To troubleshoot and enable real-time updates in the charts fetched from a REST API in a ReactJS and Tailwind CSS dashboard, here's a step-by-step approach:

Troubleshooting Steps:

1. **Check API Response:** Ensure the API delivers updated data correctly by making test requests and examining the data format.
2. **Chart Component Rendering:** Review the component responsible for rendering the charts. Confirm that it properly handles incoming data updates.
3. **Data Handling:** Verify how the data fetched from the API is managed within the React component. Ensure that updates to this data trigger re-renders of the charts.

Approach:

1. **WebSocket Integration:** Implement WebSocket or Server-Sent Events (SSE) to establish a real-time connection between the client (dashboard) and the server. This allows immediate data updates without repeated polling of the API.
2. **State Management:** Use React state or state management libraries (like Redux) to handle incoming data. Ensure that changes in this data trigger updates to the chart components.
3. **Update Mechanism:** Assess how often and by what means the component fetches new data from the API. Consider using event-driven mechanisms or WebSocket subscriptions to update data and subsequently refresh the charts.

Potential Challenges:

1. **Real-time Updates:** Ensuring that the charts reflect new data immediately as it arrives, without any delay or lag.
2. **Data Consistency:** Managing incoming real-time data without causing conflicts or irregularities in the charts.
3. **Performance Optimization:** Preventing excessive re-renders and optimizing the rendering process for efficient real-time updates without compromising performance.

Example Thought Process:

1. **Identify Data Update Triggers:** Determine what triggers the arrival of new data (e.g., server-side events, user actions). Design the update mechanism around these triggers.
2. **Architecture Review:** Evaluate how the dashboard component fetches and updates data. Ensure it can handle real-time data updates effectively.
3. **Testing and Iteration:** Test the system thoroughly, especially under different scenarios of data arrival. Iterate on the solution, making adjustments as necessary to ensure consistent real-time updates without compromising performance.