

# **Herbsthofer - CESC**

## **DIPLOMARBEIT**

verfasst im Rahmen der

**Reife- und Diplomprüfung**

an der

**Höheren Abteilung für Informatik**

Eingereicht von:

Elias Mahr  
Leopold Mistelberger  
Timon Schmalzer

Betreuer:

Thomas Stütz

Projektpartner:

Herbsthofer

Leonding, 4. April 2026

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Leonding, 4. April 2026

Elias Mahr & Leopold Mistelberger

# Abstract [Timon Schmalzer]

CESC is an automated control system that enables company managers and site supervisors to reduce their company's energy costs and monitor overall energy consumption at construction sites.

Using the developed software for energy data management, energy data can be efficiently collected, stored, and analyzed.

The application allows users to access and evaluate the energy data of their construction sites from anywhere, as well as centrally control the available power sources.

The system is based on the interaction of hardware and software components and follows the Internet of Things (IoT) principle. Each power source and all analysis components can be addressed and controlled via the MQTT protocol.

The backend, developed in C#, provides a REST API that is used by the web application to display measurement data and transmit control commands. In addition, all data is stored in a Firebase database to enable further analysis and to temporarily store scheduled actions that are to be executed on site.

Furthermore, the system includes an integrated calendar that allows the automatic control of power sources to be scheduled in order to further optimize energy consumption. The calendar can also be used to manually deactivate power sources for specific periods of time, such as public holidays or weekends.



# Zusammenfassung [Timon Schmalzer]

CESC ist ein automatisches Kontrollsysteem, das Geschäftsführern und Baustellenleitern ermöglicht, die Energiekosten ihres Unternehmens zu senken und den allgemeinen Energieverbrauch auf Baustellen zu überwachen.

Mithilfe der entwickelten Software zur Verwaltung von Energiedaten können diese effizient gesammelt, gespeichert und analysiert werden.

Die Applikation erlaubt es BenutzerInnen, von überall aus auf die Energiedaten ihrer Baustellen zuzugreifen, diese auszuwerten sowie vorhandene Stromquellen zentral zu steuern.

Das System basiert auf einem Zusammenspiel von Hardware- und Softwarekomponenten und folgt dem IoT-(Internet of Things-)Prinzip. Jede Stromquelle sowie alle Analysekomponenten können über das MQTT-Protokoll angesprochen und gesteuert werden.

Das auf C# basierende Backend stellt eine REST-API zur Verfügung, die von der Webapplikation genutzt wird, um Messdaten darzustellen und Steuerbefehle zu übermitteln. Zusätzlich werden sämtliche Daten in einer Firebase-Datenbank gespeichert, um eine spätere Analyse zu ermöglichen sowie geplante Aktionen zwischenspeichern, die vor Ort ausgeführt werden sollen.

Des Weiteren bietet das System einen integrierten Kalender, mit dem die automatische Steuerung der Stromquellen zeitlich geplant werden kann, um den Energieverbrauch weiter zu optimieren. Der Kalender kann außerdem verwendet werden, um Stromquellen für bestimmte Zeiträume manuell zu deaktivieren, beispielsweise an Feiertagen oder Wochenenden.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Ausgangslage [Elias Mahr] . . . . .	1
1.2 Ist-Zustand [Elias Mahr] . . . . .	1
1.3 Problem [Leopold Mistelberger] . . . . .	1
1.4 Aufgabenstellung [Leopold Mistelberger] . . . . .	2
1.5 Zustandsdiagramm . . . . .	3
<b>2 Umfeldanalyse</b>	<b>4</b>
<b>3 Technologien</b>	<b>5</b>
3.1 Hardware und Betriebssystem [Leopold Mistelberger] . . . . .	5
3.2 Kommunikation und Automatisierung [Leopold Mistelberger] . . . . .	7
3.3 Container- und Laufzeitumgebung [Leopold Mistelberger] . . . . .	9
3.4 Server- und Infrastrukturtechnologien [Elias Mahr] . . . . .	9
3.5 Reverse-Proxy [Elias Mahr] . . . . .	9
3.6 Auswahl der Technologie - Sicherheit(Authentifizierung) . . . . .	11
3.7 Cronjob . . . . .	13
3.8 Frontend . . . . .	13
3.9 Api . . . . .	16
3.10 Backend . . . . .	16
<b>4 Umsetzung</b>	<b>22</b>
4.1 Systemarchitektur [Elias Mahr] . . . . .	22
4.2 Containerarchitektur [Elias Mahr] . . . . .	22
4.3 Keycloak [Elias Mahr] . . . . .	23
4.4 Reverse Proxy [Elias Mahr] . . . . .	26
4.5 Angular . . . . .	27
<b>5 Zusammenfassung</b>	<b>33</b>

<b>Glossar</b>	<b>V</b>
<b>Literaturverzeichnis</b>	<b>VI</b>
<b>Abbildungsverzeichnis</b>	<b>VII</b>
<b>Tabellenverzeichnis</b>	<b>VIII</b>
<b>Quellcodeverzeichnis</b>	<b>IX</b>
<b>Anhang</b>	<b>X</b>

# 1 Einleitung

## 1.1 Ausgangslage [Elias Mahr]

Das seit 1870 familiengeführte Unternehmen Herbsthofer plant, liefert, montiert und wartet Heizungs-, Kühlungs-, Lüftungs- und Sanitäranlagen (kurz HKLS-Anlagen) für Industrie, Gesundheitswesen und Forschung. Beispielhafte Projekte wären die dm Zentrale, das MSD Krems und die Techbase in Linz. Das Unternehmen nutzt 3D-Planung und moderne, digitale Tools zur Kosten- und Qualitätskontrolle. Nationale und internationale Projekte werden vom Hauptsitz Linz aus umgesetzt. Hier mehr dazu: <https://www.herbsthofer.at/leistungsspektrum>



Abbildung 1: Herbsthofer Logo

## 1.2 Ist-Zustand [Elias Mahr]

Das Unternehmen betreibt zahlreiche Baustellen im In- und Ausland. Für die Dauer der Projekte werden auf diesen Baustellen Containeranlagen genutzt, die als temporäres Büro, Aufenthalts- und Lagerräume dienen. Die Container sind je nach Nutzung mit W-Lan, Heizkörpern, Kühlsystem und Beleuchtung ausgestattet. Um einen durchgehenden Betrieb und geeignete Arbeitsbedingungen für das Personal zu gewährleisten wird durchgehend geheizt/gekühlt, wobei die Steuerung manuell vor Ort erfolgt.

## 1.3 Problem [Leopold Mistelberger]

Baustellencontainer werden häufig ohne technische Überwachung betrieben, wodurch hohe Energiekosten entstehen, da hauptsächlich die Heizung unabhängig von der tatsächlichen Nutzung betrieben wird. Eine Kontrolle über den Zustand des Containers ist nicht gegeben, weshalb die Firma keinen Zugriff auf aktuelle Informationen über Heizung, Raumklima, geöffnete Türen, Nutzung, Anwesenheit, Temperatur, Luftfeuchtigkeit und Lichtstatus hat.

Dieser Mangel erschwert dem Betrieb eine effiziente Steuerung der Baustellencontainer und führt zu erhöhten Betriebskosten. Darüber hinaus stellen Container ein gewisses Sicherheitsrisiko dar. Einbrüche bleiben oftmals unbemerkt, da weder eine automatische Erkennung noch eine sofortige Benachrichtigung erfolgt. In solchen Fällen ist eine zeitnahe Reaktion nicht möglich, und Vorfälle können nur im Nachhinein nachvollzogen werden, was die Suche nach dem Täter erheblich erschwert.

## 1.4 Aufgabenstellung [Leopold Mistelberger]

Ziel dieser Diplomarbeit ist die Entwicklung des Systems CESC zur Überwachung, Steuerung und erhöhten Sicherheit aller Baustellencontainer der Firma Herbstrofer. Das System soll auch dazu beitragen, sowohl die Betriebssicherheit als auch die Energieeffizienz auf Baustellen nachhaltig und automatisiert zu verbessern.

Im Rahmen der Arbeit soll ein verteiltes IoT-basiertes System entwickelt werden, das sowohl in Aufenthalts- als auch Lagercontainern implementiert wird. Innerhalb eines Containers werden verschiedene Umfeld- und Zustandsdaten erfasst, verarbeitet und visualisiert. Dazu zählen Innen-/Außentemperatur, Innen-/Außenfeuchtigkeit, Türzustand sowie die Erkennung einer Anwesenheit.

Ein weiterer wichtiger Teil des Systems ist die aktive Steuerung von Heizung, Klimatisierung und Licht. Ziel ist es, den Energieverbrauch durch automatisierte und zeitgesteuerte Steuerungsmechanismen zu minimieren, ohne dabei die Arbeiter zu behindern oder die Funktionsfähigkeit des Containers einzuschränken. Bei Sonderfällen soll auch ein manueller Eingriff über eine Benutzeroberfläche möglich sein.

