

Token Refresh with Context



In a React project do I have to implement a useEffect in every component to use refresh tokens to refresh the access token? And how do I handle it that the access token which has an expiration time of 10 minutes) doesn't run out while the user is doing something?

```
import React, { useState } from 'react';
import axios from 'axios';
import axiosInstance from './api/AxiosInstance';
import LoginField from './components/LoginField';
import { FaEye, FaEyeSlash } from 'react-icons/fa';
import { HiArrowNarrowRight } from "react-icons/hi";
import { useNavigate } from 'react-router-dom';

// This is the Login Page that displays whenever the site is pulled up
const LoginPage = () => {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState(null);
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError(null);

    try {
      // First Call to get ID of username
      const responseId = await axiosInstance.get(`auth/user/${username}`);
      const id = responseId.data.user[0].id; // Zugriff auf die erste ID
      //console.log("Extracted ID:", id);

      if (!id) {
        throw new Error('Invalid user ID received');
      }

      const body = {
        user: id,
        password: password,
      };

      const responseLogin = await axiosInstance.post('auth/login', body, {
        headers: { "Content-Type": "application/json" },
      });

      const {accessToken, refreshToken} = responseLogin.data;
      localStorage.setItem("userId", id);
    }
  }
}

export default LoginPage;
```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

1/6

```
localStorage.setItem("accessToken", accessToken);
localStorage.setItem("refreshToken", refreshToken);

// Navigate to main page on success
navigate('/main');

} catch (error) {
  console.log('Error', error);
  if (error.response) {
    setError(error.response.data.message || 'Login failed. Please try again.');
  } else {
    setError('Network error or server is down.');
  }
}

return (
  <div className="flex flex-col h-screen items-center justify-center bg-cover bg-center">
    <div className="bg-white p-6 rounded-lg max-w-md w-full border-2 mt-36">
      
      <form onSubmit={handleSubmit} className="space-y-4">
        <LoginField placeholder="User" value={username} onChange={(e) => setUsername(e.target.value)} />
        <LoginField placeholder="PW" isPassword={true} value={password} onChange={(e) => setPassword(e.target.value)} />
        <div className="w-full flex justify-center">
          <button type="submit" className="bg-gray-400 text-white text-2xl py-2 hover:bg-gray-700 rounded-full w-16 h-16 flex justify-center items-center hover:font-bold transform hover:scale-105 transition-all duration-200">
            <HiArrowNarrowRight />
          </button>
        </div>
      </form>
      {error && (
        <div className="text-red-500 text-center mt-4">
          {error}
        </div>
      )}
      <div className="text-gray-400 mt-auto mb-2">
        Impressum | Kontakt
      </div>
    </div>
  </div>
)

export default LoginPage;
```

Is this possible with an axios interceptor?

import axios from 'axios';

2/6

```
const axiosInstance = axios.create({
  baseURL: 'https://api.contrude.eu',
  headers: {
    'Content-Type': 'application/json',
  },
});

export default axiosInstance;
```



Yes, you can handle token refreshing with an axios interceptor, which can automatically refresh the access token when it expires without requiring `useEffect` in each individual component. This centralizes the logic and makes your components cleaner.

Here's how you can implement this in your React project:

- Set up Axios Interceptors:** You can add a request interceptor to add the `accessToken` to every outgoing request. Then, add a response interceptor to catch any `401 Unauthorized` errors, which typically happen when the access token has expired.
- Refresh Token Logic:** When the access token has expired (i.e., a `401` status code is returned), you can use the `refreshToken` to obtain a new `accessToken` and retry the original request.

Here's an example of how to set up the interceptors:

Axios Instance with Interceptors

```
js

import axios from 'axios';

// Create an Axios instance
const axiosInstance = axios.create({
  baseURL: 'https://api.contrude.eu',
  headers: {
    'Content-Type': 'application/json',
  },
});

// Add request interceptor to attach access token to headers
axiosInstance.interceptors.request.use(
  (config) => {
    const accessToken = localStorage.getItem('accessToken');
    if (accessToken) {
      config.headers['Authorization'] = `Bearer ${accessToken}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

// Add response interceptor to handle token refresh
axiosInstance.interceptors.response.use(
  (response) => response,
  async (error) => {
    if (error.response && error.response.status === 401) {
      // Token has expired, try refreshing it
      try {
        // Your backend endpoint for refreshing tokens
        const responseRefresh = await axiosInstance.post('auth/refresh', { refreshToken });
        const { accessToken } = responseRefresh.data;
        localStorage.setItem('accessToken', accessToken);
        // Retrying the original request with the new access token
        error.config.headers['Authorization'] = `Bearer ${accessToken}`;
        return axiosInstance(error.config); // Retry original request
      } catch (refreshError) {
        console.log('Error refreshing token', refreshError);
        // Handle token refresh failure (e.g., log the user out)
      }
    }
    return Promise.reject(error);
  }
);
```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

3/6

```
const refreshToken = localStorage.getItem('refreshToken');
if (!refreshToken) {
  throw new Error('No refresh token found');
}

// Send a request to refresh the token
const response = await axios.post('auth/refresh', { refreshToken });

// Get the new access token from the response
const { accessToken } = response.data;
localStorage.setItem('accessToken', accessToken);

// Retry the original request with the new access token
error.config.headers['Authorization'] = `Bearer ${accessToken}`;
return axiosInstance(error.config); // Retry original request
} catch (refreshError) {
  console.log('Error refreshing token', refreshError);
  // Handle token refresh failure (e.g., log the user out)
}
return Promise.reject(error);
};

export default axiosInstance;
```

Key Points:

- Request Interceptor:** This ensures that each request includes the `accessToken` in the `Authorization` header.
- Response Interceptor:** This handles the case when a `401` error occurs, meaning the `accessToken` has expired. It tries to refresh the token using the `refreshToken` and retries the original request with the new token.
- Retrying the Request:** The original request is retried with the updated access token in the headers.

Notes:

- /auth/refresh Endpoint:** Make sure your backend has an endpoint for refreshing tokens (`POST /auth/refresh`), which takes the `refreshToken` and returns a new `accessToken`.
- Error Handling:** If the refresh fails (e.g., the `refreshToken` is also expired), you should log the user out or redirect them to the login page.

Usage in Components:

Now, in your components like `LoginPage`, you don't need to worry about token expiry anymore. The axios instance automatically handles the access token refresh and retries failed requests.

For instance, in `LoginPage`, when you call `axiosInstance` for making API requests, the interceptor will manage the tokens for you, so you don't need to manually check token expiration and refresh it.

This approach centralizes token management and ensures a seamless user experience, especially for longer sessions.



Assuming the refresh token expires as well then the user wont be able to do anything anymore

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

4/6

right?



Yes, that's correct. If the refresh token expires as well, the user will no longer be able to obtain a new access token, which means they won't be able to perform any authenticated actions until they log in again.

In that case, you should handle the situation by redirecting the user to the login page or showing a session expired message. Here's how you can handle it:

- Handle Refresh Token Expiration:** If the refresh token has expired or is invalid (i.e., when the refresh token request fails), you can log the user out by removing all tokens from `localStorage` and redirect them to the login page.
- Prompt the User to Re-authenticate:** You can show an alert or a modal to notify the user that their session has expired and they need to log in again.

Example Code for Refresh Token Failure

You can update the `axiosInstance` response interceptor to handle the case when the refresh token request fails:

```
js

axiosInstance.interceptors.response.use(
  (response) => response,
  async (error) => {
    if (error.response && error.response.status === 401) {
      // Token has expired or is invalid, try refreshing it
      try {
        const refreshToken = localStorage.getItem('refreshToken');
        if (!refreshToken) {
          throw new Error('No refresh token found');
        }

        // Attempt to refresh the token
        const response = await axios.post('/auth/refresh', { refreshToken });

        // Get the new access token
        const accessToken = response.data;
        localStorage.setItem('accessToken', accessToken);

        // Retry the original request with the new access token
        error.config.headers['Authorization'] = `Bearer ${accessToken}`;
        return axiosInstance(error.config); // Retry original request
      } catch (refreshError) {
        console.log('Error refreshing token', refreshError);
      }

      // Remove the expired tokens and redirect to login
      localStorage.removeItem('accessToken');
      localStorage.removeItem('refreshToken');

      // Optionally show a session expired message or alert
      alert('Your session has expired. Please log in again.');
      window.location.href = '/login'; // Redirect to login page
    }
  }

  return Promise.reject(error);
}
```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

5/6

});

Key Changes:

- catch Block for Refresh Token Error:** If the refresh token request fails, the user is logged out by removing the `accessToken` and `refreshToken` from `localStorage`.
- Session Expired Message:** You can display an alert or a modal informing the user that their session has expired.
- Redirection:** After logging the user out, you redirect them to the login page (`/login`) so they can authenticate again.

User Experience:

- If the refresh token has expired, the user will be redirected to the login page where they can log in again.
- This ensures that expired sessions are handled gracefully, and the user is not left stuck with expired tokens.

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

6/6

JS Condition Checker



Hi I need a JS Script that does the following:
you get this:

A String Sentence that looks like one of these:
Air Pressure > 101.32500 = Medium

Humidity < 50.00000 = Low

Temperature == 25.00000 = Ok

Vibration != 9.80000 = Low

Altitude > 100.00000 = Ok

Latitude > 45.00000 = Critical

Longitude < 90.00000 = High

And one line of an array like this:

```
const [tableData, setTableData] = useState([
  {environment: "Temperature", value: "200", unit: "C", alert: ""},
  {environment: "Pressure", value: "200", unit: "Pa", alert: ""},
  {environment: "Humidity", value: "50", unit: "%", alert: ""},
  {environment: "Vibration", value: "9.8", unit: "m/s^2", alert: ""},
  {environment: "Altitude", value: "100", unit: "m", alert: ""},
  {environment: "Latitude", value: "45", unit: "DD", alert: ""},
  {environment: "Longitude", value: "90", unit: "DD", alert: ""}
]);
```

For each line i want you to take the value of the Array then check with the sentence that has the correct parameter (like for environment Pressure, take the sentence that starts with Air Pressure).

