ROOMsense

Hochschule Esslingen
University of Applied Sciences

SYSTECS

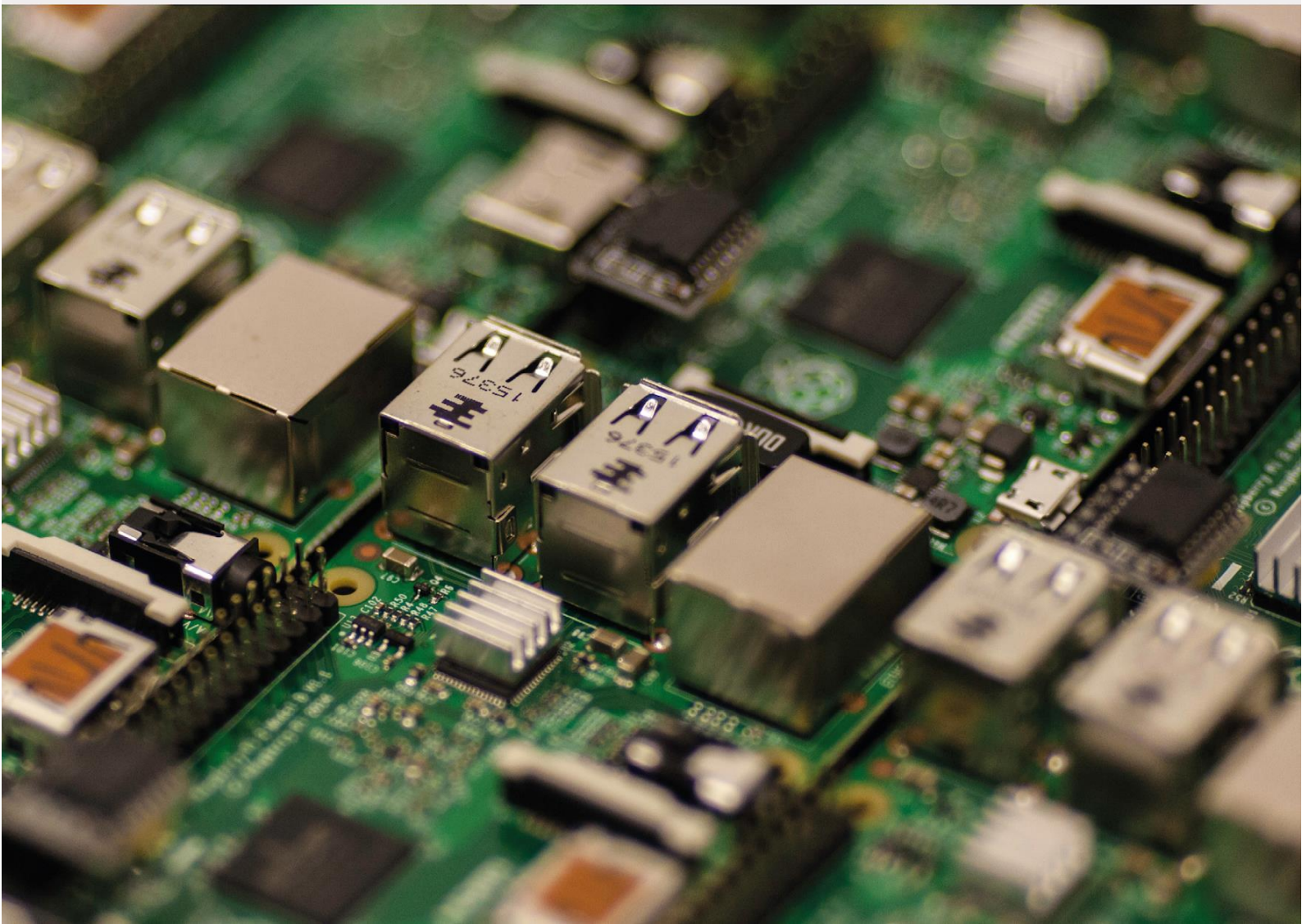
Projektarbeit SWB

Gebäude-Monitoring für die Firma SYSTECS

Dokumentation zur Umsetzung von ROOMsense

vorgelegt von:
Franziska Heil, Luca Heinrich, Sonja Tahiraj, Yannik Zbick

am 28.06.2019 an der Hochschule Esslingen



Inhaltsverzeichnis

1. Einleitung

Das Projekt ROOMsense ist Teil der Projektarbeit im Studiengang Softwaretechnik und Medieninformatik an der Hochschule Esslingen. Ziel des Projekts ist die Entwicklung einer WebApp, die mithilfe von Temperatur- und Luftfeuchtigkeitssensoren an einem Raspberry Pi Model B, ein Raummonitoring ermöglicht. Des Weiteren sollen jedem Raum Soll-Werte zugewiesen werden, um anzuzeigen, ob diese über- beziehungsweise unterschritten wurden. All diese Daten sollen in einer Datenbank zusammengefasst gespeichert werden, um auf verschiedenen Endgeräten, wie zum Beispiel Smartphones darauf zugreifen zu können.

Unterstützt wird das Projekt durch die Firma SYSTECS in Leinfelden-Echterdingen, die sowohl die Raspberrys, als auch ein monatliches Budget für Microsoft Azure bereitstellen.