«««< HEAD Ein besonderer Fokus dieser Arbeit liegt auf der Sicherheit. Das System überwacht den Türzustand der Container und speichert jede Türöffnung mit einem Zeitstempel. Dadurch kann nachvollzogen werden, wann und wie oft ein Container geöffnet wurde, insbesondere auch außerhalb der Arbeitszeiten. Dieses Protokoll trägt zur besseren Übersicht und Sicherheit auf der Baustelle bei. ===== Ein besonderer Fokus dieser Arbeit liegt auf der Sicherheit. In einem Container sollen zusätzlich Kameras verbaut werden, die eine Aufnahme starten, wenn ein unbefugtes Eintreten außerhalb der Arbeitszeit stattfindet. In solchen Fällen soll automatisch der Geschäftsführer per E-Mail oder SMS eine Benachrichtigung erhalten, um ein rasches Eingreifen zu ermöglichen.

## 1.5 Zustandsdiagramm

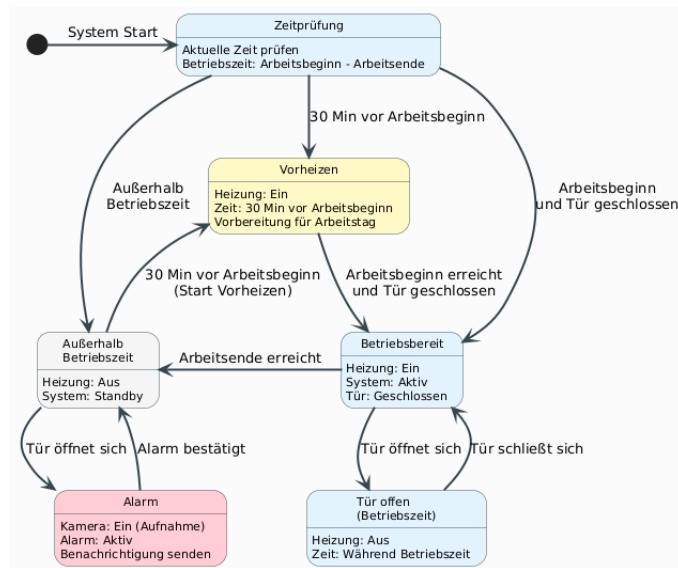


Abbildung 2: Zustandsdiagramm

»»»> 4626e689dbb1ed3ecc404c791b63c1c83b5ae5e5

## **2 Umfeldanalyse**

# **3 Technologien**

## **3.1 Hardware und Betriebssystem [Leopold Mistelberger]**

### **3.1.1 Raspberry Pi mit Ubuntu [Leopold Mistelberger]**

#### **3.1.2 Beschreibung und Rolle**

Der Raspberry Pi 5 ist ein kleiner, kompakter Computer, der als zentrales Gehirn eines oder mehrerer umliegende Baustellencontainer dient. Er übernimmt die Erfassung sämtlicher Sensordaten, die Steuerung der Aktoren und die Kommunikation mit der Datenbank sowie dem Proxmox-Server im Unternehmen.

#### **3.1.3 Technische Eigenschaften**

- Leistungsfähiger Prozessor
- 16 Gigabyte Arbeitsspeicher
- Netzwerkanschluss
- 4x USB-Anschlüsse
- GPIO-Pins
- Kompakt
- Langlebig
- Energieeffizient
- Kompatibel mit Ubuntu

#### **3.1.4 Begründung der Wahl**

Der Raspberry Pi war für CESC die perfekte Wahl, da er alle unsere technischen Voraussetzungen erfüllt. Dazu zählen ein USB-Anschluss für den Zigbee-Dongle, ein

Netzwerkanschluss für die Kommunikation mit Servern und der Datenbank sowie ein leistungsstarker Prozessor für die Ausführung unserer Programme. Ubuntu wurde als Betriebssystem gewählt, um eine bessere Wartbarkeit und Zuverlässigkeit aller eingesetzten Softwaredienste zu gewährleisten. Der Raspberry Pi ermöglicht somit eine zuverlässige Steuerung der Baustellencontainer.

### 3.1.5 Proxmox [Leopold Mistelberger]



Abbildung 3: Proxmox Virtual Environment Logo

Proxmox Virtual Environment (Proxmox VE) ist eine Plattform zur Servervirtualisierung, die Open Source ist. Sie erlaubt den Betrieb mehrerer virtueller Systeme auf einer einzigen physischen Hardware und findet häufig Anwendung in Unternehmensumgebungen.

Die Verwaltung wird zentral über eine webbasierte Benutzeroberfläche durchgeführt. Weitere Details finden Sie auf Proxmox offizieller Website (<https://www.proxmox.com>).

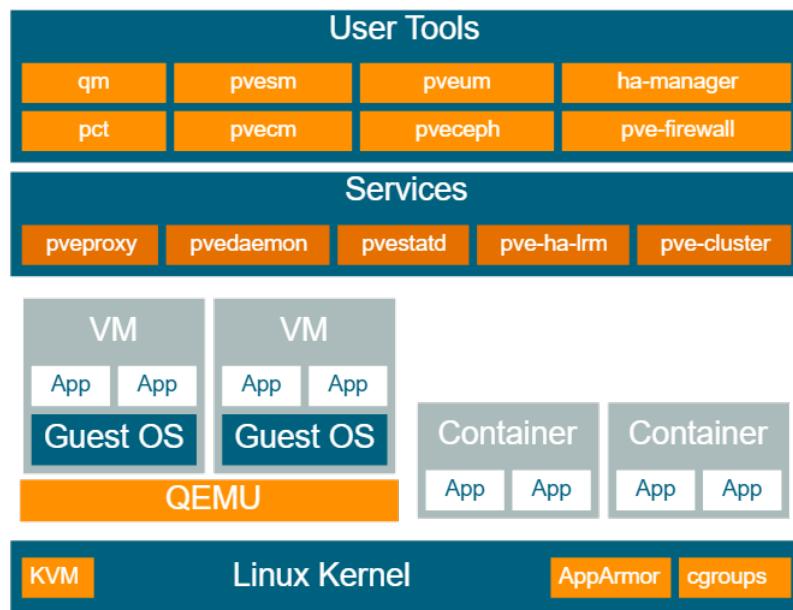


Abbildung 4: Architektur von Proxmox Virtual Environment

Proxmox läuft auf einem Linux-Hostsystem und verwendet verschiedene Virtualisierungstechnologien. Proxmox fügt zwischen der Hardware und den Anwendungen eine

Virtualisierungsschicht ein. Diese dient dazu, Systeme, die auf derselben Hardware laufen, voneinander zu trennen.

Virtuelle Maschinen (VMs) verhalten sich dabei jeweils wie ein eigener Computer mit einem eigenen Betriebssystem. Dadurch sind sie vollständig voneinander getrennt. CPU, Arbeitsspeicher und Speicherplatz werden im Vorhinein fest zugewiesen. Eine VM ist somit vollständig von anderen Systemen isoliert.

Zusätzlich unterstützt Proxmox Container auf Basis von Linux Containers (LXC). Container sind im Vergleich zu VMs leichter und schneller, da sie sich den Linux-Kernel des Servers teilen und arbeiten dadurch deutlich effizienter. Dadurch benötigen sie wenige Ressourcen und eignen sich sehr für den Betrieb Dienste oder Anwendungen.

Die Vorteile von Proxmox liegen unter anderem darin, dass keine Lizenzkosten anfallen, da Proxmox kostenlos verfügbar ist. Dadurch können die Betriebskosten eines Unternehmens gesenkt werden, insbesondere da Virtualisierungslösungen für virtuelle Maschinen zunehmend kostenintensiver werden. Zudem ermöglicht Proxmox eine effizientere Nutzung der vorhandenen Hardware, während die Systeme sauber voneinander getrennt bleiben. Durch das webbasierte Interface erhält der Benutzer eine gute Übersicht, da alle Systeme zentral verwaltet werden können.

4 dargestellt.

## 3.2 Kommunikation und Automatisierung [Leopold Mistelberger]

### 3.2.1 Zigbee [Leopold Mistelberger]

### 3.2.2 MQTT [Leopold Mistelberger]

#### Beschreibung

MQTT (Message Queuing Telemetry Transport) ist ein leichtgewichtiges Nachrichtenprotokoll, das nach dem sogenannten *Publish-Subscribe-Prinzip* arbeitet [?]. Dieses Protokoll wurde speziell für die Übertragung von Daten mit geringer Bandbreite oder instabiler Verbindung entwickelt. Es arbeitet ereignisbasiert und eignet sich besonders für Projekte, bei denen viele Geräte regelmäßig kleine Datenmengen wie Temperatur oder Luftfeuchtigkeit senden, wie beispielsweise in CESC.

## Komponenten und Funktionsweise

Bei MQTT gibt es drei zentrale Komponenten, die nach dem Pub-Sub-Prinzip arbeiten:

- **Publisher:** Ein Gerät oder Programm, das bestimmte Daten an den Broker sendet. In den meisten Fällen handelt es sich hierbei um einen Sensor.
- **Broker:** Meist ein zentraler Server, der alle Nachrichten empfängt und an die richtigen Empfänger weiterleitet.
- **Subscriber:** Ein Gerät oder Programm, das bestimmte Daten abonniert und diese empfängt.

