

**Task 3: Written Comparison (15 points) Write a short report (250–350 words) that:**

**1. Describes how React acts as the “view layer”: data (state) → component → UI (Ch. 1).**

React functions exclusively as the view layer in application architecture, as explained in Chapter 1's *"React is just the view layer"* section. This means React's sole responsibility is rendering user interfaces based on data and managing how those interfaces update when data changes. Unlike full-featured frameworks, React does not provide built-in solutions for routing, HTTP requests, or state management beyond component-level state—developers must integrate separate libraries for these concerns.

In our Counter component, this view layer principle is clearly demonstrated. The component maintains state (the counter value) using the `useState` hook, which represents the data layer. The component's JSX return statement describes the UI layer based on that data: it displays the current count value and provides buttons for user interaction. This follows the fundamental equation discussed in Chapter 1:  $UI = f(state)$ . The component is purely a function that transforms state into a visual representation.

**2. Explains how clicking the buttons causes a re-render and updates the DOM + Mentions the virtual DOM and diffing/patching (Ch. 1, “Performance matters”) at a high level.**

When a user clicks the Increment or Decrement button, the `onClick` handler calls `setCount`, updating the component's state. This state change triggers React's re-rendering process. React calls the Counter function component again, computing a new virtual DOM representation with the updated count value. As described in Chapter 1's *"Performance matters"* section, React then performs a diffing operation, comparing the new virtual DOM against the previous version to identify what changed—in this case, only the text content displaying the count number.

React's reconciliation algorithm then patches the real DOM, updating only the specific text node that displays the count. This selective updating is far more efficient than replacing the entire component in the DOM. The virtual DOM abstraction, combined with React's intelligent diffing and patching algorithms, ensures that even complex UIs with frequent updates remain performant. This architectural approach, outlined in "Setting up a new React project", allows developers to write declarative code while React handles the imperative DOM manipulation behind the scenes.

**References:**

Sakhniuk, M. B., & Boduch, A. (2024). *React and React Native - Fifth Edition: Build Cross-platform JavaScript and TypeScript Apps for the Web, Desktop, and Mobile*.