2. Planungsphase

2.1. Zeitplan/ Meilensteine

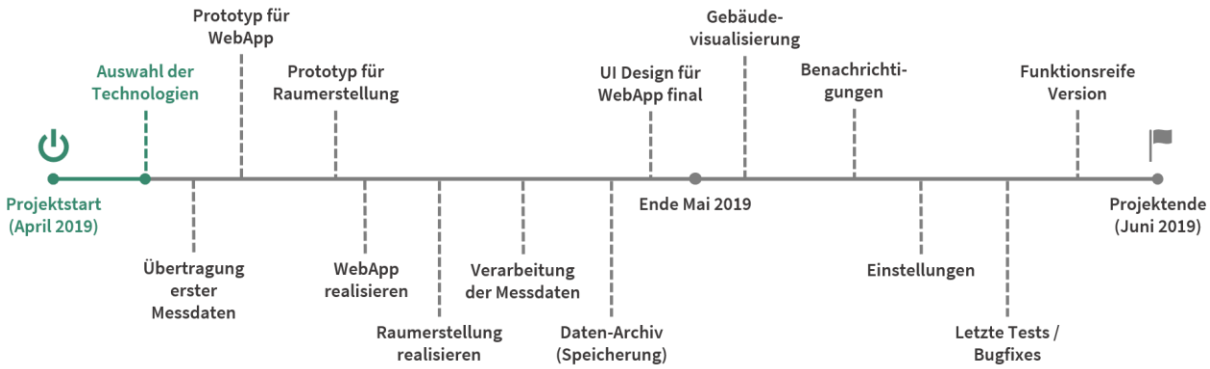


Abb. 1: Zeitplan mit Meilensteinen

Phase 1 (Vorbereitungsphase) bis Ende April 2019

- Auswahl der Technologien
- Übertragung erster Messdaten
- Prototyp für WebApp
- Prototyp für Raumerstellung

Phase 2 (Umsetzungsphase) bis Ende Mai 2019

- WebApp realisieren
- Raumerstellung realisieren
- Verarbeitung der Messdaten
- Daten-Archiv
- UI Design für WebApp final

Phase 3 (Test-/Fertigstellungsphase) bis Ende Juni 2019

- Gebäudevisualisierung
- Benachrichtigungen
- Einstellungen
- Letzte Tests und Bugfixes
- funktionsreife Version
- Dokumentation

2.2. Features/ Requirements

Must

1. Die Daten sollen in einem Datenarchiv gespeichert werden.
   Zur Erstellung von Statistiken und zur Nachverfolgung werden die Daten für einen bestimmten Zeitraum gespeichert.

2. Das System soll Meldungen und Warnungen ausgeben und darstellen.
   Meldungen und Warnungen entstehen wenn Grenzwerte überschritten oder unterschritten werden oder das Gerät einen Fehler aufweist. Bei Bedarf können Benachrichtigungen als Pop-Up oder E-Mail gesendet werden.

3. Die WebApp soll eine Funktion zur Erstellung einer Gebäudedarstellung besitzen.
   Um Räume und Stockwerke darzustellen in denen gemessen wird soll das System eine Möglichkeit bieten das Gebäude zu rekonstruieren.

4. Das System soll über Einstellungen verfügen.
   Es soll möglich sein Grenzwerte für die gemessenen Werte pro Raum festzulegen um dem Nutzer eine genaue Überwachung zu ermöglichen.

Should

1. Zur besseren Übersicht und als Rückblick sollen Statistiken eingeblendet werden.
   Statistiken sollen immer in Form eines Liniendiagramms für einen bestimmten Zeitraum angezeigt werden. Über einen Button kann man zwischen Temperatur und Luftfeuchte wechseln.

2. Der Nutzer soll die Möglichkeit haben Sensoren/Räumen und Stockwerken Namen zu geben und zu bearbeiten.
   Für eine bessere Übersicht und eine mögliche Suche soll der Nutzer bei der Raumerstellung dazu in der Lage sein z.B. Räumen Namen zu geben. Diese Namen soll er nachträglich ändern können.

3. Das System soll eine Suchfunktion besitzen.
   Diese Funktion geht mit der Namensgebung einher. Die Suchfunktion soll dem Nutzer eine schnellere Möglichkeit bieten Räume oder Sensoren zu finden.

Could

1. Bei Inputfeldern soll der Nutzer Unterstützung durch Hilfetexte bekommen.
   Wenn der Nutzer einen falschen Begriff eingibt soll das System dies erkennen und ihn darauf Hinweisen bzw. angeben, was eigentlich zu tun ist.

Would

1. Der Nutzer kann sich ein Profil anlegen.
   Mithilfe eines Profils kann der Nutzer verschiedene Rechte erhalten, um z.B. nur auf bestimmte Daten zugreifen zu können.

2. Das System erhält zusätzliche Sensoren.
   Das System kann durch weitere Sensoren (z.B. Rauch-/Gassensor oder Infrarot) erweitert werden.

3. Der Raspberry kann um ein Display erweitert werden.
   Dies ermöglicht dem Nutzer einen einfacheren Überblick. Er hat die Möglichkeit die Werte direkt abzulesen ohne auf die WebApp zugreifen zu müssen.