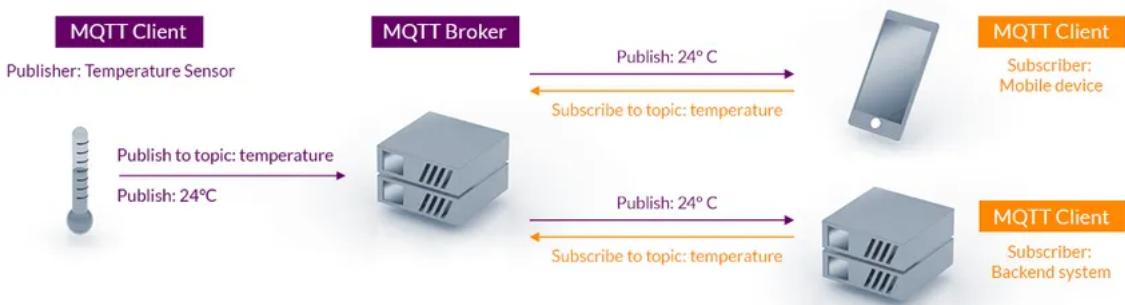


Abbildung 5: MQTT-Architektur: Publisher, Broker, Subscriber (Quelle: <https://mqtt.org/assets/img/mqtt-publish-subscribe.png>, Lizenz: CC0)

Die Daten werden in sogenannten *Topics* organisiert. Ein Publisher sendet Nachrichten an ein bestimmtes Topic, während Subscriber dieses Topic abonnieren. Der Broker übernimmt die Verteilung der Nachrichten, ohne dass Sender und Empfänger direkt kennen müssen.

## Beispielhafte Nutzung

Um Daten zu empfangen, muss ein Subscriber ein bestimmtes Topic abonnieren. Beispielsweise möchte man die Raumtemperatur eines Containers auslesen. Dazu wird ein Subscriber auf das entsprechende Topic gesetzt:

Listing 1: Subscriber für Raumtemperatur des Containers 147

```
1 mosquitto_sub -h <Broker-IP> -t "container/147/raum1/temperatur"
```

Neben dem Empfangen von Daten kann man auch Aktoren steuern, wie zum Beispiel die Heizung des Containers. Um diese einzuschalten, verwendet man den folgenden Publisher-Befehl:

Listing 2: Heizung des Containers 147 einschalten

```
1 mosquitto_pub -h <Broker-IP> -t "container/147/raum1/heizung" -m "ON"
```

## Vorteile von MQTT

MQTT bietet mehrere Vorteile gegenüber klassischen Kommunikationsprotokollen:

- Geringer Netzwerk- und Ressourcenverbrauch
- Zuverlässige Datenübertragung auch bei instabilen Verbindungen
- Klare Trennung von Sendern und Empfängern
- Hohe Skalierbarkeit bei vielen beteiligten Geräten
- Besonders geeignet für IoT- und Sensornetzwerke

### 3.2.3 Home Assistant [Leopold Mistelberger]

## 3.3 Container- und Laufzeitumgebung [Leopold Mistelberger]

### 3.3.1 Docker [Leopold Mistelberger]

### 3.3.2 GitHub Container Registry (ghcr.io) [Leopold Mistelberger]

## 3.4 Server- und Infrastrukturtechnologien [Elias Mahr]

## 3.5 Reverse-Proxy [Elias Mahr]

Ein Reverse-Proxy steht zwischen dem Webserver und dem Internet. Im Grunde fungiert es wie ein Türsteher für den Server. Anfragen werden von außen entgegengenommen und zum richtigen Dienst weitergeleitet(Frontend, Backend, Keycloak,...). Dadurch werden die echten Adressen verschleiert.

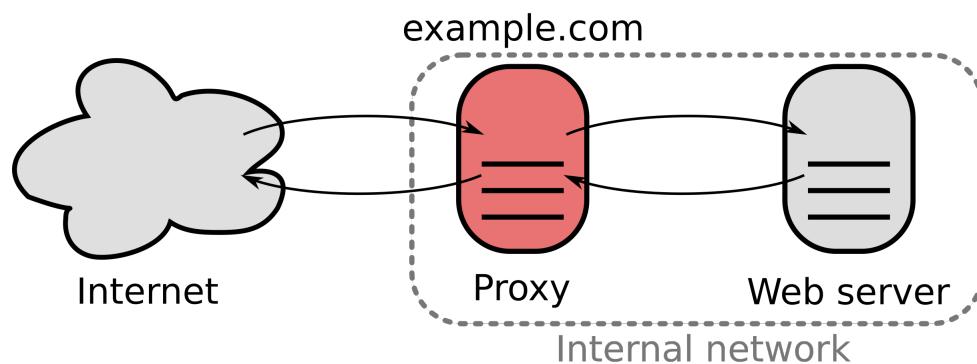


Abbildung 6: Reverse-Proxy (Quelle: [https://commons.wikimedia.org/wiki/File:Reverse\\_proxy\\_h2g2bob.svg](https://commons.wikimedia.org/wiki/File:Reverse_proxy_h2g2bob.svg), Lizenz: CC0)

### 3.5.1 Nutzen in der Diplomarbeit

Es gibt drei Hauptgründe, weshalb ein Reverse-Proxy eingebaut werden musste.

- Keycloaks https Redirects: Keycloak verwendet moderne Standards wie OpenID Connect. Diese Standards setzen auch eine HTTPS Redirect URL voraus. HTTPS setzt im Gegensatz zu HTTP die nötigen Sicherheitsstandards, um zum Beispiel Man-in-the-Middle-Angriffe zu verhindern.
- Port Chaos: Mit vielen verschiedenen Ports, wie 80 für das Frontend, 8080 für Keycloak und 5000 für das Backend wäre alles sehr unübersichtlich. Der User müsste sich 3 URLs merken.
- Sicherheit: Zur Reduzierung der Angriffsfläche sollten nur zwingend notwendige Ports öffentlich erreichbar sein. Jeder zusätzlich exponierte Dienst erhöht das Risiko potenzieller Angriffe. Interne Dienste, wie beispielsweise Backend-Services auf Port 5000, sind nicht von außen erreichbar und bleiben dadurch vor direktem Zugriff geschützt. Die Trennung zwischen öffentlich zugänglichen Komponenten und internen Services stellt eine grundlegende Sicherheitsmaßnahme dar. Auch wenn eine Webapplikation ausschließlich innerhalb des Intranets betrieben wird, ist der Einsatz zusätzlicher Sicherheitsmechanismen sinnvoll. Eine mehrschichtige Sicherheitsstrategie (Defense in Depth) trägt dazu bei, das Gesamtrisiko weiter zu reduzieren.

## 3.6 Auswahl der Technologie - Sicherheit(Authentifizierung)

Die Firma Herbstrofer legt sehr großen Wert auf Sicherheit. Die externe Bedienung von Heizung, Kühlung und Beleuchtung kann, bei unsachgemäßer Nutzung, zu erheblichen Mehrkosten und mitunter auch zu Schäden führen.

- Mitarbeiterenschutz: Bei voll eingeschalteter Kühlung an kalten Wintertagen kann es zu gesundheitlichen Schäden in Form von Verkühlungen bei Arbeitern/Arbeiterinnen führen, die auch mit einem Ausfall für ein paar Tage enden können und so der Firma Geld kosten.
- Brandgefahr: Unkontrollierte Heizbetriebe auf einer zu hohen Temperaturstufe können im schlimmsten Fall zu Bränden führen und so erheblichen Schaden anrichten und auch Personenschaden mit sich führen.
- Kosten: Strom wird immer teurer. Bei durchgehender Nutzung auch an Tagen, an denen niemand im Container ist, ist erstens der Schaden an der Umwelt durch extreme Ressourcenverschwendungen zu beachten. Zudem entstehen für das Unternehmen hohe, vermeidbare Stromkosten.

Um dieses Sicherheitsproblem zu lösen gibt es mehrere Ansätze.

### 3.6.1 Selbsterstellter Login Mechanismus

Eine nicht sichere, aber durchaus mögliche Option wäre eine selbstgeschriebene Login Komponente, die Passwort und Nutzer Gehasht in der Datenbank speichert.

#### Vorteile

Diese Option hätte den Vorteil einerseits sehr geringe Ressourcen zu beanspruchen. Die Komponente könnte man auch direkt ins Backend integrieren, dies wäre in unserem Fall, für nur einen Nutzer, nicht sehr zeitaufwändig. Alles könnte direkt nach dem Wunsch der Firma gebaut werden.

#### Nachteile

Die Sicherheit bei einer selbsterstellten Komponente ist für Firmen nicht ausreichend. Selbst erstellte Komponenten sind möglicherweise dafür nicht geeignet. Für erfahrene

Hacker könnte so ein Login ein leichtes sein um als Admin auf die Webseite zu kommen mit zum Beispiel einem Brute Force Angriff. Ein weiterer Nachteil ist, dass es sehr kompliziert ist und von einem externen Programmierer gemacht werden müsste, wenn der Login erweitert werden würde.