For example you choose this line:

Air Pressure > 101.32500 = Medium

because this says Pressure: { environment: "Pressure", value: "200", unit: "Pa", alert: "" },

Now you do something like this:

you replace "Air Pressure" with value (200):

200 > 101.32500 = Medium

If the statement is true you return "Medium" or whatever there stands if it is not true, you return Nothing



You can create a function that iterates over your `tableData` and matches the `environment` from the data to the corresponding sentence. It will then evaluate the condition and return the appropriate alert value. Here's how you can implement this in JavaScript:

javascript

```
// Sample sentence array
const sentences = [
  "Air Pressure > 101.32500 = Medium",
  "Humidity < 50.00000 = Low",
  "Temperature == 25.00000 = Ok",
  "Vibration != 9.80000 = Low",
  "Altitude > 100.00000 = Ok",
  "Latitude > 45.00000 = Critical",
  "Longitude < 90.00000 = High"
];

// Array of data
const tableData = [
  {environment: "Temperature", value: "200", unit: "C", alert: ""},
  {environment: "Pressure", value: "200", unit: "Pa", alert: ""},
  {environment: "Humidity", value: "50", unit: "%", alert: ""},
  {environment: "Vibration", value: "9.8", unit: "m/s^2", alert: ""},
  {environment: "Altitude", value: "100", unit: "m", alert: ""},
  {environment: "Latitude", value: "45", unit: "DD", alert: ""},
  {environment: "Longitude", value: "90", unit: "DD", alert: ""}
];

// Function to check the sentence and match conditions
const checkConditions = (tableData, sentences) => {
  return tableData.map((data) => {
    // Find the sentence that matches the environment
    const sentence = sentences.find((sentence) => sentence.toLowerCase().includes(data.environment.toLowerCase()));

    if (!sentence) {
      return { ...data, alert: "" }; // No matching sentence found
    }

    // Extract condition and alert from sentence
    const [condition, result] = sentence.split(" = ");
    const conditionParts = condition.split(" ");
    const parameterValue = parseFloat(data.value);
    const comparator = conditionParts[1];
    const conditionValue = parseFloat(conditionParts[2]);

    // Evaluate condition
    let isValid = false;

    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;
    }

    return { ...data, alert: isValid ? result : "" };
  });
};
```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

1/3

2/3

```
// Call the function and log the results
const updatedTableData = checkConditions(tableData, sentences);
console.log(updatedTableData);
```

Explanation:

1. `sentences` Array: Holds the conditions as strings.
2. `tableData` Array: Represents your data, with environmental parameters (like Temperature, Pressure) and their values.
3. `checkConditions` Function:
 - o For each item in `tableData`, it looks for a matching sentence based on the `environment` property.
 - o It then extracts the comparison operator (>, <, ==, etc.) and evaluates whether the condition is true.
 - o If the condition is true, it sets the `alert` property to the result from the sentence (e.g., "Medium", "Low").
 - o If the condition is false, it sets the `alert` property to an empty string.
4. Result: The function returns the `tableData` array with updated `alert` properties based on the conditions.

Example Output:

```
json
[{"environment": "Temperature", "value": "200", "unit": "°C", "alert": ""}, {"environment": "Pressure", "value": "200", "unit": "Pa", "alert": "Medium"}, {"environment": "Humidity", "value": "45", "unit": "%", "alert": "Low"}, {"environment": "Vibration", "value": "9.8", "unit": "m/s²", "alert": ""}, {"environment": "Altitude", "value": "120", "unit": "m", "alert": "Ok"}, {"environment": "Latitude", "value": "50", "unit": "DD", "alert": "Critical"}, {"environment": "Longitude", "value": "85", "unit": "DD", "alert": "High"}]
```

This script will dynamically check each environment value against the sentence and return the appropriate alert or nothing if the condition doesn't hold.



Kannst du sehr kurz in FLIEßTEXT erklären, wie das Script funktioniert?



Das Script nimmt die Umweltdaten aus dem `tableData`-Array und prüft für jede Zeile, ob es einen passenden Satz aus der `sentences`-Liste gibt, basierend auf dem `environment`-Wert. Wenn ein passender Satz gefunden wird, wird die Bedingung aus dem Satz extrahiert (z. B. Vergleichsoperator und Grenzwert) und mit dem Wert aus `tableData` verglichen. Ist die Bedingung erfüllt, wird das Ergebnis (z. B. "Medium" oder "Low") in das `alert`-Feld der Zeile geschrieben. Wenn die Bedingung nicht erfüllt ist oder kein passender Satz gefunden wird, bleibt das `alert`-Feld leer.

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

3/3

I need a js script that does the following:

There is a grid:
1x1, 2x2, 2x4, 3x4, 4x4

And there is a number of values

each section of a grid can hold however many values there are

e.g.

Grid = 2x2
values = 4x

each spot has 1

Grid = 2x2
values = 3x

the first three have 1 the last 0 (because there is nothing remaining)

However:
if i have this:

Grid = 2x2
values = 9x
then it should look like this:

3 | 2

2 | 2

How do I calculate that?

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

1/1

D3.js Container Visualisierung



```
{
  "contId": "cont4",
  "subs": [
    {
      "contId": "cont2",
      "subs": [{"contId": "cont4"}]
    },
    {
      "contId": "cont3",
      "subs": [{"contId": "cont5"}]
    }
  ]
}
```

Ertsele mir bitte auf Basis eines JSON Konstrukts wie oben zu sehen ist (beachte dass die subs Verschachtelungen auch länger sein können) einen JavaScript Code mit Dragable Graph (d3.js) welcher die Verbindungen dieser Container ids (contid) darstellt. Bitte beachte folgendes:
-> pro 2 Container sollte nur eine Verbindung sein
-> es darf keine ContainerId doppelt vorkommen
-> es darf auch alleinstehende Ids (also ohne Verbindung) geben
-> alle notwendigen Codestücke in einem File!
-> das json file heißt "graphSpecific.json" und befindet sich im selben Verzeichnis wie das File



Hier ist ein JavaScript-Code, der D3.js verwendet, um ein interaktives, dragbares Graphen-Diagramm zu erstellen, das die Verbindungen zwischen den Container-IDs darstellt. Der Code geht davon aus, dass du D3.js korrekt installiert hast und dass die Datei 'graphSpecific.json' im selben Verzeichnis wie das JavaScript-File liegt.

Stelle sicher, dass du D3.js über ein CDN in deinem HTML einbindest, falls du das nicht bereits getan hast.

HTML + JavaScript-Code

```
html
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Draggable Graph</title>
  <script src="https://d3js.org/d3.v7.min.js"></script>
  <style>
    .node {
      fill: #1f77b4;
```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

1/11

```

    cursor: pointer;
  }
  .link {
    fill: none;
    stroke: #ccc;
    stroke-width: 1.5px;
  }
  .node text {
    font: 12px sans-serif;
    pointer-events: none;
    fill: #fff;
  }
</style>
</head>
<body>
<svg width="960" height="600"></svg>

<script>
  // Asynchrones Laden der JSON-Datei
  d3.json("graphSpecific.json").then(function(graphData) {
    // Nodes und Verbindungen und Knoten vorbereiten
    const nodes = [];
    const links = [];
    const nodeMap = {}; // Um doppelte IDs zu vermeiden
    // Funktion zur Rekursion der Container-Struktur
    function traverse(subs, parentId) {
      subs.forEach(sub => {
        // Wenn der Container noch nicht im nodes-Array ist
        if (!nodeMap[sub.contId]) {
          nodes.push({ id: sub.contId });
          nodeMap[sub.contId] = true;
        }
        // Verbindung zwischen parentId und sub.contId
        links.push({ source: parentId, target: sub.contId });
        // Rekursion für tiefer verschachtelte Subs
        if (sub.subs && sub.subs.length > 0) {
          traverse(sub.subs, sub.contId);
        }
      });
    }
    // Startknoten ist der Root-Container
    if (graphData.subs) {
      traverse(graphData.subs, graphData.contId);
    }
    // Alle Knoten, die ohne Verbindung sind, als unabhängige Knoten hinzufügen
    Object.keys(nodeMap).forEach(id => {
      if (!nodes.find(node => node.id === id)) {
        nodes.push({ id });
      }
    });
    // SVG und D3-Setup
    const svg = d3.select('svg');
    const width = +svg.attr('width');
    const height = +svg.attr('height');

    const simulation = d3.forceSimulation(nodes)
      .force('link', d3.forceLink(links).id(d => d.id).distance(100))
      .force('charge', d3.forceManyBody().strength(-200))
      .force('center', d3.forceCenter(width / 2, height / 2));
    // Links erstellen
    const link = svg.append('g')
      .selectAll('.link')
      .data(links)
      .enter().append('line')
      .attr('class', 'link');
    // Knoten erstellen
    const node = svg.append('g')
      .selectAll('.node')
      .data(nodes)
```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

2/11

```

    .enter().append('g')
      .attr('class', 'node')
      .call(d3.drag()
        .on('start', dragStarted)
        .on('drag', dragged)
        .on('end', dragEnded));
  
```

// Knotenkreis und Text hinzufügen
 node.append('circle')
 .attr('r', 20)
 .attr('class', 'node');

node.append('text')
 .attr('dy', -25)
 .attr('text-anchor', 'middle')
 .text(d => d.id);

// Simulation tickt und aktualisiert Positionen der Knoten und Verbindungen
 simulation.on('tick', function() {
 link
 .attr('x1', d => d.source.x)
 .attr('y1', d => d.source.y)
 .attr('x2', d => d.target.x)
 .attr('y2', d => d.target.y);

 node
 .attr('transform', d => `translate(\${d.x},\${d.y})`);
 });

// Drag-Verhalten
 function dragStarted(event, d) {
 if (!event.active) simulation.alphaTarget(0.3).restart();
 d.fx = d.x;
 d.fy = d.y;
 }

 function dragged(event, d) {
 d.fx = event.x;
 d.fy = event.y;
 }

 function dragEnded(event, d) {
 if (!event.active) simulation.alphaTarget(0);
 d.fx = null;
 d.fy = null;
 }
}).catch(function(error) {
 console.log(error);
});
</script>
</body>
</html>

Erklärung:

- Laden der JSON-Daten:** Die JSON-Datei `graphSpecific.json` wird mit `d3.json()` geladen.
- Rekursion der `subs`:** Die Funktion `traverse` wird verwendet, um die verschachtelten `subs` zu durchlaufen und alle Knoten und Verbindungen zu extrahieren.
- Verbindungen und Knoten:** Verbindungen zwischen Container-IDs werden erstellt. Es wird sichergestellt, dass Container-IDs nicht doppelt vorkommen.
- D3.js Visualisierung:** Ein SVG wird verwendet, um die Knoten (Container) als Kreise und die Verbindungen als Linien darzustellen. Knoten sind interaktiv und können per Drag & Drop bewegt werden.
- Simulation:** D3's `forceSimulation` sorgt dafür, dass die Knoten dynamisch angeordnet und die Verbindungen entsprechend den physikalischen Kräften positioniert werden.