2.3. Use Case

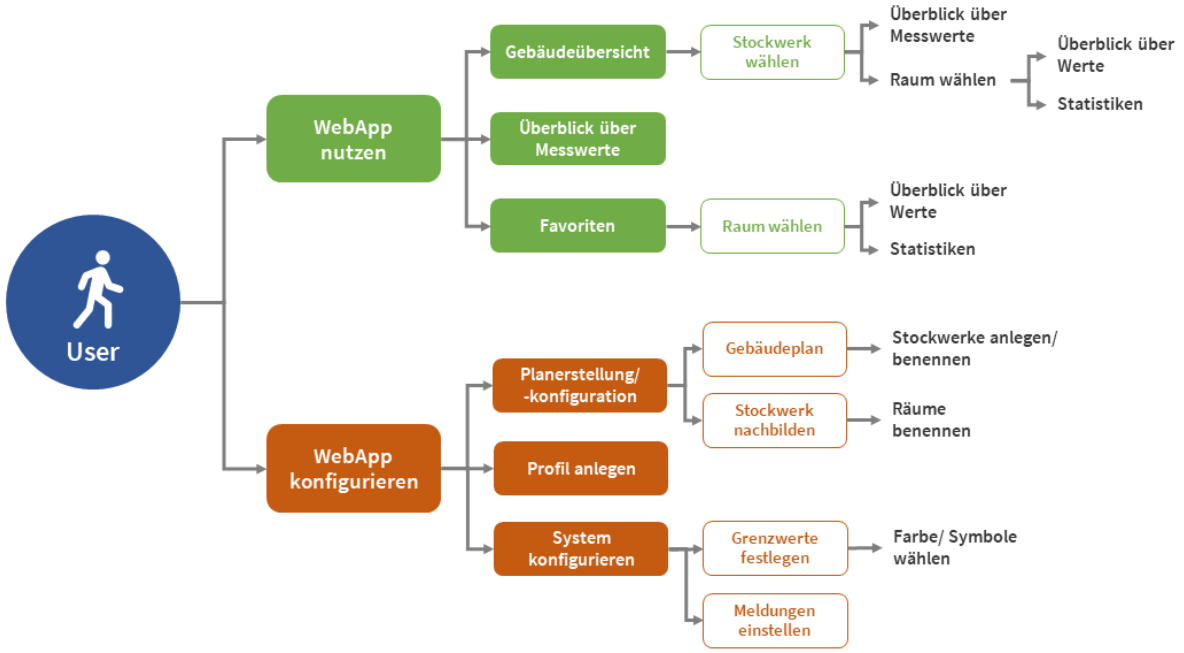


Abb. 2: UseCase-Diagramm

Der UseCase ist in zwei Bereiche unterteilt: Nutzung und Konfiguration der WebApp.

Nutzung

Bei der Nutzung soll man die Möglichkeit haben über eine Gebäudeübersicht ein Stockwerk zu wählen und in diesem über eine Schnellübersicht Meldungen und Messwerte in den Räumen angezeigt bekommen. Bei der Auswahl eines Raumes gelangt der Nutzer in eine Detailansicht. Hier werden die gemessenen Werte angezeigt und man hat die Möglichkeit über eine Statistik ältere Messwerte nachzuverfolgen.
Des weiteren soll der Nutzer jederzeit einen schnellen Überblick über die Messwerte in den einzelnen Räumen erhalten.
Über eine Favoritenliste soll er die Möglichkeit haben Räume zu favorisieren, um diese schneller zu finden und so eine schnellere Arbeitsweise zu ermöglichen.

Konfiguration

Der Nutzer sollte auch dazu in der Lage sein das System anzupassen. Die wichtigste Funktion ist dabei die Planerstellung bzw. -konfiguration. Hierbei soll er in der Lage sein sowohl das Gebäude nachzubilden, als auch die einzelnen Stockwerke anzulegen und zu benennen. In diesen Stockwerken gibt es dann die Möglichkeit Räume zu benennen und ihnen ihre jeweiligen Sensoren zuzuordnen.
Ein weiteres Feature kann das Anlegen eines eigenen Profils sein, dies ist aber vorerst optional.

Der Nutzer soll die Möglichkeit haben Grenzwerte für Temperatur und Luftfeuchtigkeit festzulegen und einstellen können, ob er Benachrichtigungen erhalten will, falls Grenzwerte unter- oder überschritten werden.

2.4. Verantwortlichkeiten/ Projektmanagement

Verantwortlichkeiten	Zugeteilte Person/-en
Projektleiter	Franziska Heil
Requirements Engineer/ Manager	alle
Grafik-/UX-Designer	Franziska Heil, Sonja Tahiraj
Webentwicklung	alle
Qualitätsmanager	alle
Softwarearchitekt	Luca Heinrich, Yannik Zbick
Softwareentwickler	alle
Datenbank-Administrator	Luca Heinrich, Yannik Zbick

Das Projektmanagement erfolgt über eine agile Arbeitsweise, die zusätzlich auf einer Verteilung von Rollen basiert. So sind vor Allem in der Planungsphase Rollen fest zugeteilt, um eine effektive und parallele Arbeitsweise zu ermöglichen. Dafür wurde die Arbeit an der Datenbank, der Softwarearchitektur und der Entwicklung und Evaluierung und Erstellung eines Designs in zwei Gruppen aufgeteilt.
Im Laufe des Projekts übernehmen die Teammitglieder alle Rollen, um einerseits schnell zu arbeiten, aber auch einen Einblick in jedes der Gebiete zu erhalten.
Über allem steht ein Projektleiter, der den Zeitplan überwacht und die Organisation übernimmt.