### 3.6.2 Vorgefertigter Login (Keycloak [Elias Mahr])

Das Ziel ist immer, so viele vorgefertigte Systeme wie möglich zu verwenden, solange sie den erforderlichen Anforderungen entsprechen. Eines davon ist in diesem Fall Keycloak.



Abbildung 7: Keycloak Logo

#### Vorteile

Keycloak ist ein Open Source System, das heißt, es ist öffentlich zugänglich und kostenlos. Diese Software übernimmt die sichere Anmeldung und die Rechteverwaltung für eine Anwendung übernimmt. Die Anwendungen schicken die Anmeldung an Keycloak und müssen das Speichern und das Prüfen von Benutzerkonten nicht mehr selbst erledigen. So kann die Anwendung sich auf das Wesentliche konzentrieren, während Keycloak die Anmeldung und die Rechte im Hintergrund mit Sicherheitsstandards für Unternehmen regelt. In der grafischen Oberfläche des Systems ist das Erweitern der Funktionen, zum Beispiel das neue Anlegen von Usern oder eine neue Rechtevergabe, auch für Menschen ohne Vorkenntnisse im Programmieren sehr einfach. Sollte sich die Firma entscheiden unsere Webapp zu erweitern ist das sehr von Vorteil, da sie keine eigene Programmierabteilung im Haus hat.

#### Nachteile

Im Vergleich zu einer selbsterstellten Lösung hat Keycloak einen sehr hohen Ressourcenverbrauch, teilweise auch für Funktionen, die in unserer Webapp nicht implementiert sind. Das Erstellen eines Logins mit Keycloak ist sehr zeitintensiv und nimmt mehrere Tage in Anspruch, besonders für Entwickler, die noch nie damit in Berührung gekommen sind.

## 3.7 Cronjob

Um die Container wirklich voll automatisch ein- und ausschalten zu können, kam für uns nur die Verwendung eines Cronjobs auf dem Server in Frage. Im Unterricht hatten wir bereits davon gehört. Ein Cron-Daemon ist ein Linux-Programm, das meistens Shell Skripte zeitgesteuert auslöst, sogenannte Cronjobs. Somit kann man Aufgaben nach einem Zeitplan immer wiederholend ausführen.

### 3.7.1 Syntax

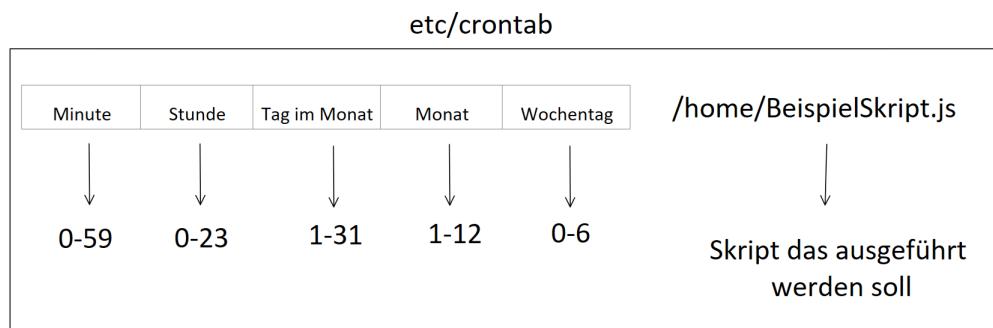


Abbildung 8: Cronjob Syntax

Jeder Cronjob ist eine Zeile in dem crontab File. Diese Zeile definiert mit der oben gezeigten Syntax, wann und wie oft welches Skript ausgeführt werden soll. So können auch komplizierte Zeitpläne für die Durchführung der Schaltungen gebaut werden. Mit `/` können Schritte definiert werden, wie zum Beispiel bei Stunde = `/6`. Das würde bedeuten, dass das angeführte Skript bzw. Programm jede 6. Stunde ausgeführt wird. Mit dem Operator `-` können Bereiche definiert werden, z.B. Wochentag = `0-6`. Das würde bedeuten, dass das ausgeführte Skript bzw. Programm täglich ausgeführt wird.

## 3.8 Frontend

### 3.8.1 Use Cases für die Webapp

#### Baustellenübersicht anzeigen

Als Geschäftsführer der Firma Herbstrofer möchte ich alle derzeitigen Baustellen auf einen Blick sehen, weil ich schnell zwischen verschiedenen Baustellen wechseln möchte.

Akzeptanzkriterien:

- Alle Baustellen werden auf einen Blick übersichtlich angezeigt.
- Wenn ich eine Baustelle auswähle, sollen in einer Detailansicht alle Container der Baustelle aufgelistet werden.
- Es sollte kein Problem sein, eine Baustelle hinzuzufügen oder zu entfernen.

### **Containerstatus überwachen**

Als Geschäftsführer der Firma Herbstrofer möchte ich den aktuellen Temperaturstatus aller Container einer Baustelle sehen, weil ich die sinnvolle Schaltung der Heizung kontrollieren muss und darauf achte, dass meine Mitarbeiter nicht beispielsweise im Winter die Türe offen lassen.

Akzeptanzkriterien:

- Innen- und Außentemperatur werden in Echtzeit angezeigt.
- Luftfeuchtigkeit wird auch klein angezeigt.

### **Anmeldung**

Als Geschäftsführer der Firma Herbstrofer möchte ich mich sicher am System anmelden wenn ich gewisse Funktionen nutzen möchte, weil sonst Mitarbeiter die Funktionen ausnützen und somit Schäden verursachen könnten.

Akzeptanzkriterien:

- Zur Anmeldung wird etwas Vorgefertigtes genommen.
- Alles sollte auf unternehmerischem Standard sein.
- Nach erfolgreicher Anmeldung sollten die Urlaube bearbeitet werden können.

### **Türzugriffe protokollieren**

Als Geschäftsführer der Firma Herbstrofer möchte ich alle Türöffnungen sehen, weil ich Einbrüche oder unbefugten Zugriff nachvollziehen muss.

Akzeptanzkriterien:

- Die Bewegungen der Türe werden aufgezeichnet und können im Nachhinein eingesehen werden. Es wird unterschieden zwischen Türbewegung in der Arbeitszeit und Türbewegung außerhalb der Arbeitszeit.
- Im Frontend gibt es eine Visualisierung dieser Daten.

### **Sensorverlaufsdaten analysieren**

Als Geschäftsführer der Firma Herbstrofer möchte ich vergangene Sensordaten visualisiert sehen, weil ich eventuelle Kosten nachvollziehen möchte.

Akzeptanzkriterien:

- Daten werden mit einem vorgefertigten Tool, zum Beispiel Grafana oder Plottly, visualisiert.
- Es soll auf verschiedene Zeiträume gefiltert werden können.

### **Sichere Kommunikation gewährleisten**

Als Geschäftsführer der Firma Herbstrofer möchte ich dass alle Verbindungen geschützt sind, weil verhindert werden sollte, dass Hacker an geheime Daten kommen.

Akzeptanzkriterien:

- Nginx ist mit Reverse Proxy implementiert und es sind nur wirklich notwendige Ports öffentlich.
- Let's Encrypt Zertifikat

### **Energiekosten reduzieren**

Als Geschäftsführer der Firma Herbstrofer möchte ich Heizung und Kühlung automatisch steuern, weil ich unnötige Energiekosten vermeiden möchte.

Akzeptanzkriterien:

- Container werden an Betriebsurlauben nicht beheizt/gekühlt
- Container werden an Feiertagen nicht beheizt/gekühlt.
- Container werden nur während der Arbeitszeit beheizt/gekühlt.
- Dafür wird ein Cronjob am Server benutzt.

## Mobiler Zugriff

Als Geschäftsführer der Firma Herbstrofer möchte ich die Webapp auf meinem Smartphone nutzen, weil ich für spontanes Nachsehen, auf zum Beispiel einer Baustelle, keinen Laptop/PC zur Verfügung habe

Akzeptanzkriterien:

- Die Webapp ist responsive.
- Alle Features sind mobil genauso verfügbar.