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

3/11

Wie man den Code verwendet:

- Speichere den HTML-Code in einer Datei (z.B. `index.html`).
- Stelle sicher, dass die Datei `graphSpecific.json` im selben Verzeichnis liegt.
- Öffne die HTML-Datei in einem Browser, um das interaktive Diagramm zu sehen.



```
Error: node not found: cont2
at Yc (d3.v7.min.js:2:102583)
at h (d3.v7.min.js:2:232576)
at Function.initialize (d3.v7.min.js:2:233061)
at p (d3.v7.min.js:2:236157)
at Object.force (d3.v7.min.js:2:236803)
at DragableGraph.html?_jtt=ia7o5h25rk8v2s1ksgm9fku96g:71:14
```



Der Fehler "node not found: cont2" tritt auf, weil D3 die Knoten in den Links referenziert, bevor sie tatsächlich im `nodes`-Array existieren. Dies passiert, weil in der Rekursionsfunktion zwar die Verbindungen zwischen den Containern erzeugt werden, aber die Knoten für diese Container nicht direkt in das `nodes`-Array eingefügt werden.

Um den Fehler zu beheben, müssen wir sicherstellen, dass alle Container, auf die durch Verbindungen verwiesen wird, auch tatsächlich im `nodes`-Array vorhanden sind. Wir sollten also die Knoten und Verbindungen gleichzeitig in der Rekursion hinzufügen, ohne dass ein Container referenziert wird, bevor er in das `nodes`-Array aufgenommen wurde.

Hier ist der aktualisierte Code, der dieses Problem behebt:

HTML + JavaScript-Code (mit Fix)

```

<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dragable Graph</title>
  <script src="https://d3js.org/d3.v7.min.js"></script>
  <style>
    .node {
      fill: #1f77b4;
      cursor: pointer;
    }
    .link {
      fill: none;
      stroke: #ccc;
      stroke-width: 1.5px;
    }
    .node text {
      font: 12px sans-serif;
      pointer-events: none;
      fill: #fff;
    }
  </style>
</head>
<body>
```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

4/11

```

<body>
  <svg width="960" height="600"></svg>

  <script>
    // Asynchrones Laden der JSON-Datei
    d3.json('graphSpecific.json').then(function(graphData) {
      // Verbindungen und Knoten vorbereiten
      const nodes = [];
      const links = [];
      const nodeMap = {}; // Um doppelte IDs zu vermeiden

      // Funktion zur Rekursion der Container-Struktur
      function traverse(subs, parentId) {
        subs.forEach(sub => {
          // Wenn der Container noch nicht im nodes-Array ist
          if (!nodeMap[sub.contId]) {
            nodes.push({ id: sub.contId });
            nodeMap[sub.contId] = true;
          }
          // Verbindung zwischen parentId und sub.contId
          links.push({ source: parentId, target: sub.contId });
          // Rekursion für tiefer verschachtelte Subs
          if (sub.subs && sub.subs.length > 0) {
            traverse(sub.subs, sub.contId);
          }
        });
      }

      // Der Root-Container (cont4) wird als Startpunkt genutzt
      nodes.push({ id: graphData.contId });
      nodeMap[graphData.contId] = true;

      // Die Verschachtelung der Subs des Root-Containers verarbeiten
      if (graphData.subs) {
        traverse(graphData.subs, graphData.contId);
      }

      // Alle Knoten, die ohne Verbindung sind, als unabhängige Knoten hinzufügen
      Object.keys(nodeMap).forEach(id => {
        if (!nodes.find(node => node.id === id)) {
          nodes.push({ id });
        }
      });

      // SVG und D3-Setup
      const svg = d3.select('svg');
      const width = +svg.attr('width');
      const height = +svg.attr('height');

      const simulation = d3.forceSimulation(nodes)
        .force('link', d3.forceLink(links).id(d => d.id).distance(100))
        .force('charge', d3.forceCharge().strength(-200))
        .force('center', d3.forceCenter(width / 2, height / 2));

      // Links erstellen
      const link = svg.append('g')
        .selectAll('.link')
        .data(links)
        .enter().append('line')
        .attr('class', 'link');

      // Knoten erstellen
      const node = svg.append('g')
        .selectAll('.node')
        .data(nodes)
        .enter().append('g')
        .attr('class', 'node')
        .call(d3.drag()
          .on('start', dragStarted)
          .on('drag', dragged)
          .on('end', dragEnded));

      // Knotenkreis und Text hinzufügen
      node.append('circle')
        .attr('r', 20);
    });
  </script>
</body>
</html>

```

Wichtige Änderungen:

- Root-Container hinzugefügt:** Der `graphData.contId` (z.B. `cont4`) wird nun als Knoten explizit zum `nodes`-Array hinzugefügt, bevor wir mit den `subs` weiterarbeiten.
- Rekursion angepasst:** Die Funktion `traverse` sorgt dafür, dass alle Container, die durch die `subs` referenziert werden, korrekt im `nodes`-Array erscheinen.
- Verbindungen erstellt:** Für jedes `sub` wird eine Verbindung mit dem übergeordneten Container (`parentId`) hergestellt.

Dieser Code sollte jetzt ohne den Fehler "node not found" laufen und dir das gewünschte dragbare Graph-Diagramm der Container-IDs anzeigen.



Mach bitte die Schriftfarbe und Knotenfarbe schwarz!

Ausserdem stelle bitte sicher, dass der gesamte Platz im Browser benutzt werden kann. Außerdem möchte ich, dass es möglich ist, wenn man auf einen Knoten klickt (dass dieser und seine Verbindungen) farblich hervorgehoben werden, wobei die Knoten mit denen der ausgewählte Verbindungen hat in einer anderen Farbe hervorgehoben wird

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

5/11

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

6/11

Hier ist der aktualisierte Code mit den angeforderten Änderungen:

1. **Schrift- und Knotenfarbe auf Schwarz.**
2. **Das Diagramm nutzt den gesamten Platz im Browser.**
3. **Klick auf einen Knoten hebt den Knoten und seine Verbindungen farblich hervor:**
 - o Der ausgewählte Knoten wird in einer Farbe (z. B. Blau) hervorgehoben.
 - o Die Knoten, die mit ihm verbunden sind, werden in einer anderen Farbe (z. B. Grün) hervorgehoben.

HTML + JavaScript-Code

```
html
<!DOCTYPE html>
<html lang="de">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Draggable Graph</title>
<script src="https://d3js.org/d3.v7.min.js"></script>
<style>
    body, html {
        margin: 0;
        padding: 0;
        overflow: hidden;
    }
    svg {
        width: 100vw;
        height: 100vh;
        background-color: #f9f9f9;
    }
    .node {
        fill: black;
        cursor: pointer;
    }
    .link {
        fill: none;
        stroke: #ccc;
        stroke-width: 1.5px;
    }
    text {
        font: 12px sans-serif;
        pointer-events: none;
        fill: black;
    }
    .highlight {
        fill: blue;
    }
    .connected {
        fill: green;
    }
    .highlight-link {
        stroke: orange;
        stroke-width: 2.5px;
    }
</style>
</head>
<body>
<svg></svg>
<script>
    d3.json('graphSpecific.json').then(function(graphData) {
        const nodes = [];
        const links = [];
        const nodeMap = {};
    });

```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

7/11

```
function traverse(subs, parentId) {
    subs.forEach(sub => {
        if (!nodeMap[sub контид]) {
            nodes.push({ id: sub контид });
            nodeMap[sub контид] = true;
        }
        links.push({ source: parentId, target: sub контид });
        if (sub.subs && sub.subs.length > 0) {
            traverse(sub.subs, sub контид);
        }
    });
}
nodes.push({ id: graphData контид });
nodeMap[graphData контид] = true;
if (graphData.subs) {
    traverse(graphData.subs, graphData контид);
}
Object.keys(nodeMap).forEach(id => {
    if (!nodes.find(node => node.id === id)) {
        nodes.push({ id });
    }
});
const svg = d3.select('svg');
const width = window.innerWidth;
const height = window.innerHeight;
const simulation = d3.forceSimulation(nodes)
    .force('link', d3.forceLink(links).id(d => d.id).distance(100))
    .force('charge', d3.forceManyBody().strength(-300))
    .force('center', d3.forceCenter(width / 2, height / 2));
const link = svg.append('g')
    .selectAll('.link')
    .data(links)
    .enter().append('line')
    .attr('class', 'link');
const node = svg.append('g')
    .selectAll('.node')
    .data(nodes)
    .enter().append('g')
    .attr('class', 'node')
    .call(d3.drag()
        .on('start', dragstarted)
        .on('drag', dragged)
        .on('end', dragended)
        .on('click', highlightNode));
node.append('circle')
    .attr('r', 20)
    .attr('class', 'node');
node.append('text')
    .attr('dy', -25)
    .attr('text-anchor', 'middle')
    .text(d => d.id);
simulation.on('tick', function() {
    link
        .attr('x1', d => d.source.x)
        .attr('y1', d => d.source.y)
        .attr('x2', d => d.target.x)
        .attr('y2', d => d.target.y);
    node
        .attr('transform', d => `translate(${d.x},${d.y})`);
});
function dragStarted(event, d) {
    if (!event.active) simulation.alphaTarget(0.3).restart();
}
function dragStarted(event, d) {
    if (!event.active) simulation.alphaTarget(0.3).restart();
}

```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

8/11

```
d.fx = d.x;
d.fy = d.y;

function dragged(event, d) {
    d.fx = event.x;
    d.fy = event.y;
}

function dragended(event, d) {
    if (!event.active) simulation.alphaTarget(0);
    d.fx = null;
    d.fy = null;
}

function highlightNode(event, selectedNode) {
    node.selectAll('circle').classed('highlight', false).classed('connected', false);
    link.classed('highlight-link', false);

    // Knoten hervorheben
    d3.select(event.currentTarget).select('circle').classed('highlight', true);

    // Verbindungen hervorheben
    link.filter(d => d.source.id === selectedNode.id || d.target.id === selectedNode.id)
        .classed('highlight-link', true);

    // Verbundene Knoten hervorheben
    node.filter(d => {
        links.some(link => {
            (link.source.id === selectedNode.id && link.target.id === d.id) ||
            (link.target.id === selectedNode.id && link.source.id === d.id)
        })
        .select('circle')
        .classed('connected', true);
    });
}
).catch(function(error) {
    console.log(error);
});
</script>
</body>
</html>
```

Änderungen:

1. **Schrift- und Knotenfarbe auf Schwarz:**
 - o Die Standardfarben für `node` und `text` wurden auf Schwarz gesetzt.
2. **SVG auf vollen Browserplatz erweitert:**
 - o Das SVG füllt den gesamten verfügbaren Platz im Browserfenster mit `width: 100vw` und `height: 100vh`.
3. **Klick-Highlight-Funktion:**
 - o Beim Klick auf einen Knoten wird der ausgewählte Knoten in Blau hervorgehoben.
 - o Verbundene Knoten werden in Grün hervorgehoben.
 - o Die Links zwischen diesen Knoten werden orange hervorgehoben.

