React Promise Error Fix

Dies ist die Kopie eines Gesprächs zwischen ChatGPT und einem anonymen Benutzer.

Unexpected Application Error!
Objects are not valid as a React child (found: [object Promise]). If you meant to render a collection of children, use an array instead.
Error: Objects are not valid as a React child (found: [object Promise]). If you meant to render a collection of children, use an array instead.
    at throwOnInvalidObjectType (https://localhost/node_modules/.vite/deps/chunk-PJEEZAML.js?v=7ac948e7:9934:17)
    at reconcileChildFibers2 (https://localhost/node_modules/.vite/deps/chunk-PJEEZAML.js?v=7ac948e7:10564:15)
    at reconcileChildren (https://localhost/node_modules/.vite/deps/chunk-PJEEZAML.js?v=7ac948e7:14290:37)
    at updateHostComponent (https://localhost/node_modules/.vite/deps/chunk-PJEEZAML.js?v=7ac948e7:14807:11)
    at beginWork (https://localhost/node_modules/.vite/deps/chunk-PJEEZAML.js?v=7ac948e7:15935:22)
    at beginWork$1 (https://localhost/node_modules/.vite/deps/chunk-PJEEZAML.js?v=7ac948e7:19753:22)
    at performUnitOfWork (https://localhost/node_modules/.vite/deps/chunk-PJEEZAML.js?v=7ac948e7:19198:20)
    at workLoopSync (https://localhost/node_modules/.vite/deps/chunk-PJEEZAML.js?v=7ac948e7:19137:13)
    at renderRootSync (https://localhost/node_modules/.vite/deps/chunk-PJEEZAML.js?v=7ac948e7:19116:15)
    at recoverFromConcurrentError (https://localhost/node_modules/.vite/deps/chunk-PJEEZAML.js?v=7ac948e7:18736:28)
💿 Hey developer 👋

You can provide a way better UX than this when your app throws errors by providing your own ErrorBoundary or errorElement prop on your route.

What does that mean?

This error occurs when you're trying to render a Promise directly as a React child, which is not allowed. React can only render strings, numbers, arrays, or other React components, but not unresolved Promises.

In your case, it seems like you're probably trying to fetch data or perform an asynchronous operation (like calling an API or loading something), but you're directly passing the Promise object into your JSX, which React can't render.

Solution:

To fix this, you should handle the Promise correctly, using `**async/await**` or `**.then()**` to wait for the result before rendering the content.

If you're using `**useEffect**` to fetch data, you can manage the loading and error states and render the result only once the Promise is resolved.

Here's an example:

```
javascript

import { useState, useEffect } from 'react';

const MyComponent = () => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch('https://api.example.com/data');
        const result = await response.json();
        setData(result);
        setLoading(false);
      } catch (error) {
        setError(error);
        setLoading(false);
      }
    };

    fetchData();
  }, []);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error.message}</div>;

  return (
    <div>
      {/* Render the data once it is fetched */}
      <pre>{JSON.stringify(data, null, 2)}</pre>
    </div>
  );
};
```

In this example:

- We fetch data asynchronously using `**useEffect**` and `**async/await**`.
- We manage the loading and error states.
- We render the data only after it has been successfully fetched.

Make sure you're not trying to render a Promise directly in your JSX like this:

```
javascript

// Incorrect:
<div>{fetchData()}</div>  // fetchData() returns a Promise, not data
```

Instead, use the `**state**` to hold the result of the Promise once it's resolved.