Kanban Board

Zur agilen Arbeitsweise gehört das Arbeiten mit einem Kanban Board, in dem Tickets mit Aufgaben angelegt und bearbeitet werden. Die Tickets werden dafür in verschiedene Prozessgruppen eingeteilt (To Do, In Progress, [Problems (Testphase)], Review, Done). Nach Abschluss eine Projektphase werden die zugehörigen und bearbeiteten Tickets in eigenen Gruppen abgelegt.

Die Wahl für das Projektmanagement-Tool fiel dabei auf Trello von Atlassian. Es ermöglicht eine Priorisierung der Tickets, das Verknüpfen mit Dokumenten und eine Zuteilung der Aufgaben.

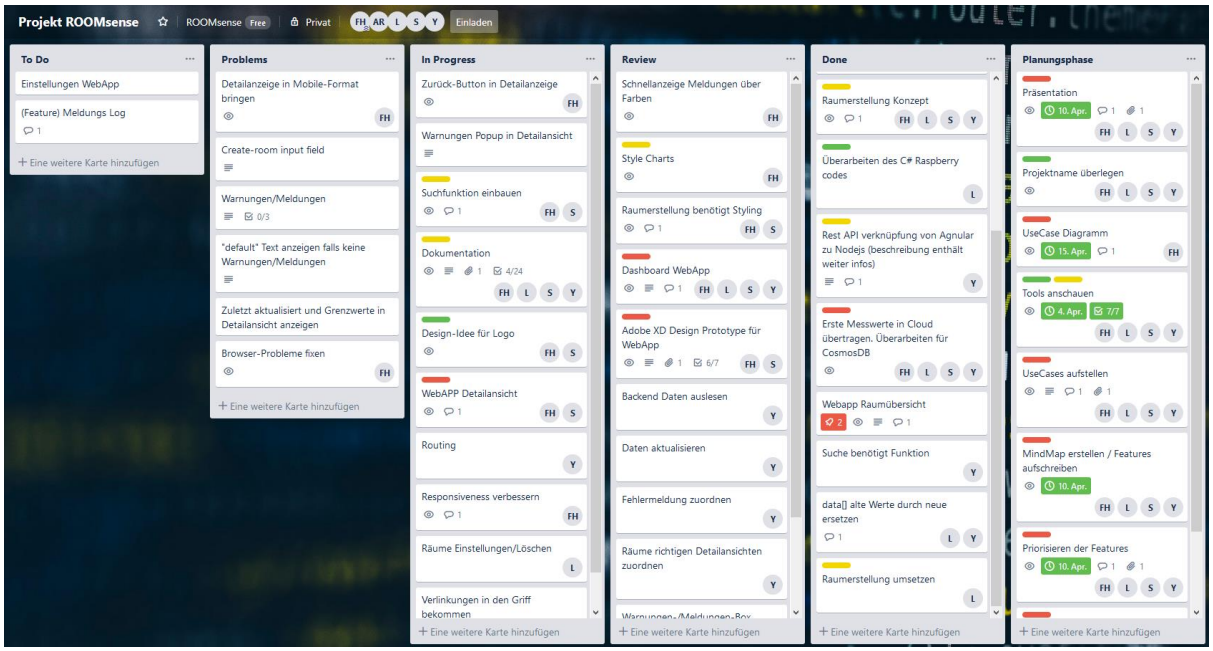


Abb. 3: Kanban Board bei Trello

2.5. Versionsverwaltung (GitHub)

In einem größeren Projekt mit verschiedenen Programmierern und gleichzeitigem Arbeiten am Programmcode und Design ist eine Versionsverwaltung / Dateiverwaltung eine wichtige Komponente um den Überblick zu behalten. So ist es mit Github möglich, dass z.B. zwei Programmierer an demselben Projekt arbeiten und die Änderungen individuell voneinander auf dem Online -Speicher zusammenzufügen ohne Dateien oder Programmzeilen kopieren zu müssen. Wie in Abb. 4 ersichtlich, wurde teilweise zu dritt am Projekt gearbeitet und dadurch lokale Abspaltungen (Blau & Grün) erzeugt, die dann später wieder im Hauptprojekt (Schwarz) zusammengefügt wurden.
Die Zahlen bedeutet hierbei den Tag im Monat an dem die Aktion getätigt wurde.