Dieser Code sollte die gewünschten Funktionen vollständig abdecken. Du kannst ihn speichern und testen!



Perfekt. Brich bitte nun den Code auf seine wichtigsten Bestandteile runter und erkläre diese in

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

einem Fließtext Format



Hier ist eine Erklärung der wichtigsten Bestandteile des Codes in Fließtextform:

JSON-Verarbeitung und Datenstruktur

Die JSON-Datei `graphSpecific.json` enthält die Container-IDs und deren verschachtelte Verbindungen. Beim Laden der Datei wird eine Funktion `traverse` verwendet, um die hierarchische Struktur zu durchlaufen. Die Container werden dabei in zwei Arrays aufgeteilt:

1. `nodes`: Enthält alle Container als individuelle Knoten, basierend auf ihren IDs.
2. `links`: Enthält die Verbindungen zwischen Containern, dargestellt als Paare von Quell- und Ziel-Knoten (`source` und `target`).

Ein Hilfsobjekt `nodeMap` stellt sicher, dass keine Container-ID mehrfach in `nodes` aufgenommen wird.

SVG-Element und D3-Setup

Ein `<svg>-Element wird dynamisch mit D3 erstellt, um den Graphen darzustellen. Es nutzt den gesamten verfügbaren Platz im Browserfenster (100w' x 100h').

D3s Kraft-gestützte Simulation (`forceSimulation`) berechnet die Positionen der Knoten basierend auf folgenden Kräften:

- `forceLink`: Verbindet Knoten basierend auf den Links.
- `forceManyBody`: Erzeugt eine Abstoßung zwischen Knoten, damit sie nicht zu dicht beieinander liegen.
- `forceCenter`: Zentriert das gesamte Diagramm im SVG-Bereich.

Knoten und Verbindungen

1. Verbindungen (Links):

- o Jede Verbindung zwischen zwei Containern wird als Linie dargestellt.
- o Die Linien werden aus den Daten in `links` generiert.

2. Knoten:

- o Jeder Container wird als Kreis dargestellt.
- o Jeder Kreis zeigt zusätzlich die Container-ID als Text oberhalb des Kreises an.

Die Knoten sind interaktiv gestaltet, sodass sie durch Ziehen (Drag-and-Drop) bewegt werden können. Dies wird mit D3s Dragging-API umgesetzt.

Highlighting bei Klick auf einen Knoten

Beim Klick auf einen Knoten werden der Knoten selbst sowie seine Verbindungen hervorgehoben:

- Der angeklickte Knoten erhält eine **blaue Farbe**.
- Die mit ihm verbundenen Knoten erhalten eine **grüne Farbe**.

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

10/11

- Die Verbindungen zwischen diesen Knoten werden **orange markiert**.

Dies wird durch Filterfunktionen umgesetzt, die prüfen, ob eine Verbindung von oder zu dem ausgewählten Knoten führt. Die Klassen `highlight`, `connected` und `highlight-link` steuern die Darstellung der markierten Elemente.

Drag-and-Drop-Funktion

Die Knoten können per Drag-and-Drop verschoben werden.

- Beim Start des Ziehens wird die Simulation neu gestartet, um die Positionen dynamisch anzupassen.
- Während des Ziehens wird die aktuelle Position des Knotens festgelegt.
- Nach Beendigung des Ziehens wird die Simulation gestoppt, und der Knoten kann frei in seiner neuen Position verbleiben.

Zusammenfassung

Der Code kombiniert D3.js-Funktionalitäten, um ein interaktives, visuell ansprechendes Graph-Diagramm zu erstellen:

- JSON-Daten werden in Knoten und Verbindungen umgewandelt.
- Eine Simulation bestimmt die Positionen der Elemente.
- Nutzer können Knoten per Drag-and-Drop bewegen und Knoten sowie Verbindungen durch Klicken hervorheben.

Das Ergebnis ist ein flexibles, interaktives Diagramm, das die Beziehungen zwischen Containern auf einfache und effektive Weise visualisiert.

Ist dieses Gespräch hilfreich?

ChatGPT

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



Es gibt folgendes Problem:

Namen:
cont0
cont10
cont100

es soll immer die Id (also 0, 10, 100) extrahiert werden und in ein Integer gepackt werden



Um die IDs aus den Namen wie `cont0`, `cont10` und `cont100` zu extrahieren und in ein Integer-Array zu packen, kannst du in Java die folgende Methode verwenden:

```
java

import java.util.ArrayList;
import java.util.List;

public class ContainerIdExtractor {
    public static void main(String[] args) {
        String[] names = {"cont0", "cont10", "cont100"};
        List<Integer> ids = extractIds(names);

        // Ausgabe der extrahierten IDs
        for (Integer id : ids) {
            System.out.println(id);
        }
    }

    public static List<Integer> extractIds(String[] names) {
        List<Integer> ids = new ArrayList<>();

        for (String name : names) {
            // Entferne "cont" und parse die restliche Zahl als Integer
            String idString = name.replace("cont", "");
            ids.add(Integer.parseInt(idString));
        }

        return ids;
    }
}
```

Erklärungen:

- `String[] names`: Ein Array mit den Namen, aus denen die IDs extrahiert werden.
- extractIds-Methode:**
 - Durchläuft jedes Element im `names`-Array.
 - Entfernt den String "cont" mit `replace`.
 - Konvertiert den verbleibenden String in einen Integer mit `Integer.parseInt`.

- Fügt den extrahierten Integer der Liste `ids` hinzu.
- Ausgabe:** Die extrahierten IDs werden auf der Konsole ausgegeben.

Du kannst die Methode `extractIds` in deinem `Graph`-Klasse verwenden, um die IDs von `Container`-Objekten zu extrahieren.



die methode soll nur eine Id zurückgeben



Wenn du eine Methode benötigst, die nur eine einzelne ID für einen gegebenen Namen zurückgibt, kannst du die Methode wie folgt anpassen. Diese Methode nimmt einen `String` als Eingabeparameter und gibt die extrahierte ID als `Integer` zurück:

```
java

public class ContainerIdExtractor {
    public static void main(String[] args) {
        String name = "cont100";
        int id = extractId(name);

        // Ausgabe der extrahierten ID
        System.out.println(id);
    }

    public static int extractId(String name) {
        // Entferne "cont" und parse die restliche Zahl als Integer
        String idString = name.replace("cont", "");
        return Integer.parseInt(idString);
    }
}
```

Erklärungen:

- extractId-Methode:**
 - Nimmt einen `String`-Namen als Parameter entgegen.
 - Entfernt "cont" aus dem Namen.
 - Konvertiert den verbleibenden String in einen `Integer` und gibt ihn zurück.
- Hauptmethode:**
 - Hier wird ein Beispielname (z.B. `cont100`) verwendet, und die Methode `extractId` wird aufgerufen, um die ID zu extrahieren.

So kannst du für jeden Namen, den du übergibst, die entsprechende ID erhalten.

⚠️ Ab hier sind Nachrichten nur für dich sichtbar

ChatGPT

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



Explain how the renderGrid works:

```
import React, { useState, useEffect } from "react";
import SearchBar from './SearchBar';
import { useNavigate } from 'react-router-dom';
import ContainerChooser from './dialogs/ContainerChooser';
import axiosInstance from '../api/AxiosInstance';
import ContainerDistributer from './util/ContainerDistributer'

const Workspace = ({ gridSize, ship }) => {
  const navigate = useNavigate();
  const [hoveredDiv, setHoveredDiv] = useState(null);
  const [isDialogOpen, setIsDialogOpen] = useState(false);
  const [selectedId, setSelectedId] = useState(null);
  const [containerDistribution, setContainerDistribution] = useState([]);
  const [containerIds, setContainerIds] = useState([]);

  const handleOpenDialog = () => setIsDialogOpen(true);
  const handleCloseDialog = () => setIsDialogOpen(false);

  useEffect(() => {
    const fetchContainerIdsOfShip = async () => {
      try {
        const accessToken = localStorage.getItem("accessToken");
        const shipId = ship.ship_id;
        const containerIdsResponse = await axiosInstance.get(`rest/ship/${shipId}/containers`);
        const headers: {
          'authorization': `Bearer ${accessToken}`
        };
        const fetchedIds = containerIdsResponse.data?.containers || [];
        if (!Array.isArray(fetchedIds)) {
          console.error("Fetched data is not an array:", fetchedIds);
          return;
        }
        setContainerIds(fetchedIds);
      } catch (error) {
        console.error("Failed to fetch containers of ship:", error.message);
      }
    }
    if (gridSize.length > 0) {
      setContainerDistribution(ContainerDistributer(gridSize.rows, gridSize.cols, fetchedIds.length));
    }
  }, []);

  return (
    <div>
      <div>
        <div>
          <div>
            <div>
              <div>
                <div>
                  <div>
                    <div>
                      <div>
                        <div>
                          <div>
                            <div>
                              <div>
                                <div>
                                  <div>
                                    <div>
                                      <div>
                                        <div>
                                          <div>
                                            <div>
                                              <div>
                                                <div>
                                                  <div>
                                                    <div>
                                                      <div>
                                                        <div>
                                                          <div>
                                                            <div>
                                                              <div>
                                                                <div>
                                                                  <div>
                                                                    <div>
                                                                      <div>
                                                                        <div>
                                                                          <div>
                                                                            <div>
                                                                              <div>
                                                                                <div>
                                                                                  <div>
                                                                                    <div>
                                                                                      <div>
                        </div>
                      </div>
                    </div>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}
```

```

}

fetchContainerIdsOfShip();

// Cleanup for body scrolling
document.body.style.overflow = 'hidden';
return () => {
  document.body.style.overflow = '';
};
}, [ship, gridSize])

//needs to be changed
const handleSelect = (value) => {
  setSelectedId(value);
  setDialogOpen(false);
  navigate(`detail/${value}`);
};

const renderGrid = () => {
  if (gridSize.rows === 0 || gridSize.cols === 0 || containerDistribution.length === 0) {
    return null;
  }

  const divs = [];
  for (let row = 0; row < gridSize.rows; row++) {
    const rowDivs = [];
    for (let col = 0; col < gridSize.cols; col++) {
      const key = `${row}-${col}`; // Unique key for each div
      rowDivs.push(
        <div
          key={key}
          className="relative flex justify-center items-center pl-10 pr-10"
          onClick={() => setDialogOpen(true)}
          onMouseEnter={() => setHoveredDiv(key)}
          onMouseLeave={() => setHoveredDiv(null)}
        >
          
          <div className="w-16 h-5 absolute flex justify-center items-center">
            <p className={`${$hoveredDiv === key ? 'font-bold' : 'font-sans'}`}>-10</p> /*-10 is still hardcoded*/
          </div>
        </div>
      );
    }
    divs.push(
      <div key={row} className="flex space-x-2"/* space-x-2 for horizontal spacing */>
        {rowDivs}
      </div>
    );
  }
  return divs;
};

