ManageEDatenLogger

(Dokumentation)

Index

Zusammenfassung

Das ist ein programm für ManageE gmbh, dass über manage und beobachten die Enegie verantwortet, zb. LiveDaten beobachten , Altdaten Lesen ,und der graph kontrollieren. Das pragramm wurde mit Node js und React aufgebaut.

Warum warum diese technologie ?

Der mann kann alle code trennen , dewegen einfach auch zubearbeiten .

installation

in dieser phase wir bruachen Node zu instulieren , und bibliothek Express und cors , mysql .

und in frontend wir brauchen , react , antd , ag-charts-react , ant-design/icons , victory , ag-charts-enterprise

Backend

```
var mysql = require('mysql');

var con = mysql.createConnection({
    host: '172.16.1.189',
    port: 3306,
    user: 'remote_user',
    password: 'IbgaNt-66',
    database: 'managee'
});

con.connect(function(err) {
  if (err) {
    console.error("Connection error:", err.message);
    return;
  }
  console.log("Connected!");
});

module.exports = con;
```

Deutsche Erklärung:

Dieser Code stellt eine Verbindung zu einer MySQL-Datenbank her. Hier ist eine Schritt-für-Schritt-Erklärung:

1. **MySQL-Modul importieren:**
    - var mysql = require('mysql');
    - Diese Zeile importiert das MySQL-Modul, um mit MySQL-Datenbanken zu interagieren.
2. **Verbindung erstellen:**
    - var con = mysql.createConnection({...});
    - Hier wird ein neues MySQL-Verbindungsobjekt mit folgenden Konfigurationen erstellt:
        - host: '172.16.1.189' - Die IP-Adresse des MySQL-Servers.
        - port: 3306 - Der Port, auf dem der MySQL-Server lauscht.
        - user: 'remote_user' - Der Benutzername für die Datenbankverbindung.
        - password: 'IbgaNt-66' - Das Passwort für die Datenbankverbindung.
        - database: 'managee' - Der Name der Datenbank, mit der verbunden werden soll.
3. **Verbindung zur Datenbank herstellen:**
    - con.connect(function(err) { ... });
    - Diese Zeile versucht, eine Verbindung zur MySQL-Datenbank mit der angegebenen Konfiguration herzustellen.
    - Bei einem Fehler während der Verbindung enthält der err-Parameter ein Fehlerobjekt.
    - Bei erfolgreicher Verbindung ist der err-Parameter null.
4. **Fehlerbehandlung:**
    - if (err) { ... }
    - Dieser Block prüft, ob während der Verbindung ein Fehler aufgetreten ist.
    - Bei einem Fehler wird die Fehlermeldung in der Konsole protokolliert und die Funktion beendet.

5. **Erfolgsmeldung:**
    - console.log("Connected!");
    - Bei erfolgreicher Verbindung wird die Meldung "Connected!" in der Konsole ausgegeben.
6. **Verbindung exportieren:**
    - module.exports = con;
    - Diese Zeile exportiert das con-Objekt, sodass es in anderen Teilen der Anwendung verwendet werden kann.

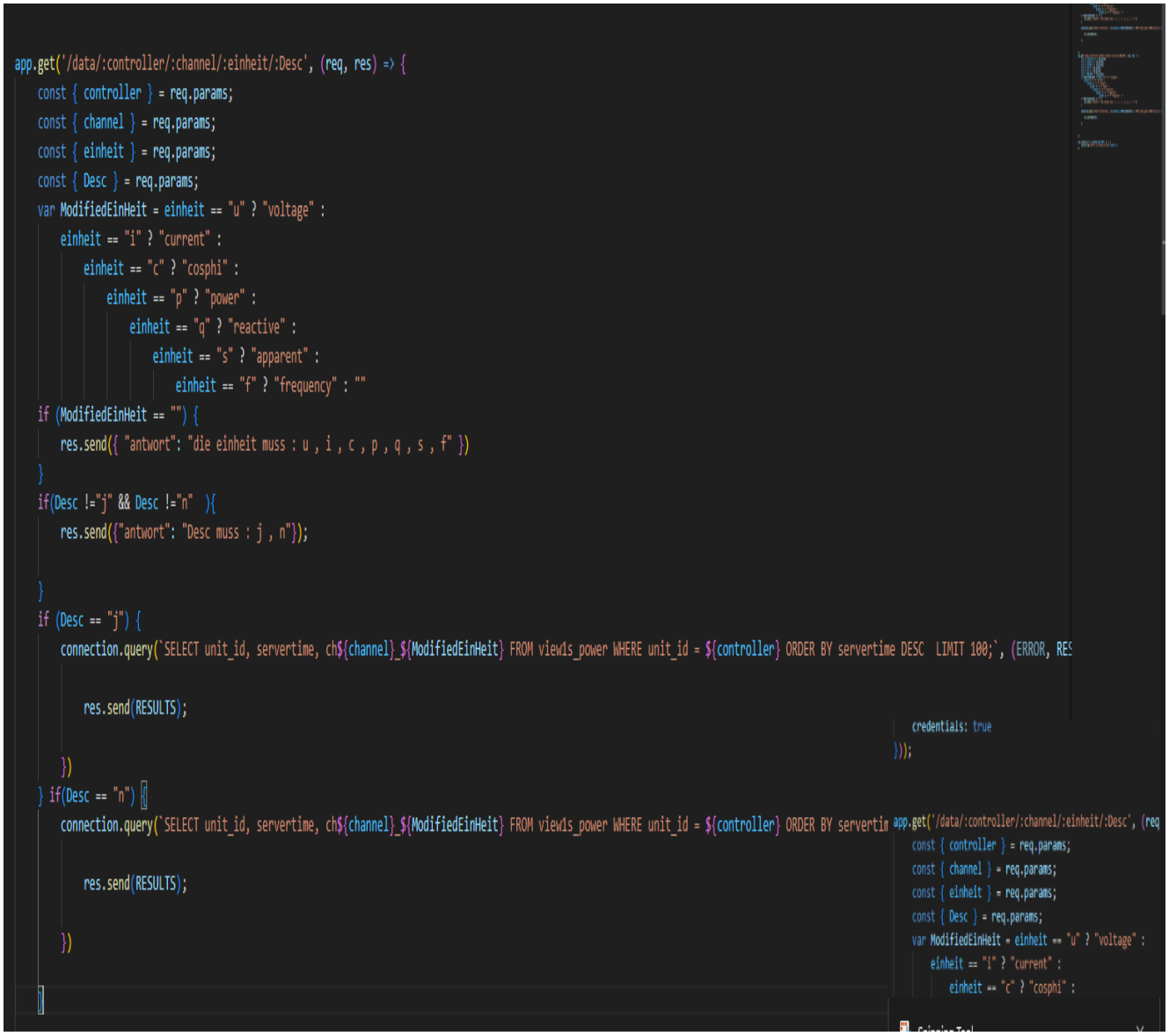
Gesamtzweck:

Dieser Code stellt eine Verbindung zu einer MySQL-Datenbank mit den angegebenen Anmeldeinformationen und Konfiguration her. Das Verbindungsobjekt wird dann exportiert, um Abfragen auszuführen und andere Datenbankoperationen innerhalb der Anwendung durchzuführen.

```
const app = express()
const port = 3000
app.use(cors());

app.use(cors({
    origin: '*',
    methods: ['GET', 'POST', 'PUT', 'DELETE'],
    allowedHeaders: ['Content-Type', 'Authorization'],
    credentials: true
}));
```

Die erste abteilung ist , Cors problem lösung , (das ist nich gut für sicherheit)

Und in diesem code ich habe gesagt dass , wenn sie ein anfrage kommt , antwort ohne anlehnung . Und 3000 ist der port , und express ist die bibliothek .



```
app.get('/data/:controller/:channel/:einheit/:Desc', (req, res) => {
    const { controller } = req.params;
    const { channel } = req.params;
    const { einheit } = req.params;
    const { Desc } = req.params;
    var ModifiedEinHeit = einheit == "u" ? "voltage" :
        einheit == "i" ? "current" :
            einheit == "c" ? "cosphi" :
                einheit == "p" ? "power" :
                    einheit == "q" ? "reactive" :
                        einheit == "s" ? "apparent" :
                            einheit == "f" ? "frequency" : ""
    if (ModifiedEinHeit == "") {
        res.send({ "antwort": "die einheit muss : u , i , c , p , q , s , f" })
    }
    if(Desc !="j" && Desc !="n"  ){
        res.send({"antwort": "Desc muss : j , n"});

    }
    if (Desc == "j") {
        connection.query(`SELECT unit_id, servertime, ch${channel}_${ModifiedEinHeit} FROM view1s_power WHERE unit_id = ${controller} ORDER BY servertime DESC  LIMIT 100;`, (ERROR, RES

            res.send(RESULTS);

        })
    } if(Desc == "n") {
        connection.query(`SELECT unit_id, servertime, ch${channel}_${ModifiedEinHeit} FROM view1s_power WHERE unit_id = ${controller} ORDER BY servertim

            res.send(RESULTS);

        })

    }
```

Endpoint: /data/:controller/:channel/:einheit/:Desc

Method: GET

Parameter:

Parameter	Beschreibung	Gültige Werte
controller	ID der Einheit	Integer
channel	Kanal innerhalb der Einheit	Integer
einheit	Maßeinheit	u (Spannung), i (Strom), c (Leistungsfaktor), p (Leistung), q (Blindleistung), s (Scheinleistung), f (Frequenz)
Desc	Sortierrichtung	j (absteigend), n (aufsteigend)

Antwort:

- **Erfolg:** JSON-Array mit den abgefragten Daten. Jedes Objekt enthält folgende Felder:
    - unit_id: ID der Einheit
    - servertime: Zeitstempel des Datenpunkts
    - ch${channel}_${ModifiedEinHeit}: Wert des angegebenen Kanals und der Maßeinheit

```
app.get('/analysis/:controller/:channel/:einheit/:von/:bis/:ROW_NUM', (req, res) => {
    const { controller } = req.params;
    const { channel } = req.params;
    const { einheit } = req.params;
    const { von } = req.params;
    const { bis } = req.params;
    const { ROW_NUM } = req.params;
    var ModifiedEinHeit = einheit == "u" ? "voltage" :
        einheit == "i" ? "current" :
            einheit == "c" ? "cosphi" :
                einheit == "p" ? "power" :
                    einheit == "q" ? "reactive" :
                        einheit == "s" ? "apparent" :
                            einheit == "f" ? "frequency" : ""
    if (ModifiedEinHeit == "") {
        res.send({ "antwort": "die einheit muss : u , i , c , p , q , s , f" })
    }

    connection.query(`SELECT servertime x, ch${channel}_${ModifiedEinHeit} y FROM view1s_power WHERE unit_id = ${controller} AND servertime BETWEEN "${von}" AND "${bis}"  ORDER BY serv

        res.send(RESULTS);

    })
```