Does this happen here somewhere?
import axiosInstance from '../api/AxiosInstance';

```
// Function to check the sentence and match conditions
const checkForAlerts = async (sentences, serialNumber, selectedShip, page) => {
    let alerts = [];  // Initialize alerts array to store any matching conditions
    const validSentences = Array.isArray(sentences) ? sentences : [];

    for (const sentence of validSentences) {
      // Extract condition and result from sentence
      const [condition, result] = sentence.split(" = ");
      const conditionParts = condition.split(" ");
      let parameterValue = 0;
      const conditionParameter = conditionParts[0];
      const comparator = conditionParts[1];
      const conditionValue = parseFloat(conditionParts[2]);

      const containerIdOfSN = await axiosInstance.get(/rest/container/by-serial-
number/${serialNumber});
      const fetchedID = containerIdOfSN.data.container;

      let ship;
      if(page === "main"){
        ship = selectedShip.id;
      }else{
        ship = selectedShip;
      }

      const environmentDataResponse = await axiosInstance.get(/rest/sensor/${ship}/${fetchedID});

      // Evaluate condition
      let isValid = false;

      // Check parameter based on condition
      switch (conditionParameter) {
        case "Air-Pressure":
          parameterValue = environmentDataResponse.data.sensor_data.air_pressure[0].value;
          break;
        case "Humidity":
          parameterValue = environmentDataResponse.data.sensor_data.humidity[0].value;
          break;
        case "Temperature":
          parameterValue = environmentDataResponse.data.sensor_data.temperature[0].value;
          break;
        case "Vibration":
          parameterValue = environmentDataResponse.data.sensor_data.vibration[0].value;
          break;
        case "Altitude":
          parameterValue = environmentDataResponse.data.sensor_data.altitude[0].value;
          break;
        case "Latitude":
          parameterValue = environmentDataResponse.data.sensor_data.latitude[0].value;
          break;
```

```
        case "Longitude":
          parameterValue = environmentDataResponse.data.sensor_data.longitude[0].value;
          break;
        default:
          isValid = false;
          break;
      }

      switch (comparator) {
        case ">":
          isValid = parameterValue > conditionValue;
          break;
        case "<":
          isValid = parameterValue < conditionValue;
          break;
        case "==":
          isValid = parameterValue === conditionValue;
          break;
        case "!=":
          isValid = parameterValue !== conditionValue;
          break;
        case ">=":
          isValid = parameterValue >= conditionValue;
          break;
        case "<=":
          isValid = parameterValue <= conditionValue;
          break;
        default:
          isValid = false;
          break;
      }

      // If the condition is valid, push the result into the alerts array
      if (isValid) {
        // Maybe change later
        if (result === 'Critical' || result === 'High' || result === 'Low'){
          alerts.push(${conditionParameter} of ${serialNumber} = ${result});
        }

      }
    };

    return alerts;
  };

  export default checkForAlerts;
```

Yes, the issue you're encountering could be related to the asynchronous `await` calls inside your `checkForAlerts` function. Specifically, the `await` calls to `axiosInstance.get()` are returning Promises,

but if you're trying to render any of the responses directly, React won't be able to handle those Promises and will throw the error about invalid React children.

Here's a breakdown of the potential issue:

1. **Asynchronous requests**: Inside `checkForAlerts`, you're making asynchronous calls to `axiosInstance.get()` to fetch data, and these calls return Promises.

   For example:

   ```
   javascript

   const containerIdOfSN = await axiosInstance.get(/rest/container/by-serial-number/${serialNumbe
   const environmentDataResponse = await axiosInstance.get(/rest/sensor/${ship}/${fetchedID});
   ```

   These calls themselves return Promises and are handled properly with `await`, but if you're trying to directly render the result (like the `alerts` array), that could lead to issues in React if the data isn't fully resolved when React attempts to render.

Solution:

- **Use `useEffect` or similar hooks** to handle the asynchronous logic in your component. This will ensure that the function completes the data fetching before attempting to render anything.
- **Handle state properly**: Store the result of `checkForAlerts` in a `useState` hook and render that state once the asynchronous operation completes.

