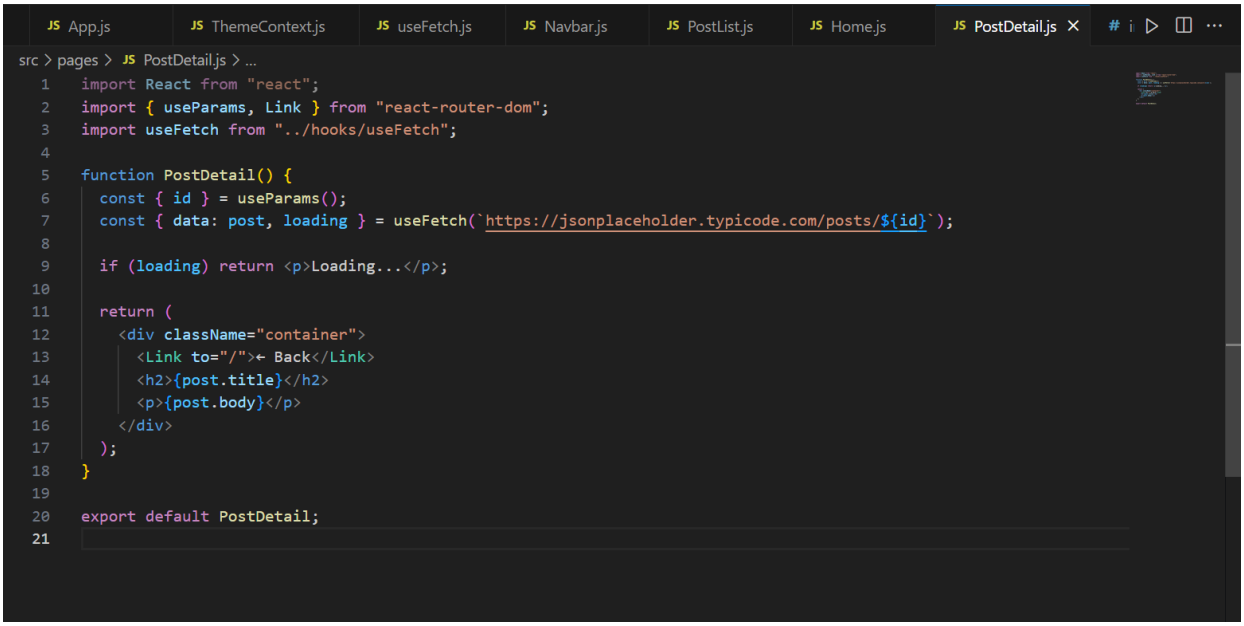
PostList.js:



Home.js:



PostDetail.js:

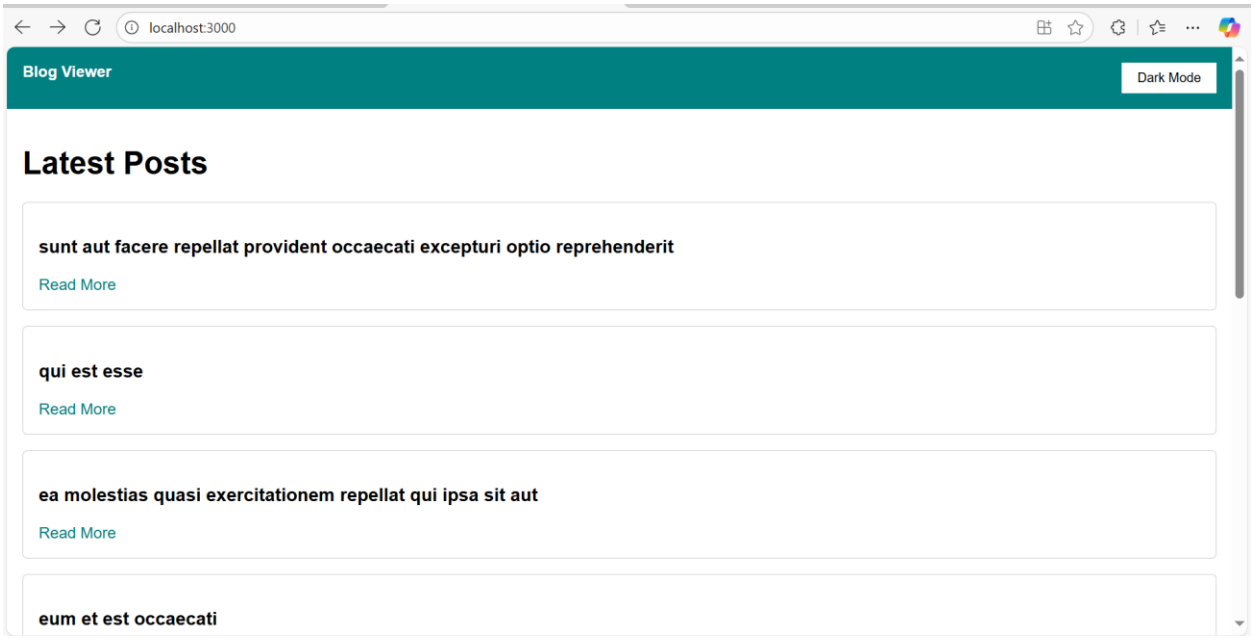


Index.css:

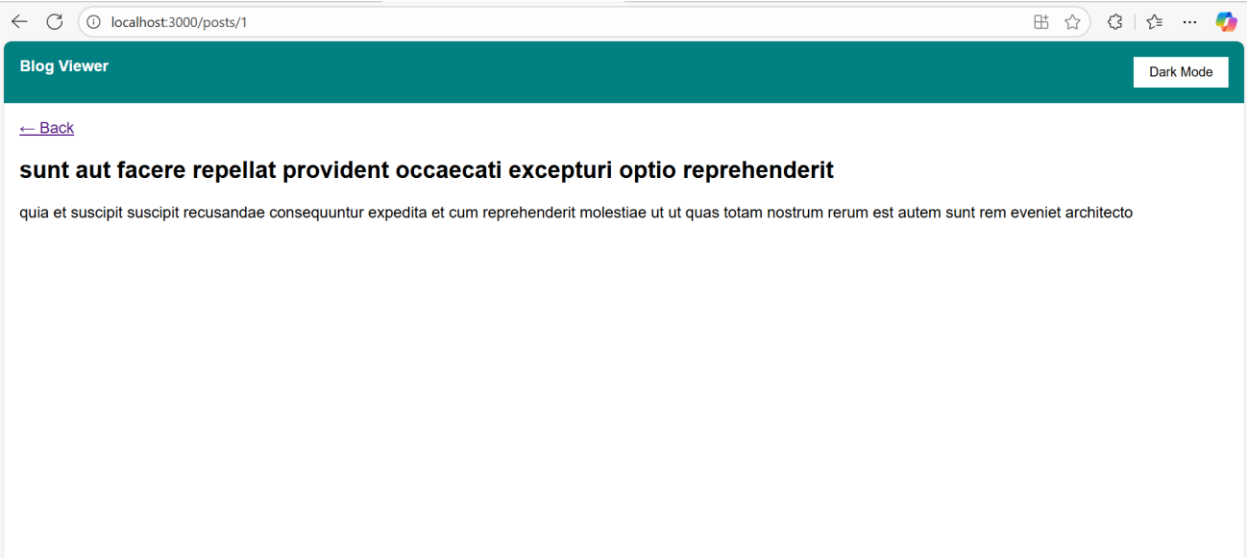
src > # index.css > .post-card

```css
button {
  background: white;
  border: none;
  padding: 0.5rem 1rem;
  cursor: pointer;
}

button:hover {
  opacity: 0.8;
}

.container {
  padding: 1rem;
}

.post-list {
  display: grid;
  gap: 1rem;
}

.post-card {
  padding: 1rem;
  border: 1px solid #ccc;
  border-radius: 5px;
}
.post-card a {
  color: teal;
  text-decoration: none;
}
```

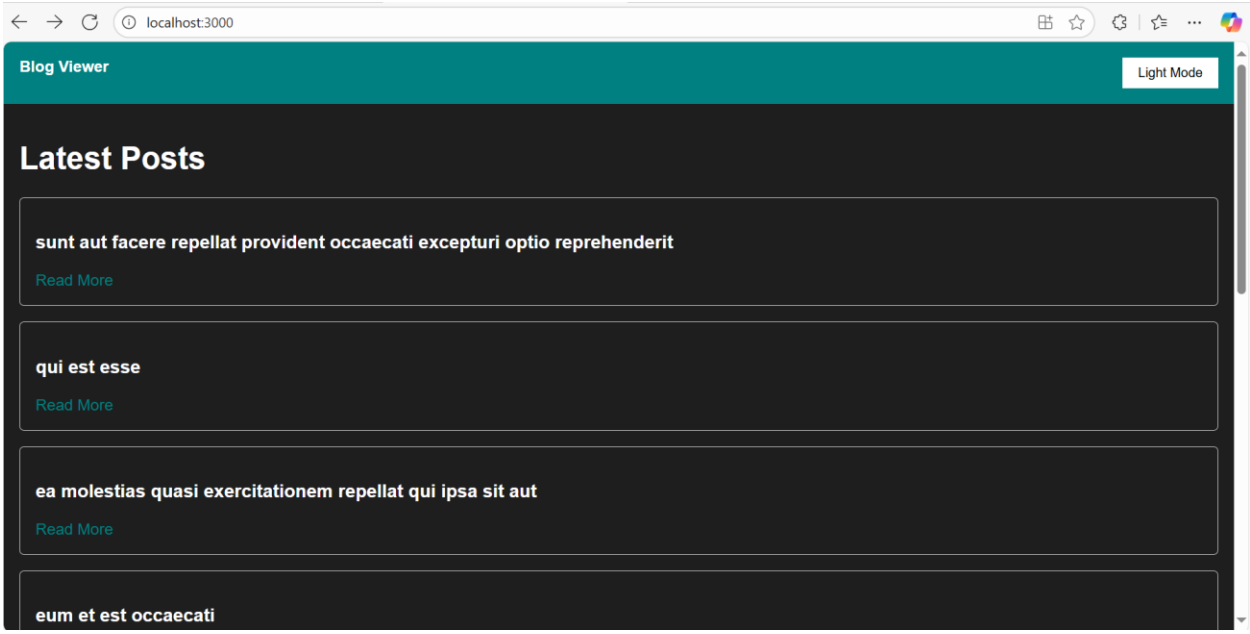
Ln 54, Col 2 Spaces: 2 UTF-8 LF {} CSS Go Live



Read more for post1:



Dark mode:



Conclusion:

Today, **The Blog Viewer using API** project successfully demonstrated the ability to work with React.js advanced concepts while integrating real-world data through an API. This mini-project enhanced understanding of routing, hooks, and API integration in React.