Endpoint: /analysis/:controller/:channel/:einheit/:von/:bis/:ROW_NUM

Method: GET

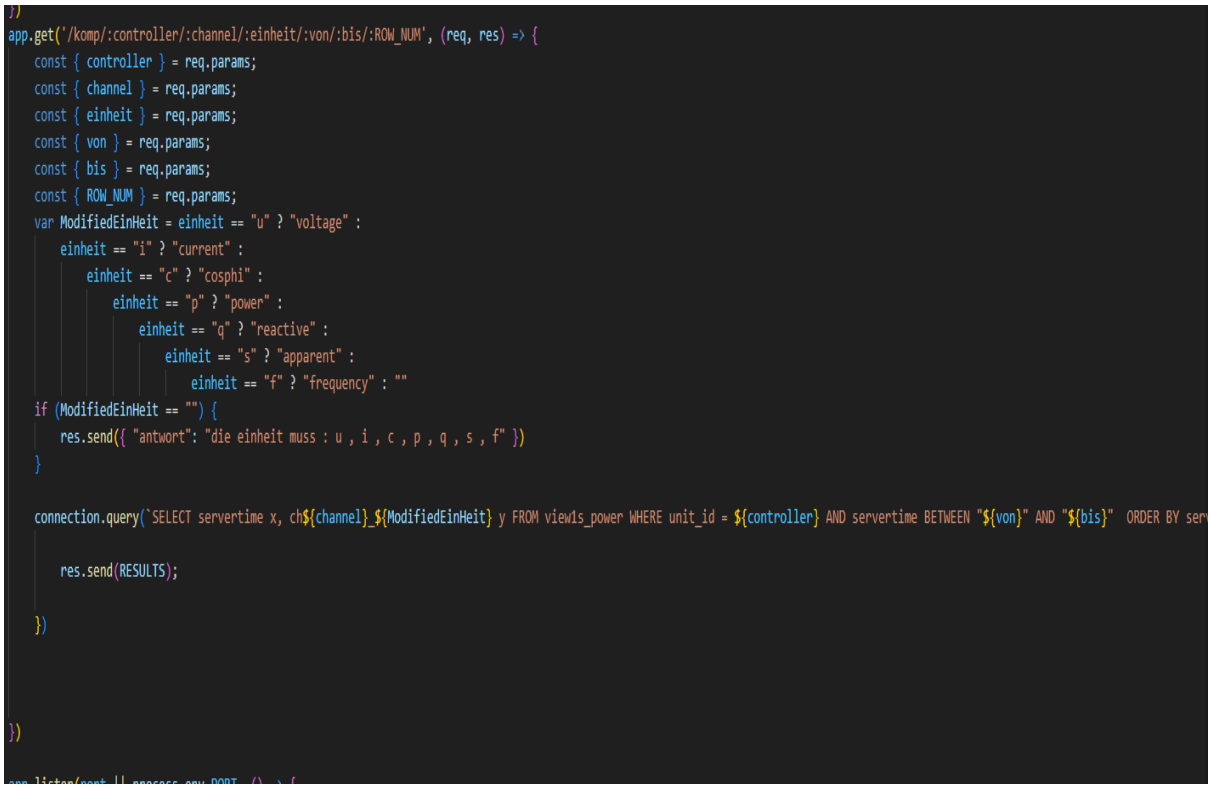
Parameter:

Parameter	Beschreibung	Gültige Werte
controller	ID der Einheit	Integer
channel	Kanal innerhalb der Einheit	Integer
einheit	Maßeinheit	u (Spannung), i (Strom), c (Leistungsfaktor), p (Leistung), q (Blindleistung), s (Scheinleistung), f (Frequenz)
von	Startzeitpunkt (ISO-8601)	String (z.B. "2023-11-22T10:00:00Z")
bis	Endzeitpunkt (ISO-8601)	String (z.B. "2023-11-23T10:00:00Z")

ROW_NUM	Anzahl der gewünschten Datensätze	Integer

Antwort:

- **Erfolg:** JSON-Array mit den abgefragten Daten. Jedes Objekt enthält folgende Felder:
    - x: Zeitstempel des Datenpunkts
    - y: Wert des angegebenen Kanals und der Maßeinheit



```
})
app.get('/komp/:controller/:channel/:einheit/:von/:bis/:ROW_NUM', (req, res) => {
    const { controller } = req.params;
    const { channel } = req.params;
    const { einheit } = req.params;
    const { von } = req.params;
    const { bis } = req.params;
    const { ROW_NUM } = req.params;
    var ModifiedEinHeit = einheit == "u" ? "voltage" :
        einheit == "i" ? "current" :
            einheit == "c" ? "cosphi" :
                einheit == "p" ? "power" :
                    einheit == "q" ? "reactive" :
                        einheit == "s" ? "apparent" :
                            einheit == "f" ? "frequency" : ""
    if (ModifiedEinHeit == "") {
        res.send({ "antwort": "die einheit muss : u , i , c , p , q , s , f" })
    }

    connection.query(`SELECT servertime x, ch${channel}_${ModifiedEinHeit} y FROM view1s_power WHERE unit_id = ${controller} AND servertime BETWEEN "${von}" AND "${bis}"  ORDER BY serv

        res.send(RESULTS);

    })



})
```

Endpoint: /komp/:controller/:channel/:einheit/:von/:bis/:ROW_NUM

Method: GET

Parameter:

Parameter	Beschreibung	Gültige Werte
controller	ID der Einheit	Integer

channel	Kanal innerhalb der Einheit	Integer
einheit	Maßeinheit	u (Spannung), i (Strom), c (Leistungsfaktor), p (Leistung), q (Blindleistung), s (Scheinleistung), f (Frequenz)
von	Startzeitpunkt (ISO-8601)	String (z.B. "2023-11-22T10:00:00Z")
bis	Endzeitpunkt (ISO-8601)	String (z.B. "2023-11-23T10:00:00Z")
ROW_NUM	Anzahl der gewünschten Datensätze	Integer

Antwort:

- **Erfolg:** JSON-Array mit den abgefragten Daten. Jedes Objekt enthält folgende Felder:
    - x: Zeitstempel des Datenpunkts
    - y: Wert des angegebenen Kanals und der Maßeinheit

Frontend

Projektstruktur: managee-dashboard

Projektübersicht:

Die vorliegende Projektstruktur ist für eine React-basierte Webanwendung namens "managee-dashboard" konzipiert. Sie ist in mehrere Ordner und Dateien unterteilt, um eine klare Organisation und Übersichtlichkeit zu gewährleisten.

Hauptordner:

- **node_modules:** Dieser Ordner enthält alle externen Abhängigkeiten (Packages), die für das Projekt benötigt werden. Diese werden über npm (Node Package Manager) installiert.
- **public:** Dieser Ordner enthält statische Assets wie Bilder, CSS-Dateien und andere Ressourcen, die direkt vom Browser abgerufen werden können.
- **BusinessLogic:** Hier befinden sich die Kernfunktionalitäten und Geschäftslogik der Anwendung, einschließlich Funktionen, Klassen oder Module für Datenverarbeitung, Berechnungen und andere wichtige Aufgaben.
- **components:** Dieser Ordner enthält wiederverwendbare Komponenten, die die Benutzeroberfläche (UI) der Anwendung bilden. Komponenten sind selbstständige Bausteine, die kombiniert werden können, um komplexe UIs zu erstellen.
- **dashboardMenu:** Dieser Ordner könnte Komponenten enthalten, die mit der Navigation und Menüstruktur der Anwendung zusammenhängen.
- **img:** Dieser Ordner speichert Bilddateien, die in der Anwendung verwendet werden.
- **App.css:** Diese Datei enthält globale CSS-Styles, die auf die gesamte Anwendung angewendet werden.
- **App.js:** Diese Datei ist wahrscheinlich der Haupteinstiegspunkt der Anwendung, wo die React-App initialisiert und gerendert wird.
- **App.test.js:** Diese Datei enthält Unit-Tests für die App.js-Komponente.
- **index.css:** Diese Datei enthält CSS-Styles für die Haupt-HTML-Seite der Anwendung.
- **index.js:** Diese Datei ist der Einstiegspunkt für die Webanwendung und ist oft dafür verantwortlich, die React-App in das DOM zu rendern.
- **logo.svg:** Diese Datei enthält das SVG-Logo der Anwendung.

- **reportWebVitals.js:** Diese Datei kümmert sich wahrscheinlich um das Reporting von Web-Performance-Metriken.
- **setupTests.js:** Diese Datei richtet die Testumgebung für die Anwendung ein.
- **.gitignore:** Diese Datei spezifiziert Dateien und Verzeichnisse, die von Git ignoriert werden sollen, um zu verhindern, dass sie verfolgt und in das Repository übertragen werden.
- **package-lock.json:** Diese Datei sperrt die Versionen der Abhängigkeiten, um konsistente Builds in verschiedenen Umgebungen zu gewährleisten.
- **package.json:** Diese Datei enthält Metadaten über das Projekt, einschließlich Abhängigkeiten, Skripten und Konfigurationsoptionen.
- **README.md:** Diese Datei bietet einen allgemeinen Überblick über das Projekt, Anweisungen zur Einrichtung und andere relevante Informationen.
- **tailwind.config.js:** Diese Datei konfiguriert Tailwind CSS, ein Utility-First-CSS-Framework, für das Projekt.

Allgemeine Struktur:

Die Verzeichnisstruktur ist so organisiert, dass sie Modularität, Wartbarkeit und Skalierbarkeit fördert. Komponenten, Geschäftslogik und statische Assets sind in verschiedenen Ordnern getrennt. Dies erleichtert die Verwaltung und das Verständnis des Codebases, insbesondere bei wachsenden Projekten.

