Web Development (React.js) (shafia odho)

Task: Explore and document the following:

1. **React Performance Optimization Techniques**

– React.memo, useCallback, useMemo
– Code-splitting and lazy loading

2. **Security in React Apps**

– XSS prevention, content sanitization
– Storing tokens securely (localStorage vs. cookies)

3. **Component Design Patterns**

– Presentational vs Container Components
– Compound components, render props, hooks-based design

1. React Performance Optimization Techniques

React.memo: Tells React, "Only re-render this component if the props change." Super handy for components that don't need to update all the time.
useCallback: Remembers your functions so they're not recreated every render—great when passing functions to child components.
useMemo: Saves the result of expensive calculations so React doesn't redo them every time.
Lazy Loading + Code-Splitting: Only load parts of your app when needed, making the initial load faster (think Netflix loading a movie only when you click it).

2. Security in React Apps

Avoid XSS (Cross-site Scripting): Never trust user input. If you must display raw HTML, always sanitize it using tools like DOMPurify.
Where to Store Tokens?
localStorage: Easy to use, but risky—can be stolen by XSS.
Cookies (HttpOnly): Safer for storing login tokens because JavaScript can't access them.

3. Component Design Patterns

Presentational vs Container:
Presentational: Just handles the look (like a mannequin).
Container: Handles logic and data (like the brains behind the mannequin).
Compound Components: A group of components that work together and share state—for example, a <Tabs> component with multiple <Tab> children.
Render Props: Instead of passing data, you pass a function as a prop to control rendering—flexible and powerful.
Hooks-Based Design: Move your logic into custom hooks to reuse it easily across components (cleaner, smarter code)