### 3.8.2 Use Case Diagram

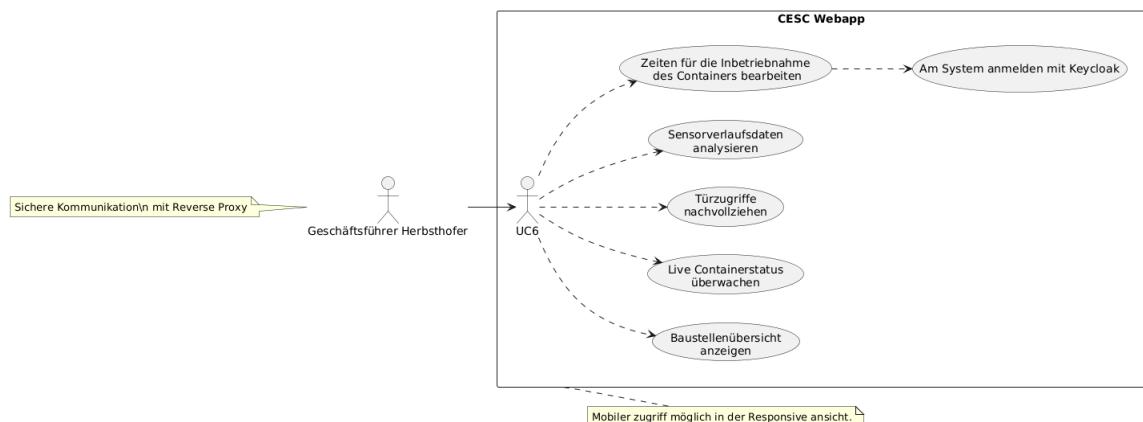


Abbildung 9: Use Case Diagram

## 3.9 Api

## 3.10 Backend

### 3.10.1 ASP.NET Core & C#



Abbildung 10: ASP.NET Core Logo (Quelle: campusMVP GitHub Repository, <https://github.com/campusMVP/dotnetCoreLogoPack>, Lizenz: CC BY-SA 4.0)

ASP.NET Core ist ein modernes, plattformübergreifendes Framework von Microsoft zur Entwicklung von Webanwendungen und APIs. Es ist die Weiterentwicklung des klassischen ASP.NET Frameworks und wurde von Grund auf neu entwickelt, um performanter, modularer und flexibler zu sein.

## Grundlagen und Eigenschaften

ASP.NET Core ist ein **Open-Source Framework**, das auf der .NET-Plattform basiert. Im Gegensatz zu seinem Vorgänger läuft ASP.NET Core nicht nur auf Windows, sondern auch auf **Linux und macOS**, was es zu einer idealen Wahl für containerisierte Anwendungen macht.

Die wichtigsten Merkmale von ASP.NET Core sind:

- **Hohe Performance:** ASP.NET Core gehört zu den schnellsten Web-Frameworks weltweit und ist besonders bei vielen gleichzeitigen Anfragen sehr effizient.
- **Plattformunabhängigkeit:** Durch die Cross-Platform-Unterstützung kann die Anwendung auf verschiedenen Betriebssystemen entwickelt und deployed werden.
- **Modular aufgebaut:** Das Framework verwendet ein schlankes Kern-System, das nur die wirklich benötigten Komponenten lädt.
- **Dependency Injection:** Ein eingebautes DI-System ermöglicht eine saubere Architektur und bessere Testbarkeit.
- **Cloud-ready:** Perfekt geeignet für Cloud-Deployments und Container-Orchestrierung mit Docker.

## C# als Backend-Sprache

Die Programmiersprache **C#** (C-Sharp) ist eine moderne, objektorientierte Sprache, die speziell für die .NET-Plattform entwickelt wurde. Sie kombiniert die Leistungsfähigkeit von C++ mit der Einfachheit von Java.

Vorteile von C# für Backend-Entwicklung:

- **Type Safety:** Durch statische Typisierung werden viele Fehler bereits zur Compile-Zeit erkannt, nicht erst zur Laufzeit.
- **Async/Await Pattern:** Native Unterstützung für asynchrone Programmierung, was besonders bei I/O-intensiven Operationen wie Datenbankzugriffen wichtig ist.

- **LINQ (Language Integrated Query):** Ermöglicht das einfache Filtern und Transformieren von Datensammlungen direkt in C#.
- **Moderne Sprachfeatures:** Pattern Matching, Records, Nullable Reference Types und viele weitere Features, die den Code sauberer und sicherer machen.
- **Große Community:** Als etablierte Enterprise-Sprache gibt es umfangreiche Dokumentation und Community-Support.

## RESTful API Architektur

Das Backend des CESC-Projekts ist als **REST API** (Representational State Transfer) konzipiert. REST ist ein Architekturstil für verteilte Systeme, der auf HTTP-Methoden basiert:

- **GET:** Daten abrufen (z.B. Container-Informationen, Sensordaten)
- **POST:** Neue Ressourcen erstellen (z.B. neue Kalenderregel)
- **PUT:** Bestehende Ressourcen aktualisieren (z.B. Container-Namen ändern)
- **DELETE:** Ressourcen löschen (z.B. Kalenderregel entfernen)

Die API kommuniziert über **JSON** (JavaScript Object Notation) als Datenformat, da dies sowohl menschen- als auch maschinenlesbar ist und sich perfekt für die Kommunikation zwischen Frontend und Backend eignet.

## Warum ASP.NET Core für CESC?

Die Wahl von ASP.NET Core für das Backend der CESC-Anwendung basiert auf mehreren Überlegungen:

- **Performance bei IoT-Daten:** Die Anwendung muss in der Lage sein, regelmäßig Sensordaten von mehreren Containern zu verarbeiten. ASP.NET Core bietet hierfür die nötige Performance.
- **Docker-Kompatibilität:** Da die gesamte Anwendung containerisiert deployed wird, ist die plattformunabhängige Natur von ASP.NET Core essentiell.
- **Enterprise-Standard:** Die Firma Herbstrofer setzt auf bewährte, sichere Technologien. ASP.NET Core wird von Microsoft Enterprise-Support unterstützt.

- **Integration mit Keycloak:** Die eingebaute JWT-Authentication in ASP.NET Core vereinfacht die Integration mit dem Keycloak Identity Provider erheblich.
- **Zukunftssicherheit:** Microsoft investiert kontinuierlich in .NET und ASP.NET Core. Die aktuell verwendete Version .NET 8.0 hat Long-Term-Support bis November 2026.
- **Entwickler-Produktivität:** Durch die starke Typisierung, IntelliSense-Unterstützung und umfangreiche Tooling-Unterstützung in Visual Studio und VS Code können Features schneller und fehlerfreier entwickelt werden.

### 3.10.2 Swagger/OpenAPI - API Dokumentation



Abbildung 11: Swagger Logo (Quelle: <https://commons.wikimedia.org/wiki/File:Swagger-logo.png>, Lizenz: CC BY-SA 4.0)

Swagger ist ein Framework zur Dokumentation und zum Testen von REST APIs. Es generiert automatisch eine interaktive Dokumentation, die alle verfügbaren Endpoints, ihre Parameter und Rückgabewerte übersichtlich darstellt.

#### Beschreibung und Zweck

OpenAPI (früher bekannt als Swagger) ist ein Standard zur Beschreibung von REST APIs. Die Spezifikation definiert, wie eine API strukturiert ist, welche Endpoints existieren und welche Daten sie erwarten und zurückgeben. In ASP.NET Core wird dies durch die **Swashbuckle.AspNetCore** Bibliothek realisiert.

Die Hauptvorteile von Swagger sind:

- **Automatische Dokumentation:** Die API-Dokumentation wird direkt aus dem Code generiert und ist immer aktuell.
- **Interaktives Testing:** Entwickler können API-Endpoints direkt im Browser testen, ohne zusätzliche Tools wie Postman zu benötigen.
- **Klare Übersicht:** Alle verfügbaren Endpoints, HTTP-Methoden und Datenmodelle werden strukturiert angezeigt.

- **Client-Code-Generierung:** Aus der OpenAPI-Spezifikation können automatisch Client-Bibliotheken für verschiedene Programmiersprachen generiert werden.

## Funktionsweise in CESC

Im CESC-Backend ist Swagger so konfiguriert, dass es nur im **Development-Modus** verfügbar ist. In der Production-Umgebung ist die Swagger-UI aus Sicherheitsgründen deaktiviert, da sie Details über die API-Struktur preisgibt.

Die Swagger-UI zeigt alle verfügbaren Endpoints übersichtlich gruppiert nach Controllern:

- **CalendarController:** Endpoints zur Verwaltung von Kalenderregeln (GET, POST, DELETE)
- **ContainersController:** Endpoints zum Abrufen und Verwalten von Container-Daten
- **SensorsController:** Endpoints für aktuelle Sensordaten
- **PlugControlController:** Endpoints zur Steuerung von Steckdosen (Heizung, Kühlung, Licht)
- **DoorsController:** Endpoints zum Abrufen des Türstatus

## HTTP-Methoden in der Swagger-UI

Swagger visualisiert die verschiedenen HTTP-Methoden farblich, um sie auf einen Blick unterscheidbar zu machen:

- **GET (Blau):** Daten abrufen, z.B. GET /api/containers
- **POST (Grün):** Neue Ressourcen erstellen, z.B. POST /api/calendar/{containerId}
- **PUT (Orange):** Ressourcen aktualisieren
- **DELETE (Rot):** Ressourcen löschen, z.B. DELETE /api/calendar/{containerId}

## Authentication in Swagger

Da die CESC-API durch Keycloak geschützt ist, muss auch in Swagger ein JWT-Token für geschützte Endpoints angegeben werden. Swagger bietet hierfür einen „**Authorize**“-Button, über den ein Bearer-Token eingegeben werden kann.

Nach erfolgreicher Anmeldung bei Keycloak kann das erhaltene JWT-Token in Swagger eingetragen werden. Alle nachfolgenden API-Aufrufe werden dann automatisch mit diesem Token authentifiziert.