Here's an example of how you could structure the logic in your component:

```
javascript

import { useState, useEffect } from 'react';
import checkForAlerts from './checkForAlerts';

const AlertsComponent = ({ sentences, serialNumber, selectedShip, page }) => {
  const [alerts, setAlerts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchAlerts = async () => {
      try {
        const result = await checkForAlerts(sentences, serialNumber, selectedShip, page);
        setAlerts(result);
      } catch (error) {
        setError(error);
      } finally {
        setLoading(false);
      }
    };

    fetchAlerts();
  }, [sentences, serialNumber, selectedShip, page]);

  if (loading) return <div>Loading alerts...</div>;
  if (error) return <div>Error: {error.message}</div>;

  return (
    <div>
      {alerts.length === 0 ? (
        <div>No alerts</div>
      ) : (
        alerts.map((alert, index) => <div key={index}>{alert}</div>)
      )}
```

```
    </div>
  );
};

export default AlertsComponent;
```

Key Changes:

- **State management**: The `alerts` state is used to store the result of `checkForAlerts`, which is fetched asynchronously in a `useEffect`.
- **Loading and error states**: You can display loading and error messages while the asynchronous operations are in progress.
- **Rendering the results**: After the alerts are fetched, you render them inside your component once the Promises are resolved.

This approach ensures that the data is fetched and resolved before trying to render it, which will avoid the issue with React throwing the `Promise` error.

So the code where the checkForAlerts is called is where the problem lies? If so its called here:
useEffect(() => {
  const fetchAndCheck = async () => {
    const currentAlerts = [];
    try {
      for (const containerId of containerIds) {
        const sentences = [];

        // Fetch serial Number of Id
        const containerSerialNumberResponse = await
axiosInstance.get(/rest/container/${containerId.id}/serial-number);
        const fetchedSerialNumber = containerSerialNumberResponse.data.serial_number;

        // Fetch thresholds of id
        const threshHoldsResponse = await
axiosInstance.get(/rest/container/${containerId.id}/thresholds);
        const fetchedThresholds = threshHoldsResponse.data?.thresholds || [];

        let thresholds = fetchedThresholds;
        const validThresholds = Array.isArray(thresholds) ? thresholds : [];

        // Fetch additional data for thresholds
        if (validThresholds.length > 0) {
          for (let i = 0; i < validThresholds.length; i++) {
            // Parameter
            const parameterId = validThresholds[i].parameter;
            const parameterResponse = await
axiosInstance.get(/rest/threshold/parameter/${parameterId});
            let parameterValue = parameterResponse.data.parameter[0].name;

            if (parameterValue === "Air Pressure") {
              parameterValue = "Air-Pressure";

```
          }

          // Rule
          const ruleId = validThresholds[i].rule;
          const ruleResponse = await axiosInstance.get(/rest/threshold/rule/${ruleId});
          const ruleValue = ruleResponse.data.rule[0].name;

          // Level
          const levelId = validThresholds[i].level;
          const levelResponse = await axiosInstance.get(/rest/threshold/level/${levelId});
          const levelValue = levelResponse.data.level[0].name;

          // Sentence
          const sentence = ${parameterValue} ${ruleValue} ${validThresholds[i].value} = ${levelValue};
          sentences.push(sentence);
        }
       }

       // Check for alerts after collecting sentences for the container
       currentAlerts.push(checkForAlerts(sentences, fetchedSerialNumber, selectedShip, page));
       sentences.length = 0;  // Clear sentences array for the next iteration
     }
     setAlerts(currentAlerts);
    } catch (error) {
     console.error("Failed to fetch containers of ship:", error.message);
    }
   };

   fetchAndCheck();
  }, [containerIds]);
```

Yes, it seems the issue is indeed here. Specifically, the problem is with the call to `**checkForAlerts**` inside the `**fetchAndCheck**` function.

The key issue is that `**checkForAlerts**` is an asynchronous function, and you are pushing its returned Promise directly into the `**currentAlerts**` array, like this:

```
javascript

currentAlerts.push(checkForAlerts(sentences, fetchedSerialNumber, selectedShip, page));
```

At this point, `**checkForAlerts**` returns a **Promise**, but you're trying to store the Promise, not the resolved value. React will eventually try to render that Promise as a child, leading to the error.