Zusätzliche Anmerkungen:

- Der genaue Inhalt jeder Datei und jedes Ordners hängt von den Funktionen und der Komplexität der Anwendung ab.
- Die Verwendung von Tailwind CSS deutet darauf hin, dass das Projekt einen Utility-First-CSS-Ansatz verwendet, der den Entwicklungsprozess vereinfachen kann.
- Die Verwendung von Unit-Tests zeigt ein Augenmerk auf Codequalität und Wartbarkeit.

```
JS Api.js
managee-dashboard > src > BusinessLogic > JS Api.js > analysisKompApi
1   export const GeneralApi ="http://localhost:3000/data/"
2   export const analysisApi ="http://localhost:3000/analysis/"
3   export const analysisKompApi ="http://localhost:3000/komp/"
```

GeneralApi: Dieser Endpunkt, der auf http://localhost:3000/data/ zeigt, könnte für allgemeine Datenabrufe oder andere allgemeine Operationen verwendet werden.

analysisApi: Dieser Endpunkt, der auf http://localhost:3000/analysis/ zeigt, könnte für spezifische Analyse- oder Datenverarbeitungsaufgaben verwendet werden.

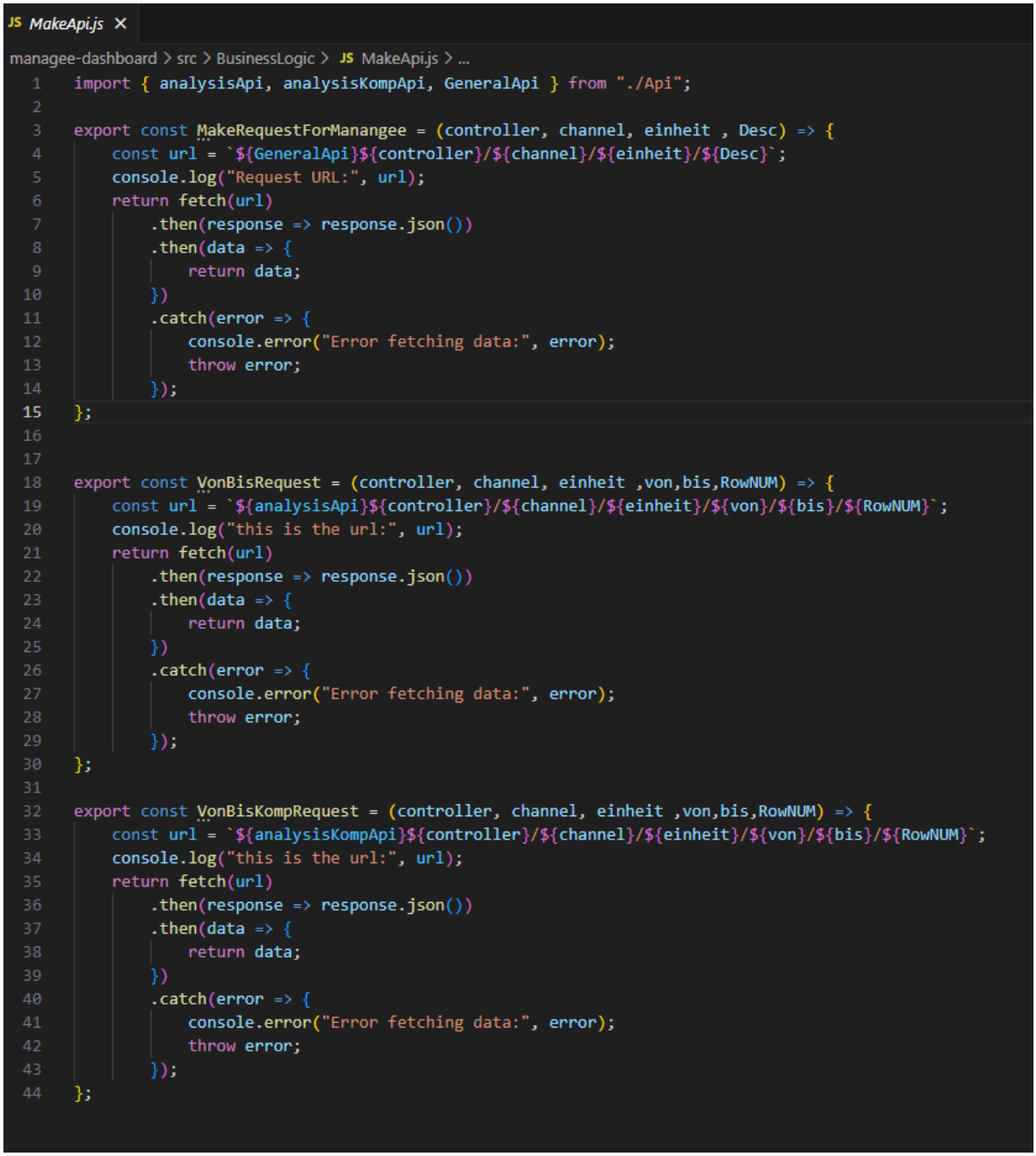
analysisKompApi: Dieser Endpunkt, der auf http://localhost:3000/komp/ zeigt, könnte für spezifische Analyse- oder Datenverarbeitungsaufgaben im Zusammenhang mit einer Komponente oder einem Modul namens "Komp" verwendet werden

```
JS ChangeDateFormat.js
-dashboard > src > BusinessLogic > JS ChangeDateFormat.js > formatToIS
export function formatToISO8601(dateString) {
    const date = new Date(dateString);

    // Check if the date is valid
    if (isNaN(date.getTime())) {
      throw new Error("Invalid date format");
    }

    // Add 12 hours (12 * 60 * 60 * 1000 milliseconds)
    date.setHours(date.getHours() + 12);

    return date.toISOString();
  }
```