```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

2/11

```

return (
  <div className="flex-grow flex flex-col justify-center items-center p-5">
    {renderGrid}
    <ContainerChooser
      open={isDialogOpen}
      onClose={handleCloseDialog}
      onSelect={handleSelect}
      values={[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]} //temporary!!!
    />
  </div>
);
};

export default Workspace;

```



The `renderGrid` function in this component is responsible for rendering a grid of div elements based on the `gridsize` prop, which includes the number of rows and columns. Here's a breakdown of how it works:

1. Input Validation:

- The function first checks if `gridsize.rows`, `gridsize.cols`, or `containerDistribution.length` are zero or undefined. If any of these conditions are true, the function returns `null`, meaning no grid is rendered.

2. Grid Rendering:

- An empty array `divs` is created to hold the rows of the grid.
- The outer loop iterates over `gridsize.rows`, representing each row of the grid.
- Inside the outer loop, another array `rowDivs` is created to hold the individual cells (divs) of the current row.

3. Cell Creation:

- For each row and column, the function generates a unique `key` based on the row and column indices, such as `0-0`, `0-1`, etc.
- Each cell is a `div` element styled with `relative`, `flex`, and other Tailwind CSS classes. This `div` acts as a container for an image and a label.
- The `onClick` handler opens a dialog (`setDialogOpen(true)`) when a cell is clicked.
- The `onMouseEnter` and `onMouseLeave` handlers update the state `hoveredDiv` to highlight the cell when the mouse is hovering over it, allowing dynamic changes in the class of the label inside each cell.

4. Image and Label:

- Each cell contains an `img` element with a source pointing to `/src/img/Container.svg`, which represents a container.
- There is also a `div` that displays a label (`0-0` in the example, though this is hardcoded) inside the cell, centered both vertically and horizontally. The font weight of the label changes depending on whether the cell is being hovered over or not.

5. Row Structure:

- Once all the columns for a row are created, the `rowDivs` array is pushed to the `divs` array.

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

3/11

- After all rows are processed, the `divs` array is returned, effectively creating the entire grid of containers.
- 6. Return:**
- The `renderGrid` function returns a series of rows, each containing the container divs. This is what is rendered inside the main container of the component.

Example Output:

- If `gridsize.rows = 3` and `gridsize.cols = 3`, the function will render a 3x3 grid of cells. Each cell contains an image of a container and a label. When you hover over a cell, the label font will become bold. Clicking on a cell will open a dialog to select a container.

Dialog Interaction:

- The `ContainerChooser` dialog is shown when a user clicks on a grid cell, and it provides an interface to select a container. This is controlled by the state `isDialogOpen` and functions `handleOpenDialog` and `handleCloseDialog` .

The function is modular, dynamically adjusting to the grid size and container distribution passed to it, allowing the grid to be flexible and interactive.



In what way is a 2x2 grid rendered?
(1st) (2nd)
(3rd) (4th)

or

(1st) (3rd)
(2nd) (4th)



In the `renderGrid` function, the grid is rendered in a row-by-row manner. Given a `2x2` grid (2 rows and 2 columns), it will be rendered as:

scss

(1st) (2nd)
(3rd) (4th)

Explanation:

- The outer loop iterates over the `rows` (in this case, 2), and for each row, an inner loop iterates over the `columns` (also 2).
- The `key` for each grid item is generated using a combination of the row and column indices, like `\${row}-\${col}`. For a 2x2 grid, this results in:
 - Row 0, Column 0 -> Key `0-0`

- Row 0, Column 1 -> Key `0-1`
- Row 1, Column 0 -> Key `1-0`
- Row 1, Column 1 -> Key `1-1`

Thus, the grid is structured like:

```

mathematica

Row 0: (0-0) (0-1)
Row 1: (1-0) (1-1)

```

Which corresponds to:

```

scss

(1st) (2nd)
(3rd) (4th)

```

This rendering pattern follows the row-major order (rows are filled sequentially from left to right).



Can you change the renderGrid so that it functions like this one?

```

import React, { useState, useEffect } from 'react';
import ContainerDistributor from './util/ContainerDistributor';

const TestingPage = () => {
  const [containerDistribution, setContainerDistribution] = useState([]);

  useEffect(() => {
    // Container IDs (for example, [1, 2, 3, 4])
    const containerIds = [1, 2, 3, 4];

    // Using ContainerDistributor to get the 2x2 grid distribution
    const distribution = ContainerDistributor(2, 2, containerIds.length);

    setContainerDistribution(distribution);
  }, []);

  const renderGrid = () => {
    if (containerDistribution.length === 0) {
      return <div>Loading...</div>; // Show loading until distribution is ready
    }

    const divs = [];
    for (let row = 0; row < containerDistribution.length; row++) {
      const rowDivs = [];
      for (let col = 0; col < containerDistribution[row].length; col++) {
        const containerId = containerDistribution[row][col];
      }
    }
  }
};

const renderGrid = () => {
  if (containerDistribution.length === 0) {
    return <div>Loading...</div>; // Show loading until distribution is ready
  }

  const divs = [];
  for (let row = 0; row < containerDistribution.length; row++) {
    const rowDivs = [];
    for (let col = 0; col < containerDistribution[row].length; col++) {
      const containerId = containerDistribution[row][col];
    }
  }
};


```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

5/11

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

4/11

```

rowDivs.push(
  <div
    key={`${row}-${col}`}
    className="relative flex justify-center items-center p-10 border border-gray-300"
  >
    <div className="w-16 h-5 flex justify-center items-center">
      <p>{containerId} || "Empty"</p> {/* Display container ID or "Empty" */}
    </div>
  </div>
);
}
divs.push(
  <div key={row} className="flex space-x-2">
    {rowDivs}
  </div>
);
}
return divs;
};

return (
  <div className="flex-grow flex flex-col justify-center items-center p-5">
    <h2>2x2 Grid with Containers</h2>
    <div className="grid-container">{renderGrid}</div>
  </div>
);
};

export default TestingPage;

```



To modify your original `renderGrid` function to behave like the one in `TestingPage`, we need to follow a similar approach of managing the container distribution using a 2D array. In this case, `ContainerDistributor` will be used to distribute the containers across the grid in the same row-major order.

Here's the updated version of `renderGrid`:

```

javascript

import React, { useState, useEffect } from "react";
import ContainerDistributor from "./util/ContainerDistributor";
import { useNavigate } from "react-router-dom";
import axiosInstance from "../api/AxiosInstance";
import ContainerChooser from "../dialogs/ContainerChooser";

const Workspace = ({ gridSize, ship }) => {
  const navigate = useNavigate();
  const [hoveredDiv, setHoveredDiv] = useState(null);
  const [isDialogOpen, setDialogOpen] = useState(false);
  const [selectedId, setSelectedId] = useState(null);
  const [containerDistribution, setContainerDistribution] = useState([]);
  const [containerIds, setContainerIds] = useState([]);

  const handleOpenDialog = () => setDialogOpen(true);
  const handleCloseDialog = () => setDialogOpen(false);
  const handleSelect = (value) => {
    setSelectedId(value);
    setDialogOpen(false);
    navigate(`/detail/${value}`);
  };
  const renderGrid = () => {
    if (containerDistribution.length === 0) {
      return <div>Loading...</div>; // Show loading until distribution is ready
    }
    const divs = [];
    for (let row = 0; row < containerDistribution.length; row++) {
      const rowDivs = [];
      for (let col = 0; col < containerDistribution[row].length; col++) {
        const containerId = containerDistribution[row][col];
        rowDivs.push(
          <div
            key={`${row}-${col}`}
            className="relative flex justify-center items-center p-10 border border-gray-300"
            onClick={() => handleOpenDialog()}
            onMouseEnter={() => setHoveredDiv(`${row}-${col}`)}
            onMouseLeave={() => setHoveredDiv(null)}
          >
            
            <div className="w-16 h-5 absolute flex justify-center items-center">
              <p>{containerId} || "Empty"</p>
            </div>
          </div>
        );
      }
      divs.push(
        <div key={row} className="flex space-x-2">
          {rowDivs}
        </div>
      );
    }
    return divs;
  };
}

return (
  <div className="flex-grow flex flex-col justify-center items-center p-5">
    <h2>2x2 Grid with Containers</h2>
    <div className="grid-container">{renderGrid}</div>
    <ContainerChooser
      open={isDialogOpen}
      onClose={handleCloseDialog}
      onSelect={handleSelect}
      values={[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]} // temporary!!!
    />
  </div>
);
};

export default Workspace;

```

Key Changes:

- Container Distribution:**
 - Instead of using a `gridSize.rows` and `gridSize.cols` to directly control the grid, we now depend on the result of `ContainerDistributor` to give us a 2D array, which represents the container placement across rows and columns.
- Rendering the Grid:**
 - The grid is now rendered dynamically based on the container distribution (`containerDistribution`). It iterates over the rows and columns of the 2D array.
 - Each cell in the grid either displays a `containerId` or "Empty" if no container is placed there.
- Event Handlers:**
 - `onClick`, `onMouseEnter`, and `onMouseLeave` have been added to the individual grid cells to open the dialog and manage hover effects.