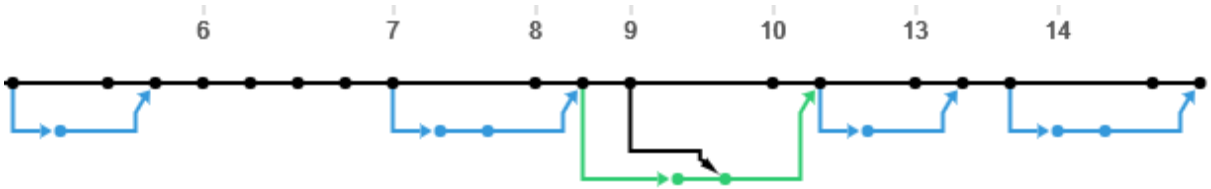


Abb. 4: Branches

Dazu ist für jede Änderung am Programmcode eine detaillierte Übersicht verfügbar, welche die bearbeiteten Dateien anzeigt und Auskunft darüber gibt, was davon gelöscht oder neu eingefügt wurde.

```
22  27      messageControl = new FormControl();
23      -   lowerControl = new FormControl();
24      -   upperControl = new FormControl();
    28  +   lowertempControl = new FormControl();
    29  +   uppertempControl = new FormControl();
    30  +   lowerhumiControl = new FormControl();
    31  +   upperhumiControl = new FormControl();
25  32      createRoomForm: FormGroup;
26      -   valid = false;
27  33
28      -   data: Roomdata [] = [];
29      -   dataToDisplay: Roomdata [];
30      -   isLoading = false;
    34  +
    35  +   allDeviceData: Devices [] = [];
    36  +   roomDeviceData: Devices [] = [];
    37  +   Once: boolean = true;
31  38      public raspberryarr: string[] = []; // holds the options for the Dropdown menu
32  39
```

Abb. 5: Anzeige von aktualisierten Daten

Die Versionsverwaltung kann über folgenden Link erreicht werden: “https://github.com/7Surfer/ROOMsense”.

3. Design/ Prototyp

3.1. Idee/ Konzept

Das Konzept hinter dem UX-Design der WebApp basiert zum Teil auf der Homepage von SYSTECS. Durch generische Formen, einen einfachen Aufbau und das Flat-Design, soll der Nutzer in der Lage sein schnell zu handeln und einen guten Überblick zu erhalten.

Ideen für das Design und den Aufbau bot z.B. das folgende Beispiel:

Abb. 6: Beispiel für den Aufbau der WebApp

An dem oben gezeigten Beispiel lässt sich sehr gut eine Aufteilung ausmachen, die für unsere Idee sehr zutreffend war. Das bedeutet, dass die WebApp ein Hauptfenster (links) hat in welchem man zu den verschiedenen Elementen wechselt und

3.2. Farbkonzept

Da die WebApp an die Homepage der Firma SYSTECS angelegt ist, wurde das Farbschema aus deren Corporate Design verwendet.

Abb. 7: Colorcodes

Dabei bilden die Blautöne meist die Basis für Texte und Dinge, die im Vordergrund stehen und die Grautöne sind meist hintergründige Farben.
Sonstige Farben wzB. der Grün- bzw. Rotton sind Teil der Temperatur- und Luftfeuchtigkeitsanzeige (Grün für alle Werte sind im Bereich, Rot über Bereich und Blau unter Bereich).

3.3. Typographie und Icons

Typographie

Auch bei der Wahl der Typographie wurde auf das Corporate Design von SYSTECS geachtet. Open Sans von Google bildet dabei die Haupt-Typographie und für die Detailanzeige wurde auf die Schrift Oswald zurückgegriffen. Beide Schriften sind modern, serifenlos und daher gut lesbar.

Open Sans

The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog

Oswald

The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog

Icons

Bei der Verwendung der Icons wurde auf Icons aus dem Material Design von Google zurückgegriffen. Diese sind Open Source und passen optisch zur gewählten Schriftart Open Sans.

3.4. Formen

Hierbei wurde vor Allem darauf geachtet, dass die verwendeten Formen modern und generisch sind und wenn nötig Aufmerksamkeit erregen.

Die Hauptfelder besitzen abgerundete Ecken und werden durch Schatten im Hintergrund hervorgehoben. Auch Elemente innerhalb dieser Hauptfelder sind abgerundet. Das führt dazu, dass alles einen einheitlichen Look erhält.

In der Detailansicht sind die gemessenen Werte für Temperatur und relative Luftfeuchtigkeit in Kreisen dargestellt, die von farbigen Feldern umrandet werden. Diese erregen Aufmerksamkeit und ermöglichen dem Nutzer eine schnelle Ansicht, ob der Wert im Bereich liegt oder ein Grenzwert unter- bzw. überschritten wurde.

Buttons werden durch Icons markiert oder ebenso durch abgerundete Formen dargestellt.