Solution:

You need to wait for `**checkForAlerts**` to resolve before pushing its result into the `**currentAlerts**` array. You can achieve this by using `**await**` inside the loop.

Here's how to fix it:

```javascript
useEffect(() => {
  const fetchAndCheck = async () => {
    const currentAlerts = [];
    try {
      for (const containerId of containerIds) {
        const sentences = [];

        // Fetch serial Number of Id
        const containerSerialNumberResponse = await axiosInstance.get(`/rest/container/${containerI
        const fetchedSerialNumber = containerSerialNumberResponse.data.serial_number;

        // Fetch thresholds of id
        const threshHoldsResponse = await axiosInstance.get(`/rest/container/${containerId.id}/thres
        const fetchedThresholds = threshHoldsResponse.data?.thresholds || [];

        let thresholds = fetchedThresholds;
        const validThresholds = Array.isArray(thresholds) ? thresholds : [];

        // Fetch additional data for thresholds
        if (validThresholds.length > 0) {
          for (let i = 0; i < validThresholds.length; i++) {
            // Parameter
            const parameterId = validThresholds[i].parameter;
            const parameterResponse = await axiosInstance.get(`/rest/threshold/parameter/${paramete
            let parameterValue = parameterResponse.data.parameter[0].name;

            if (parameterValue === "Air Pressure") {
              parameterValue = "Air-Pressure";
            }

            // Rule
            const ruleId = validThresholds[i].rule;
            const ruleResponse = await axiosInstance.get(`/rest/threshold/rule/${ruleId}`);
            const ruleValue = ruleResponse.data.rule[0].name;

            // Level
            const levelId = validThresholds[i].level;
            const levelResponse = await axiosInstance.get(`/rest/threshold/level/${levelId}`);
            const levelValue = levelResponse.data.level[0].name;

            // Sentence
            const sentence = `${parameterValue} ${ruleValue} ${validThresholds[i].value} = ${levelVa
            sentences.push(sentence);
          }
        }

        // Check for alerts after collecting sentences for the container
        // Wait for checkForAlerts to resolve before pushing the result
        const alertsForContainer = await checkForAlerts(sentences, fetchedSerialNumber, selectedShi
        currentAlerts.push(...alertsForContainer);  // Spread the alerts array into the currentAlert
        sentences.length = 0;  // Clear sentences array for the next iteration
      }
      setAlerts(currentAlerts);  // Set the alerts once all containers are processed
    } catch (error) {
      console.error("Failed to fetch containers of ship:", error.message);
    }
  };

  fetchAndCheck();
}, [containerIds, selectedShip, page]);  // Add the correct dependencies
```

Key Changes:

1. **Await the result of `checkForAlerts`**: The `await` is added to `checkForAlerts`, so the function will resolve before pushing its result into `currentAlerts`.

```
javascript

const alertsForContainer = await checkForAlerts(sentences, fetchedSerialNumber, selectedShip,
currentAlerts.push(...alertsForContainer);
```

This ensures that the actual alerts (not Promises) are added to the `currentAlerts` array.

2. **Spread operator**: Used `...alertsForContainer` to ensure that each alert is pushed individually into `currentAlerts`.
3. **Updated dependencies**: Make sure you add all necessary dependencies to the `useEffect` dependency array, such as `selectedShip` and `page`, to ensure the effect runs correctly when any of these values change.

Why This Fix Works:

- By `await`ing the result of `checkForAlerts`, you ensure that you're not trying to render a Promise. Instead, you will have an array of alerts (the resolved value) when you set the state with `setAlerts(currentAlerts)`. This fixes the issue where React encounters an unresolved Promise and throws an error.

Ab hier sind Nachrichten nur für dich sichtbar