## Vorteile für die Entwicklung

Während der Entwicklungsphase von CESC erwies sich Swagger als äußerst hilfreich:

- **Frontend-Backend-Abstimmung:** Das Frontend-Team konnte die verfügbaren Endpoints und deren Datenstrukturen direkt einsehen, ohne in den Backend-Code schauen zu müssen.
- **Schnelles Testing:** Neue Endpoints konnten sofort nach der Implementierung über Swagger getestet werden, bevor sie ins Frontend integriert wurden.
- **Fehlerbehebung:** Bei Problemen konnte schnell überprüft werden, ob das Backend korrekte Daten zurückliefert oder ob der Fehler im Frontend liegt.
- **Dokumentation für die Firma:** Die Swagger-Dokumentation dient auch als Referenz für zukünftige Entwickler oder Wartungspersonal der Firma Herbstrofer.

# 4 Umsetzung

## 4.1 Systemarchitektur [Elias Mahr]

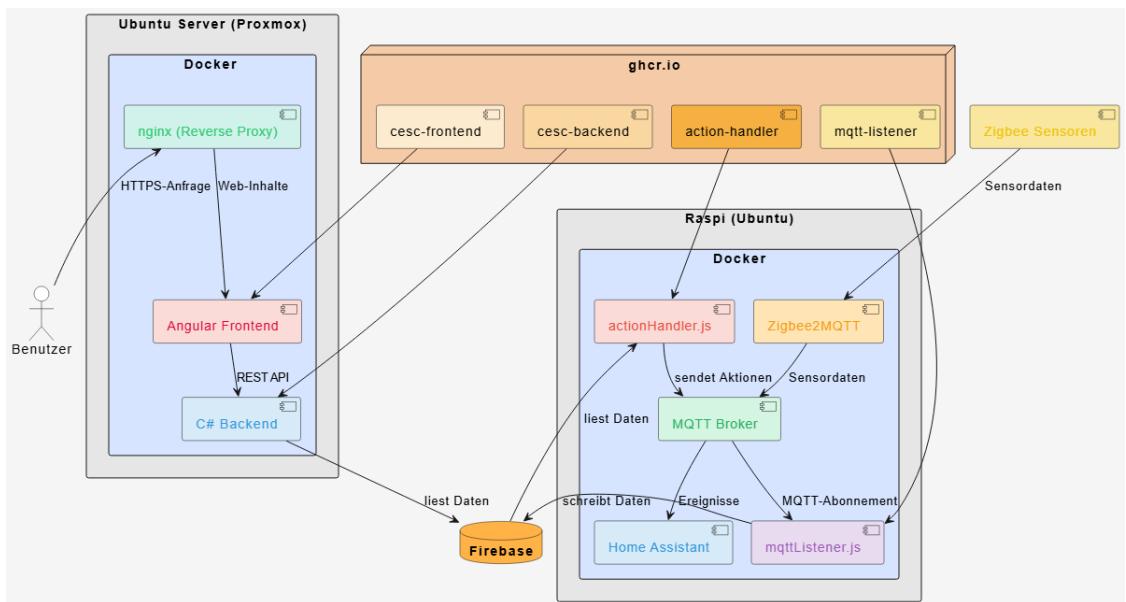


Abbildung 12: Systemarchitektur

## 4.2 Containerarchitektur [Elias Mahr]

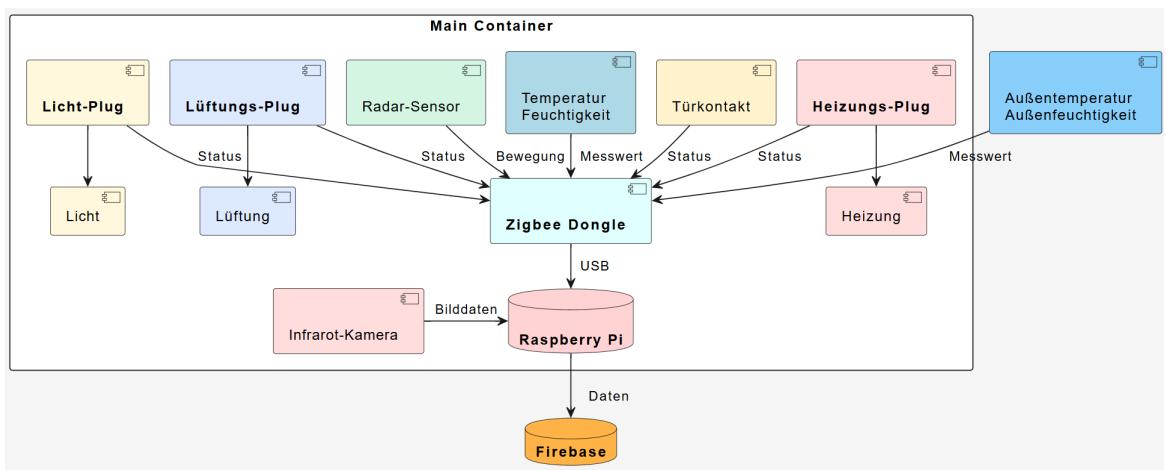


Abbildung 13: Containerarchitektur

## 4.3 Keycloak [Elias Mahr]

### 4.3.1 Einbindung ins Frontend

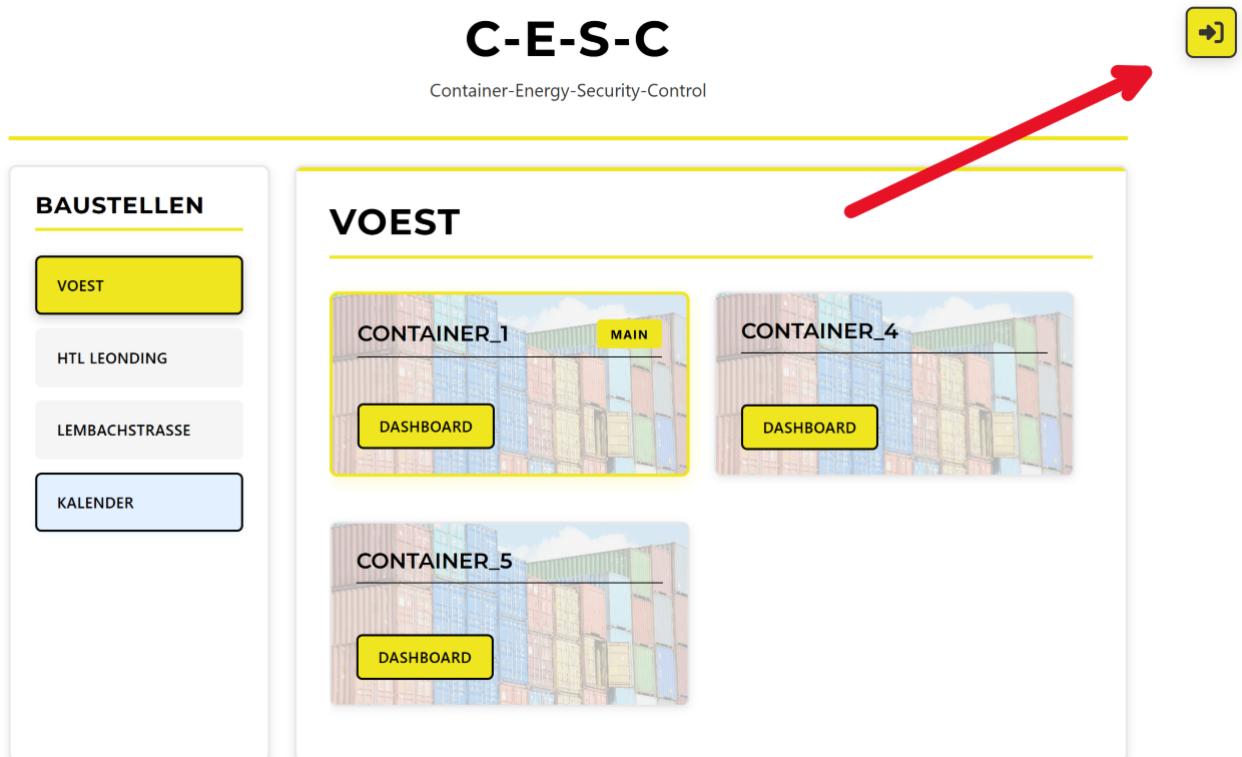


Abbildung 14: Button im Frontend

Um die Webapp möglichst unkompliziert und benutzerfreundlich zu halten, ist der Button, wie im Screenshot ersichtlich, rechts oben in der Ecke platziert. Unabhängig vom Scroll-Zustand oder der aktuell ausgewählten Page bleibt er auch immer an dieser Stelle, um ein schnelles, unkompliziertes Anmelden zu gewährleisten.



