Task 2:

Answer:

1. Analyze the API Response: Confirm that, in the event of a change, the API is providing updated data.

To examine the API answers, use tools such as Postman or the developer tools included in the browser.

2. Examine React State and Props: See how your React components handle and store data. Make that the props or state that hold the chart data are updated correctly.

3. Check Component Lifecycle Methods: If you are retrieving data from a React component, confirm that you are utilizing React Hooks like useEffect or componentDidMount, componentDidUpdate, to update the state appropriately.

4. Verify Component Re-rendering: Make sure that modifications to the state or props cause the component that displays the charts to re-render. The charts will not refresh if the component isn't re-rendering.

5. Use State Management Libraries: To centralize and manage the application state, take into consideration utilizing state management libraries like Redux or Context API. This can support preserving a steady data flow across the application.

6. Examine the documentation in the Charting Library:

To ensure that you are updating the charts appropriately, go over the instructions provided by the charting library you are using. Certain charting libraries offer particular ways or choices for updating in real time.

7. Use WebSocket or Server-Sent Events (SSE): If real-time updates are essential, think about converting from standard HTTP polling to WebSocket or Server-Sent Events technology. With the use of these technologies, the server may instantly send changes to the client.

8. Managing Asynchronous actions: Make sure you are correctly managing asynchronous actions, such API requests. When requesting data, use async/await or promises, and ensure sure the component waits for the data to be obtained before rendering.

9. Debugging Tools: To debug your application, use the React DevTools plugin for Chrome or Firefox and the browser developer tools. Look for any warnings or error messages in the console.

Issues:

Asynchronous Nature: Handling asynchronous actions can occasionally result in unexpected behavior or race situations. Make sure that promises or async/await are being used correctly.

Component Lifecycle: It's important to know when and how components re-render. Ineffective updates might result from the misuse of lifecycle methods.

General Thought Process: Isolate the issue first. Verify whether the data is coming from the API.

Examine React components' data management practices and if state updates cause re-renders.

To make sure the charting library is being used properly, see its documentation.

If required, think about utilizing state management and more sophisticated real-time technology.