Abb. 8: Buttons

3.5. Aufbau

Desktopversion

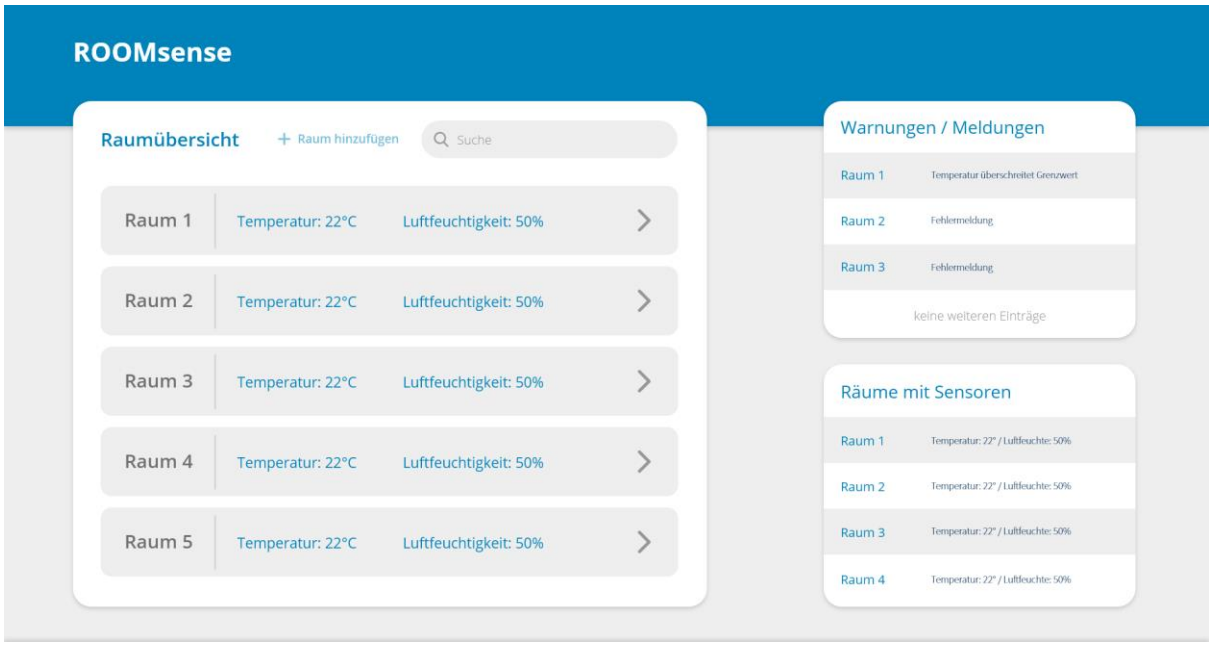


Abb. 9: Startseite

Der Grundaufbau der WebApp besteht aus einem Hauptfenster links und zwei kleineren Fenster rechts, die Warnungen und Meldungen bzw. in einer Schnellansicht alle Räume und deren Werte anzeigen. Die kleineren Felder rechts sind statisch, d.h. nur im Hauptfenster werden andere Inhalte eingeblendet.

Wenn man die WebApp öffnet gelangt man zu der oben gezeigten Ansicht und erhält einen Überblick über alle Räume, die bereits angelegt wurden und mit Sensoren verknüpft sind angezeigt. Mithilfe des Suchfeldes rechts oben kann man innerhalb der Raumliste nach einem Raum suchen, um einen schnelleren Zugang zu ermöglichen.

Über den “Raum hinzufügen”-Button gelangt man zur Raumerstellung. Diese war vorerst als Drag&Drop-Funktion geplant, doch durch das Flat-Design und Vereinfachung für den Nutzer wurde das ganze durch Inputfelder gelöst.

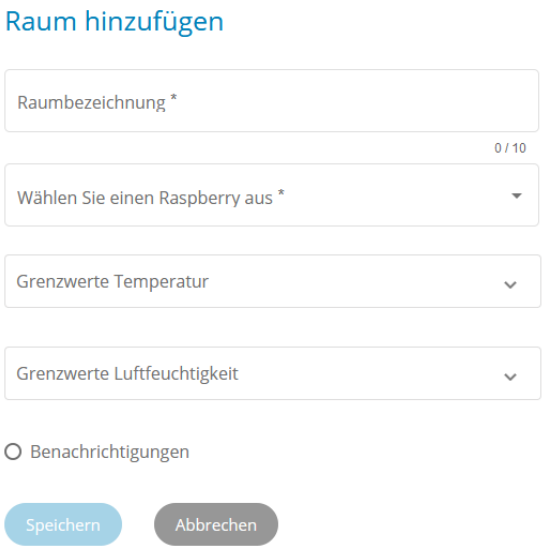


Abb. 10: Raum hinzufügen

Über Drop-Down-Menüs kann man die Grenzwerte für die einzelnen Messwerte eintragen. Ohne das Ausfüllen der Pflichtfelder (markiert mit *) ist ein Speichern nicht möglich. Durch das Auswählen des “Abbrechen”-Buttons gelangt man zurück zur Raumübersicht.

Wählt man einen der “Raum”-Buttons bzw. das Chevron in den Raumdarstellungen aus, gelangt man zur Detailansicht der Räume.