Result:

This updated `renderGrid` will now display a grid with containers distributed according to the 2D array returned by `ContainerDistributor`. If the container ID is available for a grid cell, it will show that ID; otherwise, it will display "Empty". It functions similarly to the `TestingPage` but incorporates your dynamic container fetching and distribution logic.



No but the div with the picture should still be depicted. Just instead of the "1-10" the count of containers

```

const handleOpenDialog = () => setDialogOpen(true);
const handleCloseDialog = () => setDialogOpen(false);

useEffect(() => {
  const fetchContainerIdsOfShip = async () => {
    try {
      const accessToken = localStorage.getItem("accessToken");
      const shipId = ship.ship_id;
      const containerIdsResponse = await axiosInstance.get(`/rest/ship/${shipId}/containers`, {
        headers: {
          authorization: `Bearer ${accessToken}`
        }
      });
      const fetchedIds = containerIdsResponse.data?.containers || [];
      if (!Array.isArray(fetchedIds)) {
        console.error("Fetched data is not an array:", fetchedIds);
        return;
      }
      setContainerIds(fetchedIds);
    } catch (error) {
      console.error("Failed to fetch containers of ship:", error.message);
    }
  };
  fetchContainerIdsOfShip();
}, []);

// Cleanup for body scrolling
document.body.style.overflow = "hidden";
return () => {
  document.body.style.overflow = "";
}, [ship, gridSize]);

const handleSelect = (value) => {
  setSelectedId(value);
  setDialogOpen(false);
  navigate(`/detail/${value}`);
};

const renderGrid = () => {
  if (containerDistribution.length > 0) {
    setContainerDistribution(
      ContainerDistributor(gridSize.rows, gridSize.cols, fetchedIds.length)
    );
  } catch (error) {
    console.error("Failed to fetch containers of ship:", error.message);
  }
  fetchContainerIdsOfShip();
};

// Cleanup for body scrolling
document.body.style.overflow = "hidden";
return () => {
  document.body.style.overflow = "";
}, [ship, gridSize]);

```

Printed using ChatGPT to PDF powered by PDFCrowd [HTML to PDF API](#).

6/11

Printed using ChatGPT to PDF powered by PDFCrowd [HTML to PDF API](#).

7/11

cell, we can adjust the label inside each cell to display the number of containers rather than the placeholder text.

Updated `renderGrid`:

In this version, we will count the number of containers in each grid cell (based on the `containerDistribution`) and display that count instead of the hardcoded label:

```

javascript

import React, { useState, useEffect } from "react";
import ContainerDistributor from "./util/ContainerDistributor";
import { useNavigate } from "react-router-dom";
import axiosInstance from "../api/AxiosInstance";
import ContainerChooser from "../dialogs/ContainerChooser";

const Workspace = ({ gridSize, ship }) => {
  const navigate = useNavigate();
  const [hoveredDiv, setHoveredDiv] = useState(null);
  const [isDialogOpen, setDialogOpen] = useState(false);
  const [selectedId, setSelectedId] = useState(null);
  const [containerDistribution, setContainerDistribution] = useState([]);
  const [containerIds, setContainerIds] = useState([]);

  const handleOpenDialog = () => setDialogOpen(true);
  const handleCloseDialog = () => setDialogOpen(false);

  useEffect(() => {
    const fetchContainerIdsOfShip = async () => {
      try {
        const accessToken = localStorage.getItem("accessToken");
        const shipId = ship.ship_id;
        const containerIdsResponse = await axiosInstance.get(`/rest/ship/${shipId}/containers`, {
          headers: {
            authorization: `Bearer ${accessToken}`
          }
        });
        const fetchedIds = containerIdsResponse.data?.containers || [];
        if (!Array.isArray(fetchedIds)) {
          console.error("Fetched data is not an array:", fetchedIds);
          return;
        }
        setContainerIds(fetchedIds);
      } catch (error) {
        console.error("Failed to fetch containers of ship:", error.message);
      }
    };
    fetchContainerIdsOfShip();
  }, []);

  // Cleanup for body scrolling
  document.body.style.overflow = "hidden";
  return () => {
    document.body.style.overflow = "";
  }, [ship, gridSize]);

  const handleSelect = (value) => {
    setSelectedId(value);
    setDialogOpen(false);
    navigate(`/detail/${value}`);
  };

  const renderGrid = () => {
    if (containerDistribution.length > 0) {
      setContainerDistribution(
        ContainerDistributor(gridSize.rows, gridSize.cols, fetchedIds.length)
      );
    } catch (error) {
      console.error("Failed to fetch containers of ship:", error.message);
    }
    fetchContainerIdsOfShip();
  };

  // Cleanup for body scrolling
  document.body.style.overflow = "hidden";
  return () => {
    document.body.style.overflow = "";
  }, [ship, gridSize]);

```

Printed using ChatGPT to PDF powered by PDFCrowd [HTML to PDF API](#).

8/11

Printed using ChatGPT to PDF powered by PDFCrowd [HTML to PDF API](#).

```

    navigate(`/detail/${value}`);
};

const renderGrid = () => {
  if (containerDistribution.length === 0) {
    return <div>Loading...</div>; // Show loading until distribution is ready
  }

  const divs = [];
  for (let row = 0; row < containerDistribution.length; row++) {
    const rowDivs = [];
    for (let col = 0; col < containerDistribution[row].length; col++) {
      const containerIdsInCell = containerDistribution[row][col];
      const containerCount = containerIdsInCell ? containerIdsInCell.length : 0; // Count of containers in this cell
      rowDivs.push(
        <div key={`${row}-${col}`} className="relative flex justify-center items-center p-10 border border-gray-300">
          onClick={() => setDialogOpen(true)}
          onMouseEnter={() => setHoveredDiv(`${row}-${col}`)}
          onMouseLeave={() => setHoveredDiv(null)}
        >
          
          <div className="w-16 h-5 absolute flex justify-center items-center">
            <p className={ `${hoveredDiv} === `${row}-${col}` ? 'font-bold' : 'font-sans' }>
              {containerCount > 0 ? containerCount : 'Empty'}
            </p>
          </div>
        </div>
      );
    }
    divs.push(
      <div key={row} className="flex space-x-2">
        {rowDivs}
      </div>
    );
  }
  return divs;
};

return (
  <div className="flex-grow flex flex-col justify-center items-center p-5">
    <h2>Grid with Containers</h2>
    <div className="grid grid-container">{renderGrid()}</div>
    <ContainerChooser
      open={isDialogOpen}
      onClose={handleCloseDialog}
      onSelect={handleSelect}
      values={[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]} // temporary!!!
    />
  </div>
);
};

export default Workspace;

```

Key Changes:

1. Container Count per Grid Cell:

- Instead of displaying a hardcoded label like "0-10", the number of containers in each cell is calculated using `containerIdsInCell.length`. This is based on the assumption that `ContainerDistributor` returns a structure where each grid cell can contain multiple container IDs.

2. Empty Cells:

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

10/11

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

11/11

ChatGPT

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



Hallo! Bitte nimm folgende Methode und ändere sie folgendermaßen um:
-> Neuer Name parseSpecificToJson
-> anstatt alle Container zu parsen wird ein container Objekt 'origin' und ein integer namens length übergeben
-> je nachdem wie lange der integer ist, soll rekursiv der container und seine subContainer hergenommen werden und immer wieder hinzugefügt werden
-> bsp: ein container hat 2 subcontainer, ist die tiefe 2 dann wird der container (0) seine subContainer (1) und die subcontainer der subcontainer (2) in das JSON hinzugefügt werden!
-> bitte beachte, dass die Methode innerhalb einer "Graph" Klasse liegt
-> beachte auch dass die getAllSubs null oder ein jsonArray zurückgibt
-> alles sollte innerhalb einer Methode passieren
public JSONObject parseAllContainersToJson(){
 Container[] containersArray = containers.toArray(new Container[0]);
 JSONArray containersJSON = new JSONArray();

 for(Container c : containers){
 JSONObject containerJSONObject = new JSONObject();
 containerJSONObject.put("contId", c.getName());

 if(getAllSubs(c) != null){
 JSONArray subsOfC = getAllSubs(c);
 containerJSONObject.put("subs", subsOfC);
 }
 containersJSON.put(containerJSONObject);
 }

 JSONObject finalJSON = new JSONObject();
 finalJSON.put("containers", containersJSON);
 return finalJSON;
}



Hier ist die angepasste Methode 'parseSpecificToJson', die rekursiv den Container und seine Subcontainer in das JSON hinzufügt, basierend auf der übergebenen Tiefe:

```

public JSONObject parseSpecificToJson(Container origin, int length){
  // Erstelle das JSON-Objekt für den Ursprung-Container
  JSONObject originJSONObject = new JSONObject();
  originJSONObject.put("contId", origin.getName());

```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).



- If there are no containers in a grid cell, it displays "Empty". Otherwise, it shows the number of containers in that cell.

3. Display:

- For each cell, the 'img' is still rendered, and the count of containers or "Empty" is displayed as the label. The font weight changes on hover as before, using the 'hoveredDiv' state to apply a 'font-bold' class.

Result:

- The grid will show the image for each container and the number of containers in each cell (or "Empty" if no containers are present).
- The hover effect and the dialog interaction remain the same, allowing users to interact with the grid.

Ab hier sind Nachrichten nur für dich sichtbar

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

10/11

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

11/11



// Wenn die Tiefe größer als 0 ist, rekursiv die Subcontainer hinzufügen
if (length > 0) {
 JSONArray subArray = new JSONArray();

```

  // Hole alle Subcontainer des aktuellen Containers
  JSONArray subsofOrigin = getAllSubs(origin);

  // Wenn es Subcontainer gibt, füge sie rekursiv hinzu
  if (subsofOrigin != null) {
    for (Object sub : subsofOrigin) {
      Container subContainer = (Container) sub;
      // Rufe rekursiv die Methode für jeden Subcontainer auf, verringere die Tiefe um 1
      JSONObject subContainerJSON = parseSpecificToJson(subContainer, length - 1);
      subArray.put(subContainerJSON);
    }
    originJSONObject.put("subs", subArray);
  }
}

return originJSONObject;
}
```

Erklärung:

- Die Methode nimmt einen 'Container origin' und eine 'int length', die die Rekursionstiefe angibt.
- Die rekursive Logik prüft, ob die Tiefe ('length') größer als 0 ist. Falls ja, werden die Subcontainer des Containers ('getAllSubs(origin)') abgerufen.
- Für jeden Subcontainer wird die Methode erneut aufgerufen und die Tiefe um 1 verringert, bis die Tiefe auf 0 reduziert ist.
- Am Ende wird das JSON des Containers mit seinen Subcontainern und deren Subcontainern erstellt.