(a) Anmelde-Button



(b) Abmelde-Button

Abbildung 15: Login- und Logout-Buttons

Das Icon stammt aus der Font Awesome Bibliothek:

```
<i class="fas fa-sign-in-alt"></i>
```

Bei Klick darauf wird man weitergeleitet zum Pfad:

`https://192.168.20.202/auth/realms/cesc/protocol/openid-connect`

`/auth?client_id=cesc-frontend&redirect_uri=https...` Unter dieser Url wird der Standard-Login-Dialog von Keycloak geöffnet. Nach erfolgreicher Anmeldung bekommt der User seinen Token, wird automatisch wieder zur vorher besuchten Seite umgeleitet und es werden alle Funktionen der Webseite freigeschaltet.

### 4.3.2 Der Realm

Ein Realm in Keycloak ist ein isolierter Sicherheitsbereich. Er umfasst Benutzer, Rollen, Clients und Policies für eine bestimmte Anwendung. Der Realm namens “cesc” definiert die gesamte Identifikation und Authorisierung für die Weboberfläche. Zurzeit ist dieser Bereich möglichst klein gehalten, jedoch vollkommen ausreichend. Ein Admin User(herbsthofer) steuert alles, in Zukunft kann man jedoch leicht noch andere Benutzer/Rollen hinzufügen.

#### Realm Einstellungen

Listing 3: Keycloak Realm Konfiguration

```

1  {
2      "realm": "cesc",
3      "enabled": true,
4      "sslRequired": "external",
5      "registrationAllowed": false,
6      "loginWithEmailAllowed": true,
7      "duplicateEmailsAllowed": false,
8      "resetPasswordAllowed": true,
9      "editUsernameAllowed": false,
10     "bruteForceProtected": true
11 }
```

In den wichtigsten Einstellungen wird unter anderem definiert, dass für alle externen Verbindungen SSL vorgeschrieben ist. Die Zeile `bruteForceProtected: true` ist eine einfache Absicherung gegen Brute Force Angriffe auf Keycloak. Durch sie wird ein Account nach mehreren fehlgeschlagenen Anmeldungen temporär gesperrt. Um zukünftigen Admins der Webseite eine leichte und unkomplizierte Anmeldung zu ermöglichen, ist auch das Einloggen mit der E-Mail-Adresse erlaubt.

#### Rollen

Listing 4: Keycloak Realm Konfiguration

```

1  {
2      "roles": [
3          "realm": [
4              {
5                  "name": "admin",
6                  "description": "Administrator role for CESC
7                      application",
8                  "composite": false,
9                  "clientRole": false
10             }
11         ]
12     }
13 }
```

Der aktuell einzige Benutzer "admin" hat sozusagen uneingeschränkte Rechte. Durch die Zeile "clientRole : false" ist dieser Benutzer auch nicht auf einen bestimmten Client beschränkt, sondern hat im ganzen Realm auf alles Zugriff.

## Benutzer

Listing 5: Keycloak Realm Konfiguration

```

1  {
2      "users": [
3          {
4              "username": "herbsthofer",
5              "enabled": true,
6              "emailVerified": true,
7              "firstName": "Admin",
8              "lastName": "Herbsthofer",
9              "email": "herbsthofer@example.com",
10
11             "realmRoles": [
12                 "admin"
13             ]
14         }
15     ]
16 }
```

Der Zurzeit einzige User `herbsthofer` erhält durch die Zuweisung der admin Rolle, "`realmRoles` " : [" admin "] alle Administratorrechte.

## Clients

Listing 6: Keycloak Realm Client Konfiguration

```

1  {
2      "clients": [
```

```

3      {
4          "clientId": "cesc-frontend",
5          "name": "CESC Frontend Application",
6          "enabled": true,
7          "publicClient": true,
8          "protocol": "openid-connect",
9          "standardFlowEnabled": true,
10         "redirectUris": [
11             "https://192.168.20.202/*",
12             "http://localhost:4200/*"
13         ],
14         "webOrigins": [
15             "https://192.168.20.202",
16             "http://localhost:4200"
17         ]
18     },
19     {
20         "clientId": "cesc-backend",
21         "name": "CESC Backend API",
22         "enabled": true,
23         "publicClient": false,
24         "bearerOnly": true,
25         "protocol": "openid-connect"
26     }
27 ]
28 }
```

Für unsere Anwendung werden nur zwei Clients benötigt. Der `cesc-frontend` Client ist der öffentliche Client (`publicClient: true`) für die Angular Anwendung. Die `redirectUris` und `webOrigins` legen die erlaubten URLs für die Produktion (192.168.20.202) und die lokale Entwicklung (localhost:4200) fest. Der `cesc-backend` Client ist als vertraulicher Client gemacht, da er keine Login Page bereitstellt. (`publicClient: false`).

## 4.4 Reverse Proxy [Elias Mahr]

Nginx ist als eigener Docker Container mit `nginx.conf` implementiert. Nur der Port 443 ist öffentlich mit Let's Encrypt-Zertifikaten. Die Ports werden pfadbasiert weitergeleitet:

- <http://Keycloak:8080> → location /auth

Listing 7: Nginx Konfiguration /auth

```

1      location /auth {
2          proxy_pass http://keycloak:8080;
3          proxy_set_header Host $host;
4          proxy_set_header X-Real-IP $remote_addr;
5          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
6          proxy_set_header X-Forwarded-Proto https;
```

```
7         proxy_set_header X-Forwarded-Host $host;
8         proxy_set_header X-Forwarded-Port 443;
9         proxy_buffer_size 128k;
10        proxy_buffers 4 256k;
11        proxy_busy_buffers_size 256k;
12    }
```

- http://frontend:80 → location /

Listing 8: Nginx Konfiguration /

```
1     location / {
2         proxy_pass http://frontend:80;
3     }
```

- http://backend:5000 → location /api

Listing 9: Nginx Konfiguration /api

```
1     location /api {
2         proxy_pass http://backend:5000;
3     }
```

## 4.5 Angular

Da das Projekt nicht nur am Desktop, sondern auch auf mobilen Devices zum Einsatz kommen soll, wurde die gesamte Applikation in einem responsive Design entwickelt.

## 4.5.1 Components

### start-list-container



Abbildung 16: Startlist Component

Die “StartListContainerComponent” dient als Startseite der Anwendung. Beim Öffnen wird eine komplette Übersicht über alle in der Datenbank gespeicherten Baustellen angezeigt. Die Seite ist in drei wesentliche Abschnitte unterteilt:

- **Titel**

Ganz oben sieht man sofort den Titel unserer Arbeit(C-E-S-C) mit dem ausgeschriebenen Titel darunter(Container-Energy-Security-Controll).

- **Baustellenübersicht**

Links auf 1/4 der Breite sind alle Baustellen mit ihrem in der Datenbank gespeicherten Namen aufgelistet. Die ausgewählte Baustelle wird immer gelb mit Rahmen markiert, so auch beim Überfahren mit der Maus. In diesem Fall wird das Element auch ein wenig eingerückt. Immer am Ende des Abteils ist das Kalenderelement. Dieses ist blau eingefärbt, damit man es gut von den Baustellen unterscheiden kann.

- **Containerübersicht**

Rechts bis mittig zu 3/4 der Breite ist die Übersicht der jeweiligen Container zu einer Baustelle. Mit dem Wechsel der Baustelle wird auch die Ansicht der Container aktualisiert. Bei dieser Ansicht werden Hauptcontainer optisch hervorgehoben.



(a) Main Container

(b) normaler Container

Abbildung 17: Unterschiedliche Darstellung der Container

Hauptcontainer erkennt man am gelben Rand und der Kennzeichnung “MAIN” neben dem Containernamen. Mit Klick auf eine Componente wird man zum jeweiligen Dashboard weitergeleitet. Der Name(im Screenshot z.B. Container\_1), wird aus der Datenbank genommen und kann natürlich auf Wunsch geändert werden. Defaultmäßig ist es zurzeit “Container\_n”.

## Calendar

Datum	Bundesland	Feiertag / Betriebsurlaub
05.07.2025	-	Sommer-Betriebsurlaub (05.07.2025 - 14.07.2025) <input type="button" value="Bearbeiten"/>
23.12.2025	-	Winter-Betriebsurlaub <input type="button" value="Bearbeiten"/>

Abbildung 18: Kalenderelement

Die **Calender** Componente erscheint, wenn man in der start-list-component keine Baustelle, sondern das blaue Kalenderelement antippt. Als erstes sieht man sofort den Kalender, bei dem immer das aktuelle Monat ausgewählt ist. Im Kalender wird zwischen zwei Arten von Urlauben unterschieden.



Abbildung 19: Unterschied der Urlaube

Betriebsurlaub wird gelb markiert und ein Feiertag grau. An diesen markierten Tagen werden die Container nicht eingeschaltet. Unter dem Kalender findet man noch eine Liste mit allen Urlauben für das aktuelle Jahr und den Betriebsurlauben. Die Feiertage werden von einer API geladen. Nach erfolgreicher Anmeldung wird der “bearbeiten” Button freigeschaltet und Umänderungen können vorgenommen werden.