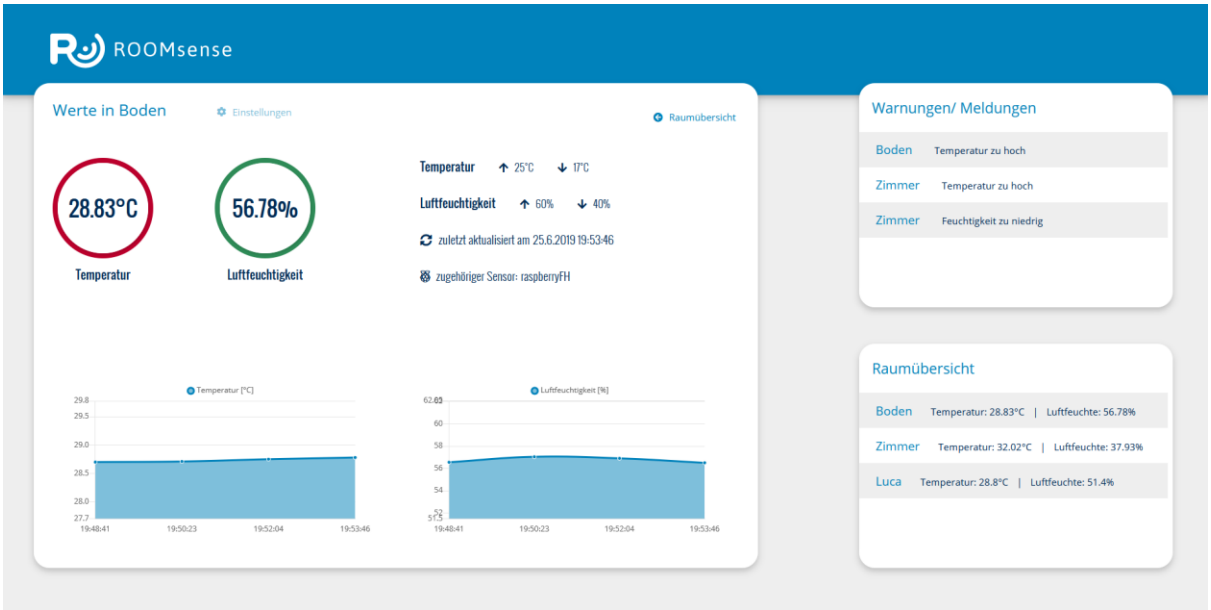


Abb. 11: Detailansicht

Hier erhält man neben den Messwerten zusätzlich die Grenzwerte und wann die Werte zuletzt aktualisiert wurden. Auch der Name des zugehörigen Sensors wird angezeigt.

Die Kreise mit den Messwerten sind farbig umrandet. Jede Farbe hat dabei eine Bedeutung: Grün→ Wert liegt im Bereich, Rot→ Wert liegt oberhalb des oberen Grenzwerts, Blau→ Wert liegt unterhalb des unteren Messwerts.

Im unteren Teil des Hauptfensters werden Liniendiagramme dargestellt. Diese zeigen die fünf zuletzt gemessenen Werte.

Über den “Raumübersicht”-Button bzw. über einen Klick auf das Logo gelangt man zurück zur Raumübersicht.

Über den “Einstellungen”-Button gelangt man zu einer weiteren Ansicht, die der der Raumerstellung entspricht. Hier hat der Nutzer die Möglichkeit alle Werte, die er bei der Erstellung der Räume angegeben hat zu ändern und anschließend zu aktualisieren. Es besteht zudem die Möglichkeit den Raum zu löschen.

Das Ziel des Aufbaus ist, dem Nutzer lange Wege zu ersparen. Mit einem Mausklick erreicht er bereits die Detailansicht und durch die kleinen Felder rechts, hat er immer die Möglichkeit zwischen Räumen zu wechseln bzw. eine schnelle Anzeige, ob es in einem Raum zu warm oder zu kalt ist.

Mobile-Version
In der Mobile-Version stehen nur noch die beiden kleinen Felder zur Verfügung. Sie enthalten die wesentlichen Dinge der WebApp und ermöglichen eine Schnellansicht.

3.6. Versionen/ Verworfene Ideen

Wie es bei jeder Entwicklung üblich ist wurde auch der Prototyp oft verändert. Das lag zum einen daran, dass Umsetzungen nicht möglich waren oder eine einfachere Lösung im Laufe der Umsetzung gefunden wurde.

Der ursprüngliche Plan enthielt eine Gebäudedarstellung inklusive einer Abbildung der einzelnen Stockwerke. Diese wurde im Laufe der Entwicklung entfernt, da eine Ansicht der Räume schnellere Wege ermöglicht.

Auch die Breadcrumbs und Menüpunkte wurden entfernt, da es immer einen Button gibt um zur vorherigen Ansicht zurückzukehren und es durch die kurzen Wege leicht verständlich ist, wo man sich gerade befindet.

Da die Suche auf Räume begrenzt wurde, befindet sich das Suchfeld nun in der Raumübersicht.