Die ChangeDateFormat.js-Datei enthält eine Funktion namens formatToISO8601, die dazu dient, ein gegebenes Datumsformat in das ISO-8601-Format zu konvertieren. Das ISO-8601-Format ist ein internationaler Standard für die Darstellung von Datums- und Zeitangaben.

Funktionsweise:

1. **Eingabe:** Die Funktion nimmt eine Zeichenkette dateString als Eingabe, die ein Datum in einem beliebigen unterstützten Format darstellt.
2. **Datumsobjekt erstellen:** Ein neues Date-Objekt wird erstellt, wobei der Eingabe-String als Parameter verwendet wird.
3. **Validierung:** Die Funktion überprüft, ob das erstellte Datum gültig ist, indem sie prüft, ob isNaN(date.getTime()) true zurückgibt. Wenn das Datum ungültig ist, wird ein Error geworfen.
4. **Zeitzonenanpassung:** Die Funktion fügt 12 Stunden zum Datum hinzu, möglicherweise um eine Zeitzonenanpassung vorzunehmen.
5. **ISO-8601-Formatierung:** Das Datum wird in das ISO-8601-Format konvertiert und als Zeichenkette zurückgegeben



Die MakeApi.js-Datei enthält drei Funktionen, die HTTP-Anfragen an verschiedene API-Endpunkte senden und die Antworten verarbeiten. Diese Funktionen werden wahrscheinlich verwendet, um Daten von einem Backend-Server abzurufen.

Funktionen:

1. **MakeRequestForManangee:**
    - Konstruiert eine URL basierend auf den übergebenen Parametern controller, channel, einheit und Desc.
    - Sendet eine GET-Anfrage an die konstruierte URL.
    - Parst die JSON-Antwort und gibt sie zurück.

2. **VonBisRequest:**
    - Konstruiert eine URL basierend auf den übergebenen Parametern controller, channel, einheit, von, bis und RowNUM.
    - Sendet eine GET-Anfrage an die konstruierte URL.
    - Parst die JSON-Antwort und gibt sie zurück.
3. **VonBisKompRequest:**
    - Konstruiert eine URL basierend auf den übergebenen Parametern controller, channel, einheit, von, bis und RowNUM.
    - Sendet eine GET-Anfrage an die konstruierte URL.
    - Parst die JSON-Antwort und gibt sie zurück.

Datei ChannelundEinheitAltData.js

(Übersicht für alte Daten)

Die Komponente ChannelundEinheitAltData scheint dazu gedacht zu sein, Daten von einem Backend-Server abzurufen, basierend auf verschiedenen Parametern wie Controller-ID, Kanal-ID und Einheit. Diese Daten werden dann in einem Liniendiagramm visualisiert.

Funktionsweise:

1. **Zustandshandhabung:**
    - Die Komponente nutzt den useState-Hook, um den Zustand der verschiedenen Parameter wie controllerState, channelState, einheitState, Daten und VoBis zu verwalten.
2. **Dropdown-Menüs:**
    - Die Komponente verwendet Dropdown-Komponenten von Ant Design, um dem Benutzer die Auswahl von Controller, Kanal und Einheit zu ermöglichen.
    - Die Optionen für die Einheit werden aus der EinHeitSealedClass bezogen.
3. **Datenabruf:**
    - Die handleFetchData-Funktion wird aufgerufen, wenn der Benutzer auf die "Suchen"-Schaltfläche klickt.

- Diese Funktion verwendet die MakeRequestForManangee-Funktion aus der BusinessLogic-Schicht, um eine API-Anfrage an den Backend-Server zu senden.
- Die empfangenen Daten werden im Zustand gespeichert und das Diagramm aktualisiert.

4. **Diagrammdarstellung:**
    - Die AgCharts-Komponente von ag-Charts-React wird verwendet, um das Liniendiagramm zu rendern.
    - Die Daten werden dynamisch an die options-Eigenschaft der AgCharts-Komponente übergeben.
    - Die X-Achse zeigt die Zeitachse, und die Y-Achse zeigt den Wert der ausgewählten Einheit.

Datei ChannelundEinheitDateSuchen

Die Komponente ChannelundEinheitDateSuchen baut auf der vorherigen Komponente ChannelundEinheitAltData auf und erweitert sie um die Möglichkeit, Daten nach einem Datumsbereich zu filtern. Sie ermöglicht es dem Benutzer, historische Daten für einen bestimmten Controller, Kanal, Einheit, Zeitraum und Row-Anzahl abzurufen und in einem Liniendiagramm zu visualisieren.