Du kannst die Methode dann folgendermaßen verwenden, um sie auf einen Container anzuwenden und die gewünschte Tiefe zu setzen:

java

```
JSONObject resultJSON = parseSpecificToJson(originContainer, 2); // Beispiel für Tiefe 2
```

Dabei wird der 'originContainer' und seine Subcontainer bis zur Tiefe 2 in das JSON eingebaut.



Exception in thread "main" java.lang.ClassCastException: class org.json.JSONObject cannot be cast to class org.example.Container (org.json.JSONObject and org.example.Container are in unnamed module of loader 'app')

```
at org.example.Graph.parseSpecificToJson(Graph.java:155)
at org.example.Graph.exportToJsonFile(Graph.java:214)
at org.example.Main.main(Main.java:97)
```

Das Casten wirft einen Fehler!



Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

1/5

2/5

Der Fehler tritt auf, weil versucht wird, ein `JSONObject` in einen `Container` zu casten. Dies passiert vermutlich in der folgenden Zeile:

```
java
Container subContainer = (Container) sub;
```

Der Fehler bedeutet, dass das `sub`-Objekt, das aus der Methode `getAllSubs(origin)` stammt, kein `Container`-Objekt ist, sondern ein `JSONObject`. Wir müssen sicherstellen, dass wir nur die tatsächlichen `Container`-Objekte durchlaufen und nicht die `JSONObject`-Instanzen. Hier ist eine angepasste Version der Methode, die den Fehler beheben sollte:

Lösung:

1. Wenn `getAllSubs` ein `JSONArray` zurückgibt, das Objekte vom Typ `JSONObject` enthält, müssen wir sicherstellen, dass wir entweder die `Container`-Instanzen direkt abrufen oder die notwendigen Daten in `Container`-Objekte umwandeln.
2. Falls `getAllSubs` nur `Container`-Objekte zurückgibt, sollten wir den Cast beibehalten. Falls sie jedoch `JSONObject`-Instanzen zurückgibt, müssen wir diese in `Container`-Instanzen umwandeln.

Hier ist die angepasste Methode:

```
java
public JSONObject parseSpecificToJson(Container origin, int length) {
    // Erstelle das JSON-Objekt für den Ursprung-Container
    JSONObject originJsonObject = new JSONObject();
    originJsonObject.put("contId", origin.getName());

    // Wenn die Tiefe größer als 0 ist, rekursiv die Subcontainer hinzufügen
    if (length > 0) {
        JSONArray subsArray = new JSONArray();
        // Hole alle Subcontainer des aktuellen Containers
        JSONArray subsOfOrigin = getAllSubs(origin);

        // Wenn es Subcontainer gibt, füge sie rekursiv hinzu
        if (subsOfOrigin != null) {
            for (Object sub : subsOfOrigin) {
                // Wenn sub ein JSONObject ist, müssen wir es zu einem Container umwandeln
                if (sub instanceof JSONObject) {
                    // Beispiel: Umwandlung von JSONObject zu Container
                    JSONObject subJsonObject = (JSONObject) sub;
                    Container subContainer = convertJsonToContainer(subJsonObject);

                    // Rufe rekursiv die Methode für den Subcontainer auf
                    subContainerJson = parseSpecificToJson(subContainer, length - 1);
                    subsArray.put(subContainerJson);
                } else if (sub instanceof Container) {
                    // Falls es bereits ein Container ist, direkt verwenden
                    Container subContainer = (Container) sub;
                    JSONObject subContainerJson = parseSpecificToJson(subContainer, length - 1);
                    subsArray.put(subContainerJson);
                }
            }
            originJsonObject.put("subs", subsArray);
        }
    }
}
```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

3/5

```
return originJsonObject;
}

// Hilfsmethode zum Umwandeln eines JSONObject in einen Container
private Container convertJsonToContainer(JSONObject json) {
    // Hier kannst du die Umwandlung von einem JSONObject zu einem Container anpassen
    String name = json.getString("contId");
    return new Container(name); // Beispiel für eine einfache Umwandlung
}
```

Änderungen:

- Umwandlung von `JSONObject` zu `Container`:** Falls `sub` ein `JSONObject` ist, wird es nun mit einer Hilfsmethode `convertJsonToContainer` in ein `Container`-Objekt umgewandelt. Du kannst diese Methode je nach den Daten in deinem `JSONObject` anpassen, um den richtigen Container zu erstellen.
- Überprüfung des Typs von `sub`:** Wir prüfen, ob `sub` ein `JSONObject` oder ein `Container` ist, um den richtigen Umgang mit den Objekten sicherzustellen.

Mit dieser Anpassung solltest du nun den Fehler beim Casting vermeiden können.



Schreibe nun bitte noch folgende Methode:
void exportToJsonFile(boolean specific, int depth, Container spc)

Welche wenn specific TRUE ist: die parseSpecificToJson mit den übergebenen Werten aufruft und in ein File namens "graphSpecific.json" das rückgegebene Json Objekt schreibt

Wenn specific auf FALSE ist: die parseAllContainersToJson aufruft und dessen Rückgabe in "graph.json" schreibt



Hier ist die Methode `exportToJsonFile`, die je nach Wert von `specific` entweder die Methode `parseSpecificToJson` oder `parseAllContainersToJson` aufruft und das zurückgegebene JSON in die entsprechenden Dateien schreibt:

java

```
import java.io.FileWriter;
import java.io.IOException;

public void exportToJsonFile(boolean specific, int depth, Container spc) {
    JSONObject jsonToExport;

    // Wenn specific TRUE ist, rufe die parseSpecificToJson Methode auf
    if (specific) {
        jsonToExport = parseSpecificToJson(spc, depth);

        // Schreibe das JSON in eine Datei namens "graphSpecific.json"
        try (FileWriter file = new FileWriter("graphSpecific.json")) {
            file.write(jsonToExport.toString(4)); // 4 ist die Indentation für das Format
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

4/5

ChatGPT

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



Why is the table data not showing the updates after the fetching??

```
import React, { useState, useEffect } from 'react';
import { useParams } from 'react-router-dom';
import axiosInstance from '../api/AxiosInstance';

const Detailspace = () => {

    const {spcId, containerId} = useParams();

    const [container, setContainer] = useState(null);
    const [combinedSerialNumber, setCombinedSerialNumber] = useState("Loading...");
    const [notes, setNotes] = useState("");

    const [temperature, setTemperature] = useState("");
    const [pressure, setPressure] = useState("");
    const [humidity, setHumidity] = useState("");
    const [vibration, setVibration] = useState("");
    const [altitude, setAltitude] = useState("");
    const [latitude, setLatitude] = useState("");
    const [longitude, setLongitude] = useState("");

    const [saved, setSaved] = useState(false);
    const [notSaved, setNotSaved] = useState(false); // If the save fails this is used

    const tableData = [
        {environment: "Temperature", value: "0C", level: "", problem: ""},
        {environment: "Pressure", value: "Pa", level: "Critical", problem: "High Pressure"},
        {environment: "Humidity", value: "%", level: "", problem: ""},
        {environment: "Vibration", value: "m/s2", level: "", problem: ""},
        {environment: "Altitude", value: "m", level: "", problem: ""},
        {environment: "Latitude", value: "DD", level: "", problem: ""},
        {environment: "Longitude", value: "DD", level: "", problem: ""}
    ];

    const updateTableData = (newValue, sensor) => {
        return tableData.map(item => {
            switch(sensor) {
                case "temperature":
                    if (item.environment === "Temperature") {
                        return { ...item, value: newValue };
                    }
                    break;
            }
        });
    }
}
```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

1/8

Erklärung:

- Die Methode überprüft den Wert von `specific`. Wenn `specific` true ist, wird die Methode `parseSpecificToJson(spc, depth)` aufgerufen und das Ergebnis in einer Datei namens "graphSpecific.json" gespeichert.
- Wenn `specific` false ist, wird die Methode `parseAllContainersToJson()` aufgerufen und das Ergebnis in der Datei "graph.json" gespeichert.
- Das JSON wird mit einer Indentation von 4 Leerzeichen formatiert, um es leserlicher zu machen.
- Ein `FileWriter` wird verwendet, um das JSON in die entsprechende Datei zu schreiben. Die Methode ist in einem `try-with-resources`-Block enthalten, um sicherzustellen, dass der `FileWriter` korrekt geschlossen wird, auch wenn eine Ausnahme auftritt.

Du kannst nun `exportToJsonFile(true, 2, someContainer)` oder `exportToJsonFile(false, 0, null)` aufrufen, je nachdem, welche Variante du exportieren möchtest.

Ab hier sind Nachrichten nur für dich sichtbar

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

5/5

```

case "air_pressure":
  if (item.environment === "Pressure") {
    return { ...item, value: newValue };
  }
  break;
case "humidity":
  if (item.environment === "Humidity") {
    return { ...item, value: newValue };
  }
  break;
case "vibration":
  if (item.environment === "Vibration") {
    return { ...item, value: newValue };
  }
  break;
case "altitude":
  if (item.environment === "Altitude") {
    return { ...item, value: newValue };
  }
  break;
case "latitude":
  if (item.environment === "Latitude") {
    return { ...item, value: newValue };
  }
  break;
case "longitude":
  if (item.environment === "Longitude") {
    return { ...item, value: newValue };
  }
  break;
default:
  return item;
}
return item; // Return the item unchanged if no conditions match
};

const handleNotesChange = (event) => {
  setNotes(event.target.value);
};

const handleBlur = async () => {
  if (!container) return;

  try {
    const accessToken = localStorage.getItem("accessToken");
    const updatedContainer = { ...container, notes };

    const updateResponse = await axiosInstance.put(` /rest/container/${containerId}`, updatedContainer, {
      headers: {
        'authorization': `Bearer ${accessToken}`
      }
    });
    const vibrationValue = updateResponse.data.vibration[0].value;
    setVibration(vibrationValue);

    // Altitude
    const altitudeResponse = await axiosInstance.get(` /rest/sensor/altitude/${shipId}/${containerId}/latest`, {
      headers: {
        'authorization': `Bearer ${accessToken}`
      }
    });
    const altitudeValue = altitudeResponse.data.altitude[0].value;
    setAltitude(altitudeValue);

