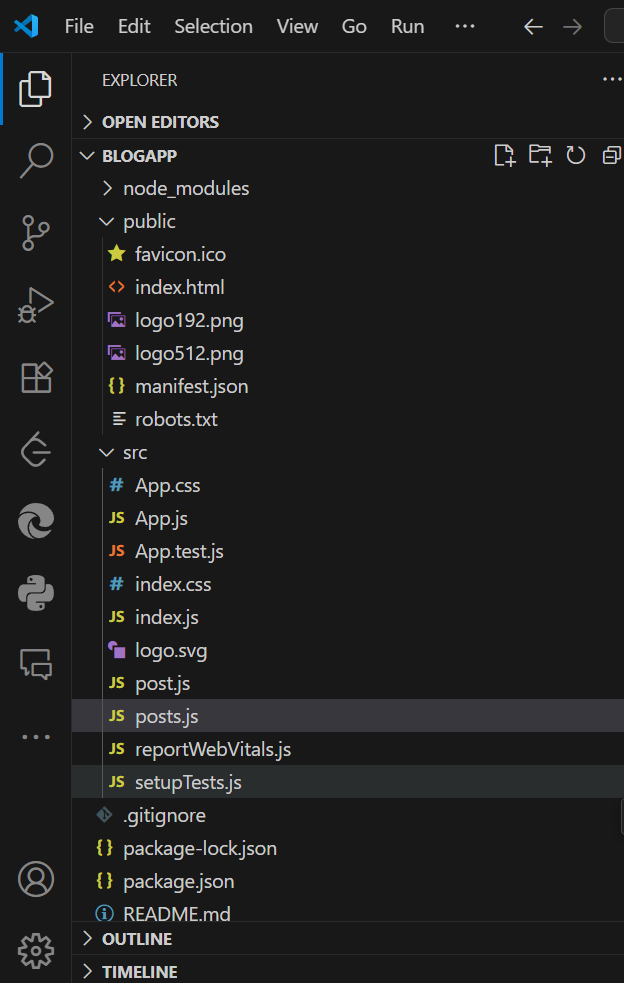
**Week6 React HandsOn**

**4. ReactJS-HOL**

1. Create a new react application using *create-react-app* tool with the name as “blogapp”
2. Open the application using VS Code
3. Create a new file named as **Post.js** in **src folder** with following properties
4. Create a new class based component named as **Posts** inside **Posts.js** file
5. Initialize the component with a list of Post in state of the component using the constructor
6. Create a new method in component with the name as **loadPosts()** which will be responsible for using Fetch API and assign it to the component state created earlier. To get the posts use the url (<https://jsonplaceholder.typicode.com/posts>)
7. Implement the **componentDidMount()** hook to make calls to **loadPosts()** which will fetch the posts
8. Implement the **render()** which will display the title and post of posts in html page using heading and paragraphs respectively.
9. Define a **componentDidCatch()** method which will be responsible for displaying any error happing in the component as alert messages.
10. Add the Posts component to App component.
11. Build and Run the application using *npm start* command.

Folder Structure



**App.js**

import React from 'react';

import Posts from './posts';

function App() {

  return (

    <div className="App"

      style={{

        display: 'flex',

        flexDirection: 'column',

        alignItems: 'center',

      }}

    >

      <h1 style={{ textAlign: 'center'}}>Blog App</h1>

      <Posts />

    </div>

  );

}

export default App;

**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: [],

      hasError: false

    };

  }

  loadPosts = () => {

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

      .then(res => res.json())

      .then(data => {

        const postList = data.map(

          item => new Post(item.id, item.title, item.body)

        );

        this.setState({ posts: postList });

      })

      .catch(error => {

        console.error("Failed to load posts:", error);

        this.setState({ hasError: true });

      });

  };

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

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

    console.error("Error info:", info);

    this.setState({ hasError: true });

  }

  render() {

    if (this.state.hasError) {

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

    }

    return (

      <div style={{ padding: '0px 50px', fontFamily: 'Arial' }}>

        <h2 style={{ marginBottom: '20px' }}>Posts</h2>

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

          <div

            key={post.id}

            style={{

                border: '1px solid #ddd',

                padding: '5px',

                marginBottom: '10px',

                borderRadius: '8px',

                backgroundColor: '#f9f9f9'

            }}

          >

            <h3 style={{ color: '#3780c9ff' }}>

              #{post.id} - {post.title}

            </h3>

            <p style={{ color: '#137c4bff' }}>{post.body}</p>

          </div>

        ))}

      </div>

    );

  }

}

export default Posts;

**Output:**

PS C:\Users\umaya\Desktop\-Cognizant-Nurture-4.0-practice\Week6\blogapp> npm start

> blogapp@0.1.0 start

> react-scripts start

(node:20040) [DEP\_WEBPACK\_DEV\_SERVER\_ON\_AFTER\_SETUP\_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.

(Use `node --trace-deprecation ...` to show where the warning was created)

(node:20040) [DEP\_WEBPACK\_DEV\_SERVER\_ON\_BEFORE\_SETUP\_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.

Starting the development server...

Compiled successfully!

You can now view blogapp in the browser.

Local: http://localhost:3000

On Your Network: http://192.168.140.50:3000

Note that the development build is not optimized.

To create a production build, use npm run build.

webpack compiled successfully

Compiling...

Compiled successfully!

You can now view blogapp in the browser.

Local: http://localhost:3000

On Your Network: http://192.168.140.50:3000

Note that the development build is not optimized.

To create a production build, use npm run build.

Compiled successfully!

You can now view blogapp in the browser.

Local: http://localhost:3000

On Your Network: http://192.168.140.50:3000

Note that the development build is not optimized.

To create a production build, use npm run build.

webpack compiled successfully