Funktionsweise:

1. **Zustandshandhabung:**
    - Die Komponente nutzt useState für die Verwaltung verschiedener Zustände wie Controller-ID (controllerState), Kanal-ID (channelState), Einheit (einheitState), Datumsbereich (von_, bis_), Row-Anzahl (RowNUM) und die empfangenen Daten (Daten).
2. **Dropdown-Menüs:**
    - Dropdown-Menüs von Ant Design werden verwendet, um die Auswahl von Controller, Kanal und Einheit zu ermöglichen.
3. **Datumsauswahl:**
    - Die Komponente DatePicker von Ant Design wird verwendet, um den Datumsbereich für die Abfrage festzulegen.
4. **Datenabruf:**

- Die handleFetchData-Funktion wird aufgerufen, wenn der Benutzer auf die "Suchen"-Schaltfläche klickt.
- Sie verwendet die VonBisRequest-Funktion aus der BusinessLogic-Schicht, um eine API-Anfrage an den Backend-Server zu senden.
- Die ausgewählten Parameter (controllerState, channelState, einheitState, von_ (formattiert mit formatToISO8601), bis_ (formattiert mit formatToISO8601), RowNUM) werden an die API übergeben.
- Die empfangenen Daten werden im Zustand (Daten) gespeichert und das Diagramm aktualisiert.

5. **Diagrammdarstellung:**
   - Die VictoryChart-Komponente von Victory Charts wird verwendet, um das Liniendiagramm zu rendern.
   - Die Daten werden dynamisch an die data-Eigenschaft der VictoryLine-Komponente übergeben.
   - Die X-Achse zeigt den Datumsbereich, und die Y-Achse zeigt den Wert der ausgewählten Einheit.

Datei ChannelundEinheitLiveDaten

Die Komponente ChannelundEinheitLiveDaten dient zur Anzeige von Live-Daten aus einem Backend-System in einem Radial-Gauge-Diagramm. Sie ermöglicht die Auswahl von Controller, Kanal und Einheit, um die entsprechenden Live-Daten anzuzeigen.

Funktionsweise:

1. **Zustandshandhabung:**
   - Die Komponente nutzt den useState-Hook, um den Zustand der verschiedenen Parameter wie controllerState, channelState, einheitState und die Diagrammkonfiguration (options) zu verwalten.
2. **Dropdown-Menüs:**
   - Dropdown-Menüs von Ant Design werden verwendet, um die Auswahl von Controller, Kanal und Einheit zu ermöglichen.
3. **Datenabruf und Aktualisierung:**
   - Die handleFetchData-Funktion wird aufgerufen, um die Daten vom Backend-Server abzurufen.

- Die empfangenen Daten werden verwendet, um die options des Diagramms zu aktualisieren.
- Der useEffect-Hook wird verwendet, um die handleFetchData-Funktion in einem bestimmten Intervall (3 Sekunden) wiederholt aufzurufen, um die Daten live zu aktualisieren.

4. **Diagrammdarstellung:**
   - Die AgCharts-Komponente von ag-Charts-React wird verwendet, um das Radial-Gauge-Diagramm zu rendern.
   - Die Diagrammkonfiguration wird über die options-Eigenschaft an die Komponente übergeben.

Datei DashboardMainPage

Die Komponente DashboardMainPage dient als Hauptkomponente des Dashboards und organisiert die Anordnung der anderen Komponenten auf der Seite. Sie ist verantwortlich für die gesamte Layoutstruktur und die Platzierung der einzelnen Komponenten.

Funktionsweise:

1. **Komponentenimport:**
   - Die Komponente importiert die Komponenten ChannelundEinheitAltData, ChannelundEinheitLiveDaten, ChannelundEinheitDateSuchen und KomponiertGraph.
2. **Layout:**
   - Die Hauptkomponente verwendet Flexbox für das Layout.
   - Die ersten beiden Komponenten (ChannelundEinheitAltData und ChannelundEinheitLiveDaten) werden nebeneinander in einer Reihe angeordnet, wenn der Bildschirm breit genug ist (ab md-Breakpoint).
   - Die dritte Komponente (ChannelundEinheitDateSuchen) wird in einer eigenen Reihe angezeigt.
   - Die vierte Komponente (KomponiertGraph) wird in einer eigenen Reihe angezeigt.

Datei KomponiertGraph.js

Die Komponente KomponiertGraph dient zur Darstellung eines Liniendiagramms mit mehreren Linien. Die Daten für das Diagramm werden über eine API abgefragt und basieren auf den ausgewählten Parametern wie Controller, Kanal, Einheit, Datumsbereich und Anzahl der anzuzeigenden Datenpunkte.