    // Latitude
    const latitudeResponse = await axiosInstance.get(` /rest/sensor/latitude/${shipId}/${containerId}/latest`, {
      headers: {
        'authorization': `Bearer ${accessToken}`
      }
    });
    const latitudeValue = latitudeResponse.data.latitude[0].value;
    setLatitude(latitudeValue);

    // Longitude
    const longitudeResponse = await axiosInstance.get(` /rest/sensor/longitude/${shipId}/${containerId}/latest`, {
      headers: {
        'authorization': `Bearer ${accessToken}`
      }
    });
    const longitudeValue = longitudeResponse.data.longitude[0].value;
    setLongitude(longitudeValue);

    updateTableData();
  } catch (error) {
    console.error('Error fetching data:', error);
  }
};

fetchContainerData();
}, [containerId, shipId])

```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

2/8

```

  authorization: `Bearer ${accessToken}`
};

if (updateResponse.status === 204) {
  setSaved(true);
  setTimeout(() => { setSaved(false); }, 2000);
} else {
  setNotSaved(true);
  setTimeout(() => { setNotSaved(false); }, 2000);
}

} catch (error) {
  console.error("Failed to save notes:", error.message);
}

// Fetch Data
useEffect(() => {
  const fetchContainerData = async () => {
    try {
      const accessToken = localStorage.getItem("accessToken");
      let sensor = "";
      // Temperature
      const temperatureResponse = await axiosInstance.get(` /rest/sensor/temperature/${shipId}/${containerId}/latest`, {
        headers: {
          'authorization': `Bearer ${accessToken}`
        }
      });
      const temperatureValue = temperatureResponse.data.temperature[0].value;
      sensor = temperatureResponse.data.temperature[0].sensor;
      updateTableData(temperatureValue, sensor);

      // Air Pressure
      const airPressureResponse = await axiosInstance.get(` /rest/sensor/air_pressure/${shipId}/${containerId}/latest`, {
        headers: {
          'authorization': `Bearer ${accessToken}`
        }
      });
      const airPressureValue = airPressureResponse.data.air_pressure[0].value;
      setPressure(airPressureValue);

      // Humidity
      const humidityResponse = await axiosInstance.get(` /rest/sensor/humidity/${shipId}/${containerId}/latest`, {
        headers: {
          'authorization': `Bearer ${accessToken}`
        }
      });
      const humidityValue = humidityResponse.data.humidity[0].value;
      setHumidity(humidityValue);
    } catch (error) {
      console.error("Failed to fetch sensor data:", error.message);
    }
  };
  fetchContainerData();
};

// Fetch combined Container Serial Number when id changes
useEffect(() => {
  const fetchSerialNumberOfContainer = async () => {
    try {
      const accessToken = localStorage.getItem("accessToken");
      const containerSerialNumberResponse = await axiosInstance.get(` /rest/container/${containerId}/serial-number`, {
        headers: {
          'authorization': `Bearer ${accessToken}`
        }
      });
      const fetchedSerialNumber = containerSerialNumberResponse.data.serial_number;
      setCombinedSerialNumber(fetchedSerialNumber);
    } catch (error) {
      console.error("Failed to fetch Serial Numbers of containers:", error.message);
    }
  };
  fetchSerialNumberOfContainer();
}, [containerId])

// Fetch entire Container by ID when the id changes
useEffect(() => {
  const fetchContainerData = async () => {
    try {
      const accessToken = localStorage.getItem("accessToken");

      const containerResponse = await axiosInstance.get(` /rest/container/${containerId}`);
      const containerResponseData = containerResponse.data.container[0];
      setContainer(containerResponseData);
      setNotes(containerResponseData.notes);

    } catch (error) {
      console.error("Failed to fetch container data:", error.message);
    }
  };
  fetchContainerData();
}, [containerId]);

return (
  <div className="flex-grow flex justify-center items-center pt-5 pb-5 pl-24 pr-24 gap-10">
    <div className="h-5/6 w-3/4 p-10">
      <h1 className="text-2xl font-bold border-2 border-black p-1 mb-2">Container
      {combinedSerialNumber}</h1>
      <table className="w-full border-collapse border-2 border-black">

```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

3/8

```

const humidityValue = humidityResponse.data.humidity[0].value;
setHumidity(humidityValue);

// Air Pressure
const vibrationResponse = await axiosInstance.get(` /rest/sensor/vibration/${shipId}/${containerId}/latest`, {
  headers: {
    'authorization': `Bearer ${accessToken}`
  }
});
const vibrationValue = vibrationResponse.data.vibration[0].value;
setVibration(vibrationValue);

// Altitude
const altitudeResponse = await axiosInstance.get(` /rest/sensor/altitude/${shipId}/${containerId}/latest`, {
  headers: {
    'authorization': `Bearer ${accessToken}`
  }
});
const altitudeValue = altitudeResponse.data.altitude[0].value;
setAltitude(altitudeValue);

// Latitude
const latitudeResponse = await axiosInstance.get(` /rest/sensor/latitude/${shipId}/${containerId}/latest`, {
  headers: {
    'authorization': `Bearer ${accessToken}`
  }
});
const latitudeValue = latitudeResponse.data.latitude[0].value;
setLatitude(latitudeValue);

// Longitude
const longitudeResponse = await axiosInstance.get(` /rest/sensor/longitude/${shipId}/${containerId}/latest`, {
  headers: {
    'authorization': `Bearer ${accessToken}`
  }
});
const longitudeValue = longitudeResponse.data.longitude[0].value;
setLongitude(longitudeValue);

updateTableData();

} catch (error) {
  console.error('Error fetching data:', error);
}
};

fetchContainerData();
}, [containerId, shipId])

```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

4/8

```

    
```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

5/8

```

<thead>
  <tr>
    <th>Environment Data</th>
    <th>Value</th>
    <th>Level</th>
    <th>Problem</th>
  </tr>
</thead>
<tbody>
  {tableData.map((row, index) => (
    <tr key={index}>
      <td>(row.environment)</td>
      <td>(row.value)</td>
      <td>(row.level)</td>
      <td>(row.problem)</td>
    </tr>
  )));
</tbody>
</table>
</div>
<div className="h-5 w-1/4 pt-10 pr-10">
  <textarea className="resize-none overflow-auto text-base w-full border-2 border-dashed border-gray-400 bg-gray-100 text-gray-700 h-[385px] p-2"
    name="notes"
    id="notes"
    placeholder="Notes..."
    value={notes || ""}
    onChange={handleNotesChange}
    onBlur={handleBlur}
  ></textarea>
  {saved && (
    <div className="text-green-400 mt-1">Notes saved!</div>
  )}
  {notSaved && (
    <div className="text-red-400 mt-1">Failed to save notes!</div>
  )}
</div>
</div>
);
};

/*VALUES ('Air Pressure', 'Pa'),
('Humidity', '%'),
('Temperature', '°C'),
('Vibration', 'm/s²'),
('Altitude', 'm'),
('Latitude', 'DD'),
('Longitude', 'DD');*/
export default Detailspace;

```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

6/8

The issue you're facing with table updates likely stems from how the `tableData` is managed. The `tableData` array is not being updated correctly, as you're directly mutating it inside the `updateTableData` function without triggering a re-render.

In React, you should avoid directly mutating state, because it doesn't trigger a re-render. Instead, you should create a new copy of the array with the updated values and then use `setState` to trigger a re-render.

Here's a suggestion on how to fix it:

- Store the table data in state:** Instead of using a regular array for `tableData`, store it in a state variable so you can trigger a re-render when it updates.
- Update the table data properly:** When you want to update the table data, use the state setter to update the table data.

Here's how you can modify your code:

```

js

const [tableData, setTableData] = useState([
  { environment: "Temperature", value: "0°C", level: "", problem: "" },
  { environment: "Pressure", value: "Pa", level: "Critical", problem: "High Pressure" },
  { environment: "Humidity", value: "%", level: "", problem: "" },
  { environment: "Vibration", value: "m/s²", level: "", problem: "" },
  { environment: "Altitude", value: "m", level: "", problem: "" },
  { environment: "Latitude", value: "DD", level: "", problem: "" },
  { environment: "Longitude", value: "DD", level: "", problem: "" }
]);

const updateTableData = (newValue, sensor) => {
  setTableData((prevData) => {
    prevData.map(item => {
      switch(sensor) {
        case "temperature":
          if (item.environment === "Temperature") {
            return { ...item, value: newValue };
          }
          break;
        case "air_pressure":
          if (item.environment === "Pressure") {
            return { ...item, value: newValue };
          }
          break;
        case "humidity":
          if (item.environment === "Humidity") {
            return { ...item, value: newValue };
          }
          break;
        case "vibration":
          if (item.environment === "Vibration") {
            return { ...item, value: newValue };
          }
          break;
        case "altitude":
          if (item.environment === "Altitude") {
            return { ...item, value: newValue };
          }
          break;
        case "latitude":
          if (item.environment === "Latitude") {
            return { ...item, value: newValue };
          }
          break;
      }
    });
  });
};

```

Printed using [ChatGPT to PDF](#) powered by PDFCrowd [HTML to PDF API](#).

7/8

```

    case "longitude":
      if (item.environment === "Longitude") {
        return { ...item, value: newValue };
      }
      break;
    default:
      return item;
  }
  return item; // Return the item unchanged if no conditions match
});
};

With this change:

```

- The `tableData` is now stored in a state variable, so updates will trigger re-renders automatically.
- `updateTableData` now uses `setTableData` to properly update the state.

Additionally, ensure that `updateTableData` is called properly in your `useEffect` after fetching data:

```

js

useEffect(() => {
  const fetchContainerData = async () => {
    try {
      const accessToken = localStorage.getItem("accessToken");

      // Temperature
      const temperatureResponse = await axiosInstance.get(`rest/sensor/temperature/${shipId}/${containerId}/latest`, {
        headers: {
          'authorization': `Bearer ${accessToken}`,
        },
      });
      const temperatureValue = temperatureResponse.data.temperature[0].value;
      const temperatureSensor = temperatureResponse.data.temperature[0].sensor;
      updateTableData(temperatureValue, temperatureSensor);

      // Repeat for other sensors...
    } catch (error) {
      console.error(`Error fetching data: ${error}`);
    }
  };
  fetchContainerData();
}, [containerId, shipId]);

```

This should ensure that the data updates properly in the table after fetching.

 Ab hier sind Nachrichten nur für dich sichtbar