NAME : PRAGHAADEESH R

ROLL NO : CH.EN.U4AIE21035

Case Study 1: Real-Time Stock Market Dashboard

1. How would you implement a WebSocket connection in a React component for real-time data fetching?

To fetch data in real-time using WebSockets in a React component, a WebSocket connection can be established inside a useEffect hook. This ensures the connection is initiated when the component mounts and is closed when it unmounts. Keeping the connection open while the component is active allows the component to receive data continuously, which can then be used to update the state and render it dynamically on the dashboard.

2. Describe how you would create a responsive table to display stock prices.

To create a responsive table, CSS or a library like react-bootstrap would be employed to maintain a visually appealing layout across all devices. Flexbox or Grid layouts could be used to organize the table structure effectively. The table would present stock data in rows and columns, adapting its layout to different screen sizes. On smaller devices, columns might collapse or change into a more compact format to ensure easy readability.

3. How can you implement a search bar to filter stocks based on the user's input?

A search bar can be created using an input field in React, with the user's input managed via React state. As the user types, the stock list in the table would be filtered dynamically by applying the input as a filter criterion (such as stock symbol or company name). This would involve generating a filtered version of the stock data array and rendering that filtered list in the table.

4. Explain the steps for handling connection errors and displaying appropriate messages to the user.

If the WebSocket connection encounters an error, a user-friendly error message should be displayed. This can be achieved by implementing error handling in the WebSocket setup and catching any issues related to the connection or data transfer. Upon detecting an error, the UI could be updated to display a message such as "Connection lost. Attempting to reconnect…" and implement an automatic reconnection mechanism after a specified delay.

5. How would you ensure the efficient updating of stock prices in the UI without performance degradation?

To maintain performance while updating stock prices, particularly when dealing with a large dataset, it's crucial to minimize unnecessary UI updates. While React's diffing algorithm inherently optimizes this, additional techniques such as React.memo can be utilized to prevent components from re-rendering unnecessarily. Additionally, employing request animation frames can help synchronize updates, ensuring smoother UI rendering and grouping updates together to reduce browser workload.