Abbildung 20: Bearbeiten Button

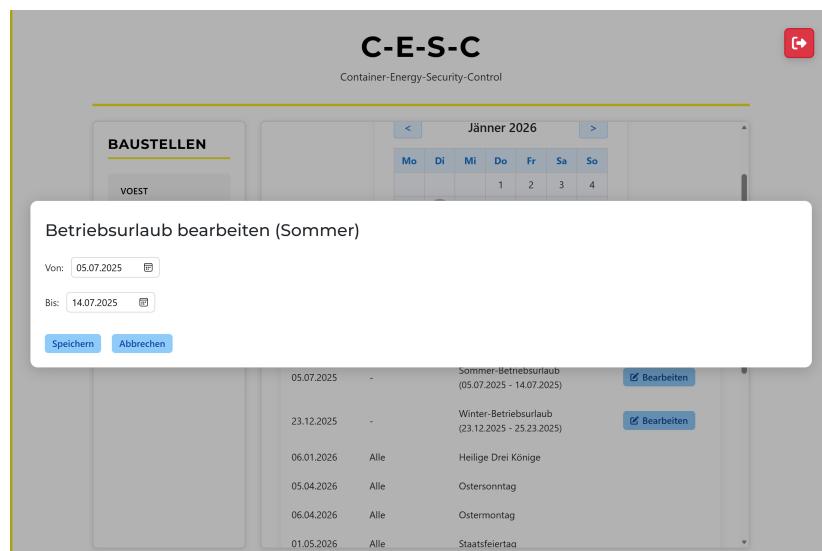


Abbildung 21: Betriebsurlaube bearbeiten

Bei Klick auf den Button kann der Betriebsurlaub bearbeitet werden. Die neuen Daten werden direkt in die Datenbank gespeichert und sofort vom Cronjob beachtet. Dabei wird natürlich auch geprüft, ob die Daten im korrekten Format eingaben wurden. Um dies zu erleichtern wird beim Eingeben eines neuen Datums sofort ein kleiner Kalender geöffnet, auf dem ein neues Datum ausgewählt werden kann. Dies ist eine vorgefertigte Funktion von html wenn man die input type date verwendet.

```
Listing 10: label im html
```

```

1   {
2     <label>Von: <input type="date"
3       [(ngModel)]="modalFrom"></label>

```

## dashboard

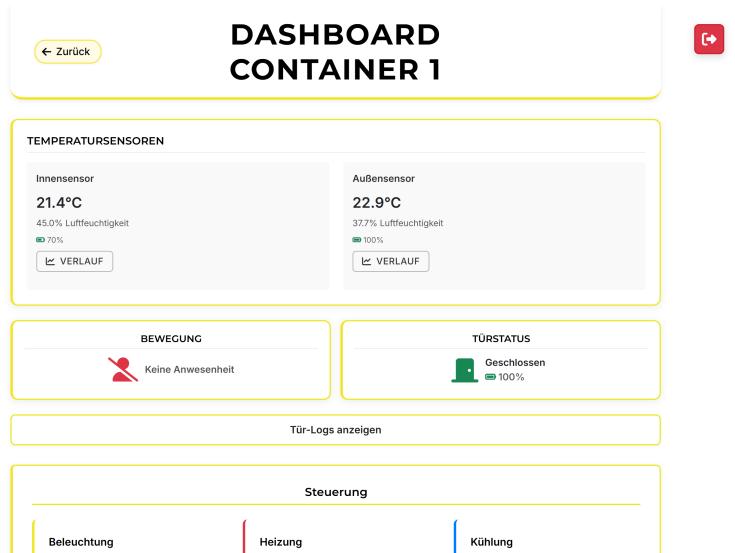


Abbildung 22: Dashboard

Wenn man auf den Button `Dashboard` in der `start-list-componente` bei einem Container klickt, kommt man zur `dashboard` Komponente. Diese Hauptfunktion der Applikation ist für jeden Container individuell abhängig, von den jeweiligen Daten. Der Aufbau ist jedoch überall gleich. Die Komponente ist so gestaltet, dass sie jederzeit erweitert werden kann. Für jeden Sensor ist ein eigener abgetrennter Abschnitt auf der Webseite. Bei Erweiterung des Systems durch die Firma kann ein weiterer Abschnitt jederzeit leicht hinzugefügt werden, ohne Rücksicht auf die anderen Componenten nehmen zu müssen.

Ganz oben in der Komponente sieht man die Überschrift mit dem Namen des Containers. Der erste Abschnitt ist eine übersichtliche Temperaturanzeige von Innen- und Außen-sensor. Fettgedruckt ist die Temperatur, da das die wichtigste Anzeige im Container ist. Gleich darunter steht etwas kleiner die Luftfeuchtigkeit und abschließend noch die Anzeige der übrigen Batterie, um rechtzeitig zu erkennen, wann die Batterien gewechselt werden sollten. Ein detaillierter Verlauf kann dann noch mit dem Button "Verlauf" angezeigt werden. Dazu aber dann in dieser Komponente mehr.

## door-logs

Tür-Logs		
Datum	Zeit geöffnet	Zeit geschlossen
3.12.2025	16:12	16:38
2.12.2025	13:17	13:17
2.12.2025	13:17	13:17
2.12.2025	13:05	13:13
2.12.2025	12:35	12:36
2.12.2025	12:27	12:28
1.12.2025	18:49	12:18
1.12.2025	18:49	18:49
1.12.2025	18:49	18:49
1.12.2025	18:37	18:49
1.12.2025	18:37	18:37
1.12.2025	18:33	18:33

Abbildung 23: Door Logs Komponente

## plotly-chart

- can- und Logout-Buttonslender-editor
- dashboard
- door-logs
- kalender
- plotly-chart
- start-list-container

## Services

Siehe tolle Daten in Tab. 1.

Siehe und staune in Abb. ??.

Dann betrachte den Code in Listing ??.

Regular Customers	Random Customers
Age	20-40
Education	university

Tabelle 1: Ein paar tabellarische Daten

# 5 Zusammenfassung

Aufzählungen:

- Itemize Level 1
  - Itemize Level 2
    - Itemize Level 3 (vermeiden)
- 1. Enumerate Level 1
  - a. Enumerate Level 2
    - i. Enumerate Level 3 (vermeiden)

**Desc** Level 1

**Desc** Level 2 (vermeiden)

**Desc** Level 3 (vermeiden)

# Glossar

**GUID** Globally Unique Identifier

# **Literaturverzeichnis**

# Abbildungsverzeichnis

1	Herbsthofer Logo . . . . .	1
2	Zustandsdiagramm . . . . .	3
3	Proxmox Virtual Environment Logo . . . . .	6
4	Architektur von Proxmox Virtual Environment . . . . .	6
5	MQTT-Architektur: Publisher, Broker, Subscriber (Quelle: <a href="https://mqtt.org/assets/img/mqtt-publish-subscribe.png">https://mqtt.org/assets/img/mqtt-publish-subscribe.png</a> , Lizenz: CC0) . . . . .	8
6	Reverse-Proxy (Quelle: <a href="https://commons.wikimedia.org/wiki/File:Reverse_proxy_h2g2bob.svg">https://commons.wikimedia.org/wiki/File:Reverse_proxy_h2g2bob.svg</a> , Lizenz: CC0) . . . . .	10
7	Keycloak Logo . . . . .	12
8	Cronjob Syntax . . . . .	13
9	Use Case Diagram . . . . .	16
10	ASP.NET Core Logo (Quelle: campusMVP GitHub Repository, <a href="https://github.com/campusMVP/dotnetCoreLogoPack">https://github.com/campusMVP/dotnetCoreLogoPack</a> , Lizenz: CC BY-SA 4.0) . . . . .	16
11	Swagger Logo (Quelle: <a href="https://commons.wikimedia.org/wiki/File:Swagger-logo.png">https://commons.wikimedia.org/wiki/File:Swagger-logo.png</a> , Lizenz: CC BY-SA 4.0) . . . . .	19
12	Systemarchitektur . . . . .	22
13	Containerarchitektur . . . . .	22
14	Button im Frontend . . . . .	23
15	Login- und Logout-Buttons . . . . .	23
16	Startlist Component . . . . .	28
17	Unterschiedliche Darstellung der Container . . . . .	29
18	Kalenderelement . . . . .	29
19	Unterschied der Urlaube . . . . .	30
20	Bearbeiten Button . . . . .	30
21	Betriebsurlaube bearbeiten . . . . .	30
22	Dashboard . . . . .	31
23	Door Logs Componente . . . . .	32

# **Tabellenverzeichnis**

1	Ein paar tabellarische Daten . . . . .	32
---	--	----

# **Quellcodeverzeichnis**

1	Subscriber für Raumtemperatur des Containers 147 . . . . .	8
2	Heizung des Containers 147 einschalten . . . . .	9
3	Keycloak Realm Konfiguration . . . . .	24
4	Keycloak Realm Konfiguration . . . . .	25
5	Keycloak Realm Konfiguration . . . . .	25
6	Keycloak Realm Client Konfiguration . . . . .	25
7	Nginx Konfiguration /auth . . . . .	26
8	Nginx Konfiguration / . . . . .	27
9	Nginx Konfiguration /api . . . . .	27
10	label im html . . . . .	30

# **Anhang**