Funktionsweise:

1. **Zustandshandhabung:**
   - Die Komponente nutzt den useState-Hook, um den Zustand verschiedener Parameter zu verwalten:

        - controllerState: Der ausgewählte Controller
        - channelState: Der ausgewählte Kanal
        - einheitState: Die ausgewählte Einheit
        - von_: Startdatum für die Datenabfrage
        - bis_: Enddatum für die Datenabfrage
        - RowNUM: Anzahl der anzuzeigenden Datenpunkte
        - Daten: Array, das die abgerufenen Daten speichert
        - GraphColors: Array, das die Farben für die einzelnen Linien speichert
2. **Dropdown-Menüs:**
    - Verschiedene Dropdown-Menüs von Ant Design werden verwendet, um die Auswahl von Controller, Kanal und Einheit zu ermöglichen.
3. **Datenabruf:**
    - Die handleFetchData-Funktion wird aufgerufen, um die Daten vom Backend-Server abzurufen. Die Funktion nimmt die ausgewählten Parameter und das Datumsintervall entgegen.
    - Bei erfolgreichem Abruf werden die Daten dem Daten-State hinzugefügt und für jede Datenreihe eine zufällige Farbe dem GraphColors-State hinzugefügt.
    - Fehlerbehandlung ist mit try...catch implementiert.
4. **Diagrammdarstellung:**
    - Die VictoryChart-Komponente von Victory Charts wird verwendet, um das Liniendiagramm zu rendern.
    - Die Komponente wird mit Daten aus dem Daten-State und Farben aus dem GraphColors-State gefüttert.
    - Verschiedene Konfigurationen für das Diagramm wie Zoomfunktion und Hintergrundfarbe werden festgelegt.
    - Die Datenreihen werden mit VictoryLine innerhalb einer map-Funktion für Daten gerendert.
    - Responsive Verhalten: Es wird zwischen zwei verschiedenen Layouts für Desktop und Mobile Geräte unterschieden.

Datei DashboardItems.js

Die Komponente DashboardItems dient zur Darstellung einer Reihe von Items, die wahrscheinlich verschiedene Funktionen oder Abschnitte eines Dashboards repräsentieren. Sie verwendet eine Kombination aus Text und Icons, um diese Items visuell darzustellen.

Funktionsweise:

1. **Import von Komponenten und Icons:**
   - Die notwendigen Komponenten und Icons werden importiert.
2. **Komponentenstruktur:**
   - Die Komponente verwendet ein div als Hauptcontainer mit einem flexiblen Layout, das mit CSS Grid und Flexbox gestaltet wird.
   - Das Layout passt sich an die Bildschirmgröße an:
     - Auf kleineren Bildschirmen (Mobilgeräte) werden die Items in einer einzigen Spalte angezeigt.
     - Auf größeren Bildschirmen (Desktop) werden die Items in mehreren Spalten angezeigt.
3. **Logo:**
   - Das Logo LogoManageE wird oben in der Komponente angezeigt.
4. **Item-Darstellung:**
   - Die Item-Komponente wird verwendet, um einzelne Items darzustellen, wobei jedes Item aus einem Icon und einem Textlabel besteht.

Datei Item.js

Die Komponente Item ist ein wiederverwendbares UI-Element, das zur Darstellung einzelner Items in einem Dashboard verwendet wird. Jedes Item besteht aus einem Icon und einem zugehörigen Textlabel.

Funktionsweise:

1. **Props:**
   - Die Komponente nimmt zwei Props entgegen:
     - name: Der Text, der neben dem Icon angezeigt wird.
     - DrawComponent: Eine Funktion, die das gewünschte Icon-Element rendert.
2. **Layout:**
   - Die Komponente verwendet Flexbox für das Layout, um das Icon und den Text nebeneinander anzuordnen.
   - Das Icon wird in einem quadratischen Container mit einem Hintergrund platziert.

- Der Text wird in einem p-Tag angezeigt, das auf größeren Bildschirmen sichtbar ist.

3. **Responsivität:**
    - Die Komponente passt sich an verschiedene Bildschirmgrößen an, indem sie das Layout und die Anzeige des Textlabels anpasst.

Datei dashboard.js

Die Komponente dashboard dient als Hauptlayout für die Dashboard-Anwendung. Sie organisiert den Hauptinhaltsbereich (DashboardMainPage) und eine Seitenleiste (DashboardItems) in einem grid-basierten Layout.

Funktionsweise:

1. **Import von Komponenten:**
    - Die notwendigen Komponenten DashboardItems und DashboardMainPage werden importiert.
2. **Layout:**
    - Die Komponente verwendet ein div mit einem Grid-Layout, um die DashboardItems und DashboardMainPage Komponenten anzuordnen.
    - Die CSS-Eigenschaften grid-rows und grid-cols werden verwendet, um das Zeilen- und Spaltenlayout zu definieren.
    - Ein Media-Query wird verwendet, um das Layout für Bildschirme größer als 883px anzupassen und ein zweispaltiges Layout zu erstellen.

Datei **IntroPage**.js

Die Komponente IntroPage dient als einfache Startseite, die ein Logo anzeigt und den Benutzer nach einer bestimmten Verzögerung zur Dashboard-Seite weiterleitet.

Funktionsweise:

1. **Import von Komponenten:**
    - Die notwendigen Komponenten useNavigate aus react-router-dom und LogoManageE für das Logo-Bild werden importiert.
2. **Navigationshandling:**
    - Der useEffect-Hook wird verwendet, um einen Timer einzurichten, der nach 4 Sekunden die Navigation zur /Dashboard-Route auslöst.
3. **Komponenten-Rendering:**

- Die Komponente rendert einen Vollbild-Container mit dem zentrierten Logo.
- Die Klasse animate-pulse wird auf das Logo angewendet, um eine Ladeanimation zu erzeugen.