ADS Assignment 5

Name : Sourabh Yashwant Chaugule

Prn : 21510037

Class : TY CSE

Aim : To perform performance tuning on Assignments 3 and 4

1. Efficiency is paramount in web applications, where users value their time. React, a favored framework for rendering performance, is trusted by industry giants like LinkedIn, Dropbox, CSDN, Baidu, among others.
2. Despite React's prowess, unnecessary rendering of components can lead to performance issues. Developers can employ astute React performance optimization methods to mitigate such bottlenecks.
3. Discover refined techniques to streamline React applications and combat resource-intensive DOM operations:
4. List Virtualization for Swift Rendering: Long lists in React apps often incur performance drawbacks, overwhelming the DOM and causing UI lag. Employ list virtualization, using libraries like react-window or react-virtualized, to render only visible items, enhancing app responsiveness.
5. Optimizing List Rendering with Key Assignment: Dynamically rendering lists demands prudent key assignment to elements. Incorrect key assignments can trigger unnecessary re-renders. Ensure each list component has a unique key value to optimize rendering performance.
6. Enhance Image Loading Efficiency: Image-heavy React applications can suffer performance degradation due to simultaneous rendering of all images. Leverage lazy loading techniques, such as those provided by react-lazyload or react-lazy-load-image-component, to defer image loading until they are within the viewport, minimizing unnecessary DOM manipulation.
7. Embrace Functional Components: Simplify and expedite React application development by embracing functional components. Smaller components are not only easier to comprehend but also facilitate code reuse, testing, and maintenance.
8. Mastering 'this' Binding: While functional components obviate the need for 'this' binding, it's essential to understand binding techniques in ES6 class components. Explore various binding methods like render binding, arrow functions, constructor binding, or class property binding for optimal performance.
9. Optimal State Management with setState: Instead of directly updating state with an object, utilize setState with a function. This approach ensures correct state updates by considering the previous state, thereby enhancing React application performance.

10. Type-Safe Props Validation with Prop-Types: Implement Prop-Types to enforce type validation on component props, ensuring code reliability and early detection of potential bugs during development.
11. Efficient JavaScript Bundling: Trim redundant code and eliminate duplicates in JavaScript bundles to enhance React app performance significantly. Conduct thorough analysis to identify and remove unnecessary code bloat.
12. Strategic Server-Side Rendering (SSR): Deliberately opt for SSR based on actual SEO requirements, as it can impose a considerable server load. Utilize frameworks like Next.js for efficient SSR implementation, optimizing both performance and development speed.
13. React Redux Optimization Strategies: Alleviate performance overhead in React Redux applications by leveraging techniques like Reselect for efficient data memoization and Immutable.js for immutable data structures, mitigating unnecessary state updates and enhancing overall application responsiveness.
14. Boosting Performance with React Memoization: Employ React Memoization to optimize performance further. Memoization enhances component rendering efficiency by caching previous results, thereby reducing unnecessary re-renders and boosting overall React app performance.

Some of Doing these are :

1. Original Code:

```
this.setState({count: this.state.count + 1});
```

Recommended Approach:

```
this.setState((prevState, props) => {
  return {count: prevState.count + 1};
});
```

In this example, we're incrementing the count state value by 1. Instead of directly updating the state with an object, we're using a function in setState that receives the previous state as its first argument. This ensures that the state is updated correctly, especially in scenarios where multiple state updates are queued.

2. Utilizing Prop-Types:

Original Code:

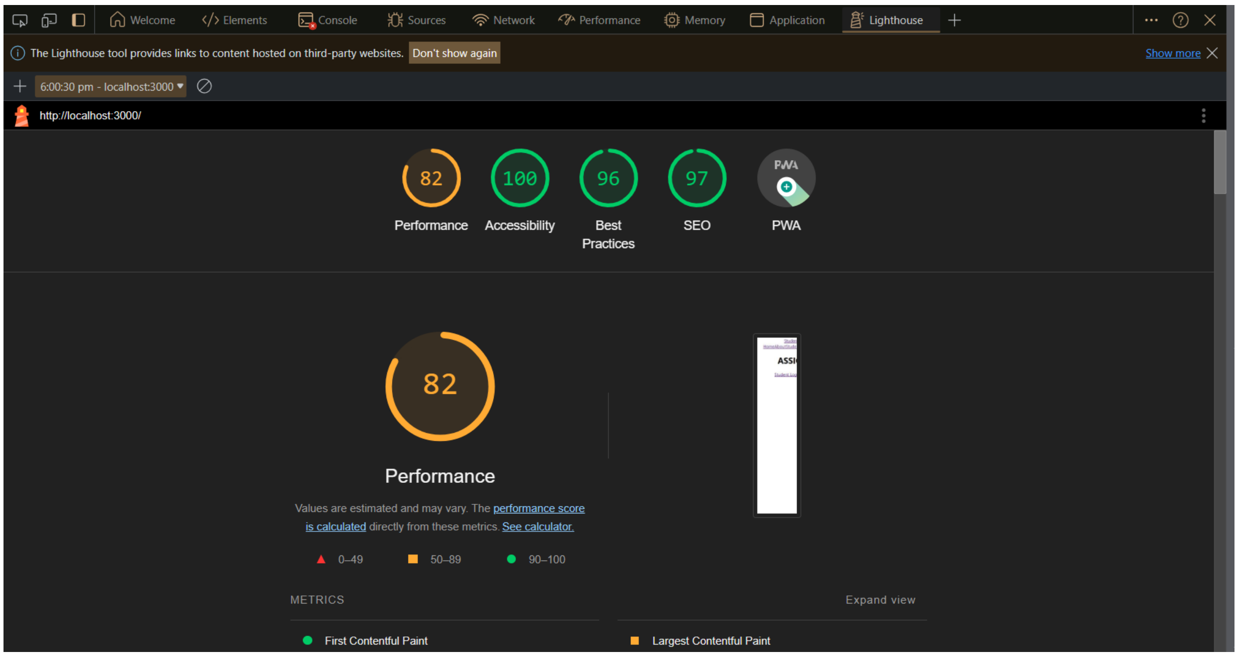
```
import React, { Component } from 'react';
class Greeting extends Component {
render() {
  return <h1>Hello, {this.props.name}</h1>;
  }
}
Greeting.propTypes = {
  name: 'John'
};
export default Greeting;
```

Recommended Approach:

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
class Greeting extends Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
Greeting.propTypes = {
  name: PropTypes.string.isRequired
};
export default Greeting;
```

In this example, we have a Greeting component that receives a name prop. Instead of manually specifying the prop type within the component code, we're utilizing PropTypes from the prop-types library to define the prop type as a required string. This ensures type safety and provides early detection of potential bugs during development.

Performance :



The Lighthouse tool provides links to content hosted on third-party websites.

http://localhost:3000/

82	100	96	97	
Performance	Accessibility	Best Practices	SEO	PWA

82

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS

● First Contentful Paint

■ Largest Contentful Paint