Task 2: You are working on a dashboard using ReactJS and Tailwind CSS. The dashboard displays data charts which are fetched from a REST API. However, the charts are not updating in real-time as new data arrives from the API.
Question: How would you troubleshoot and resolve this issue ensuring the charts update in real-time as the data changes? Explain your approach, potential challenges and the overall thought process.
Answer: By following a systematic method, I can troubleshoot and resolve the issue of chatrs not updating in real-time as new data arrives from the API in a React JS and Tailwind CSS dashboard. Here is a step-by-step guide that includes the general idea and possible challenges:

1. **Check API Response**: Examine the API response's structure to see how it delivers updates. Find out if any certain fields or endpoints contain new data.
2. **Update State**: Verify that when new data is received, the React state is updated appropriately. Make sure the state is updated with the most recent information by using the proper state management mechanism, such as the 'useState' hook.
3. **Use 'useEffect' Hook:** When the component mounts, use the 'useEffect' hook to retrieve data from the API and create a subscription for real-time updates. When new data is received, this entails updating the state within the 'useEffect' callback.
4. **Handle Error:** Provide error-handling procedures to deal with scenarios when the API call is unsuccessful or there are no real-time updates. For troubleshooting purposes, record errors to the console.
5. **Consider Performance:** Make sure that the frequency of updates is in line with the needs of the application and does not negatively impact user experience or place an unwarranted burden on the server.

Some of the challenges and considerations of the given problem are given below:

1. **Cross-origin Resource Sharing (CORS)**: Make sure that React JS application can send queries to the API, particularly if it hosted on a different domain.
2. **Backend Implementation**: Make sure that the backend API – whether via WebSocket or other methods – is built to accommodate updates in real-time.
3. **Handling Large Data Sets:** Consider how to efficiently handle and update large datasets without causing performance issues on the client side.
4. **Security Concern:** Implement secure communication practices, especially if dealing with sensitive data or real-time updates.

Understanding how the API offers real-time changes, choosing a suitable real-time communication method (WebSocket, polling), and making sure the React state is updated appropriately are crucial to fixing the problem. Crucial components of the troubleshooting process include testing, debugging, and knowing what the API is capable of. Furthermore, taking performance, security, and possible difficulties into account will help create a solid real-time update implementation.