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:

Troubleshooting:

1. At first, I would have examined if the API calls are working and new data is being fetched accurately.
2. After that, I would use "**useState**" to confirm whether the fetched data is stored correctly in React state or not.
3. I would also try to use React tools to observe state changes and re-renders as well as to examine component props.

Potential Challenges:

- Performance Optimization: Excessive updates can impact performance. Techniques like debouncing or throttling are needed to be implemented to manage update frequency.

- Chart Library Limitations and Compatibility: Chart libraries with better real-time support ought to be used if needed. Moreover, the chart library should be compatible with the latest version of the dependencies.

Resolving the Issue:

- I would use **"setInterval"** and **"useEffect"** to fetch data and update state time to time. The process is stated below:
    - At first importing the **"useEffect"** and **"useState"**

```
import React, { useState, useEffect } from 'react';
```

    - Then creating an EventSource instance:

```
useEffect(() => {
  const source = new EventSource('SSE-endpoint');
  source. addEventListener('message', (event) => {
    const data = JSON.parse(event.data);
    setState(data); // Update state with received data
  };
```

- After that, handling the error:

```
// Error handling and cleanup:
  source. addEventListener('error', (error) => {
    console.error('SSE error:', error);
    source.close(); // Closing the connection on error
  };

  return () => source.close();
}, []);
```

- Finally Rendering the component with the fetched data

```
return (
    <div>
      {/* ---- */}
    </div>
  );
```

- I would also consider using "**WebSockets**" because they can maintain bidirectional communication between the client and the server. As a result, the server will be able to update the client data as soon as new data is available. The process of hoe the data is updated is decribed below:

  At first the Reactjs application sets up a WebSocket connection to the server. When new data becomes available, the server sends this data to the connected clients through the WebSocket connection. As the server pushes updates, the corresponding event listener on the client side of the application is triggered. Then inside the event listener, the React state is updated with the new data received from the server.

```
const socket = new WebSocket('Api-endpoint');

socket.addEventListener('message', (event) => {

  const newData = JSON.parse(event.data);   // Handle the
incoming data

  setData(newData);    // Update React state with the new data

});
```