Task2

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.

Solutions:

I have done a project regarding task 2 where users can update and preview data simultaneously but it is not real-time updating, which means if users update data at the backend it won't update automatically at the front end.

For this to solve I will use websocket with express js.

In my current task 2 project I simply have used express for REST API along with CRUD operations.

Backend
Using Express JS and Websocket(socket.io)

```
// server.js
const express = require('express');
const http = require('http');
const { Server } = require('socket.io');

const app = express();
const server = http.createServer(app);
const io = new Server(server);

io.on('connection', (socket) => {
    console.log('A user connected');
});
const PORT = process.env.PORT || 3001;
server.listen(PORT, () => {
    console.log(`Server running on http://localhost:${PORT}`);
});
```

On the above code, implemented the socket io for to use of the websocket, which will ensure our real-time communication between the backend to frontend

```
// server.js continued
app.get('/api/data', (req, res) => {
    const newData = [
    { month: "january", value: "70" },
    { month: "february", value: "65" },
    { month: "march", value: "80" },
    ... ... ...];

    // Broadcast the new data to all connected clients
    io.emit('updatedData', newData);
    res.json(newData);
});
```

Here the main data will broadcast through the websocket, which will broadcast whenever a new request is made to the `/api/data` .

Front-end

Using React JS and Websocket(socket.io-client)

```
// React component
import { useEffect } from 'react';
import io from 'socket.io-client';

const YourComponent = () => {
    useEffect(() => {
        // Connecting to the WebSocket server
        const socket = io('http://localhost:3001');

        // Listen for updates from the server
        socket.on('updatedData', (data) => {
            // Update your React component state with the new data
            console.log('Received updated data:', data);
        });

        return () => {
            // Disconnect the socket when the component unmounts
            socket.disconnect();
        };
    }, []);

    // Your component rendering and other logic
};

export default YourComponent;
```

When the WebSocket server broadcasts updates, the React component will be able to receive them and can update its state/components accordingly.

Thoughts

There are also multiple ways to update in real-time without using the websocket, one of them could be polling after a certain time or always polling for new data. This could be an easy solution but this could use too much memory and network call.

Using websocket, we are creating a broadcasting channel between the front-end and the backend, and through that broadcasting channel, the backend will notify the front-end about the data update and the front-end will update accordingly.

Challenges

I would like to mention two main challenges for websocket or this type of solution.

1. **Scalability:**

- Scaling WebSocket applications can be challenging, especially in scenarios where a large number of concurrent connections are required. So the implementation of a load balancer becomes imminent.

2. **Resource Consumption:**

- WebSocket connections consume server resources as I have already mentioned. Large numbers of concurrent connections may lead to extreme use of memory, CPU and network usage, which can impact the performance.