# Real Time Stock Market Dashboard

**Question 1: How would you implement a WebSocket connection in a React component for real-time**

While our current implementation doesn't use WebSockets, we could implement them as follows:

- Create a WebSocket connection in a useEffect hook when the component mounts.
- Set up event listeners for 'open', 'message', 'error', and 'close' events.
- Update the state (e.g., stock prices) when new data is received through the WebSocket.
- Close the WebSocket connection in the cleanup function of the useEffect hook.
- Instead of fetching data via REST API, we'd send subscription messages through the WebSocket for each stock symbol added.

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

Our implementation includes a responsive table:

- We use a standard HTML table structure for semantic markup.
- The table adjusts to the container width using percentage-based widths.
- We apply CSS to make the table responsive, potentially adding horizontal scroll for small screens.
- Each row represents a stock, with columns for symbol, price, change, change percentage, and volume.
- We use conditional rendering to apply different styles (e.g., colors for positive/negative changes).

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

Our current implementation has a simple input for adding stocks. To create a search/filter functionality:

- We could modify the input to filter the existing stocks as the user types.

- Implement a debounce function to avoid excessive filtering on each keystroke.
- Update the displayed stocks based on the filter, either client-side or by making API calls with the search term.
- Show matching results in a dropdown, allowing users to select and add stocks from there.

**Question 4: Explain the steps for handling connection errors and displaying appropriate messages**

While our current implementation doesn't explicitly handle connection errors, we could:

- Wrap API calls in try-catch blocks to catch and handle errors.
- Create a state variable to store error messages.
- Display error messages to the user when API calls fail or when adding invalid stock symbols.
- Implement a retry mechanism for failed API calls.
- Show loading indicators during API calls and disable relevant UI elements.

**Question 5: How would you ensure the efficient updating of stock prices in the UI without performa**

Our implementation includes several efficiency measures:

- We use React's state management to handle updates efficiently.
- The chart only re-renders when stocks are added or removed, not on every data update.
- We could implement memoization (React.memo, useMemo) to prevent unnecessary re-renders.
- For real-time updates, we'd batch updates and limit the update frequency to avoid excessive re-renders.
- We use efficient data structures (arrays and objects) for quick lookups and updates.
- The chart animation is optimized to smoothly handle multiple data series.