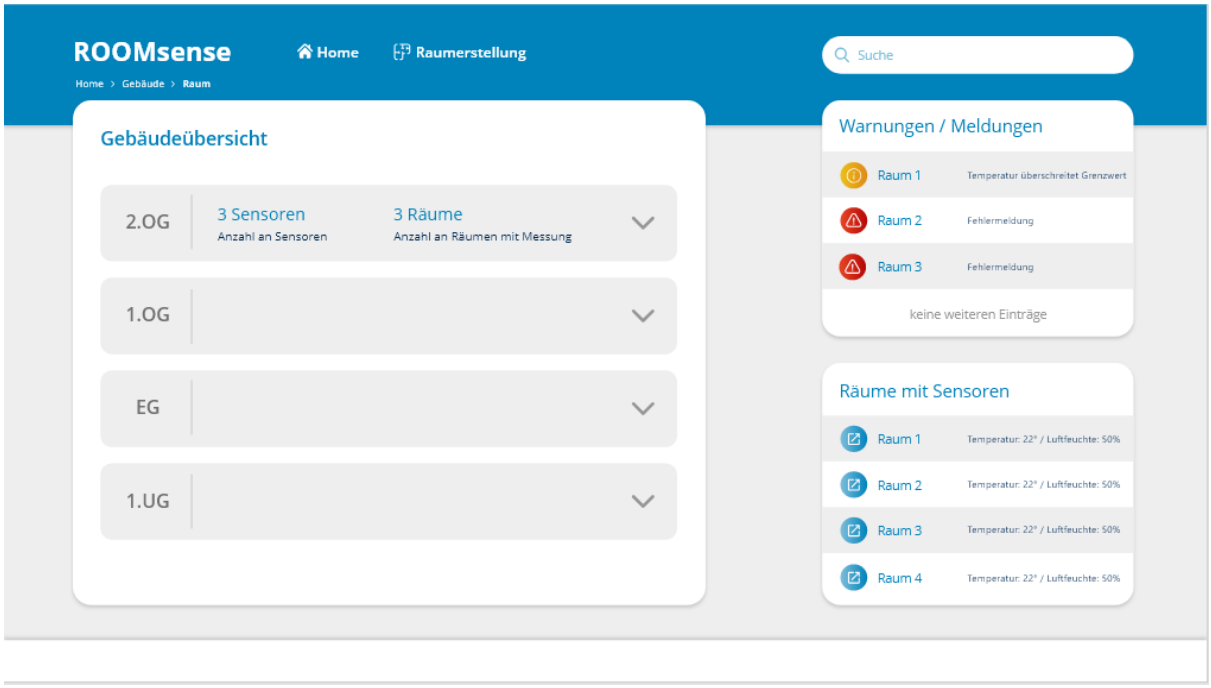


Abb. 12: Startseite Prototyp

Auch die Einstellungen waren komplizierter angelegt. Nun gleichen sie der Raumerstellung und ermöglichen so die einzelnen Werte anzupassen.

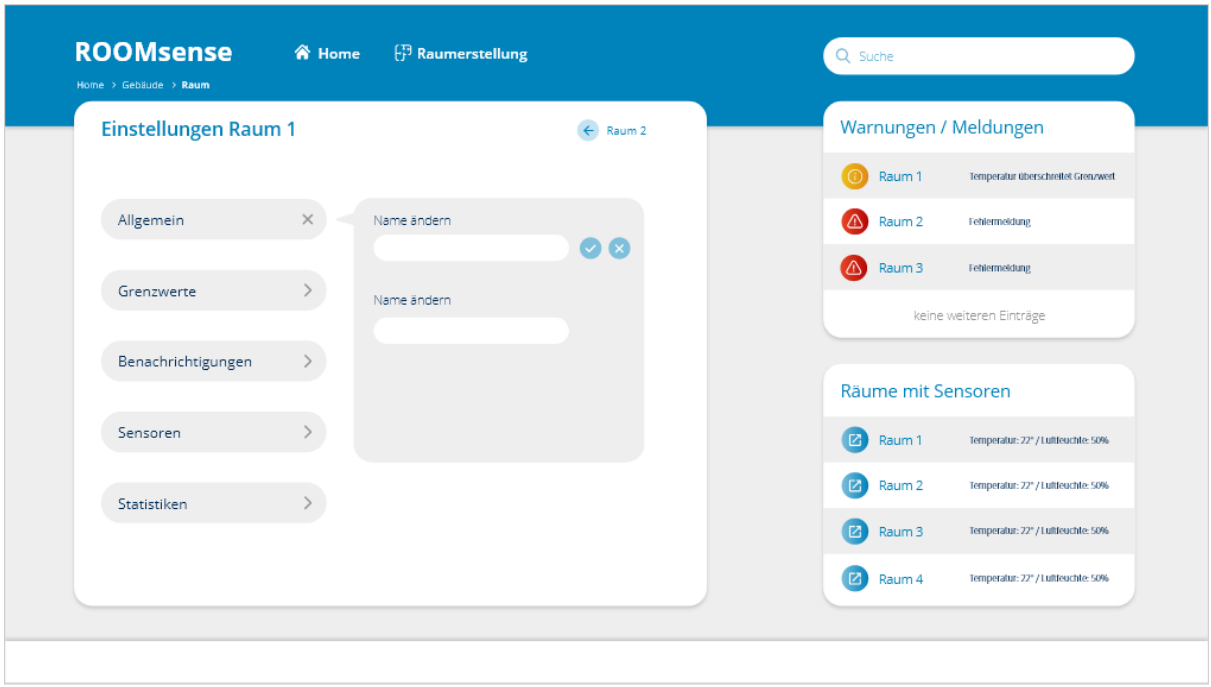


Abb. 13: Einstellungen Prototyp

Ursprünglich war es geplant ein Datum und die aktuelle Uhrzeit anzugeben. Allerdings wurde dieses Feld durch eine Anzeige der Grenzwerte und der letzten Aktualisierung der Werte ersetzt.

Zudem werden statt einem Fehlertext (geplant als Alternative für die Statistiken) eben diese Statistiken eingebaut. Die Icons in den Anzeigen wurden durch die Werte ersetzt.

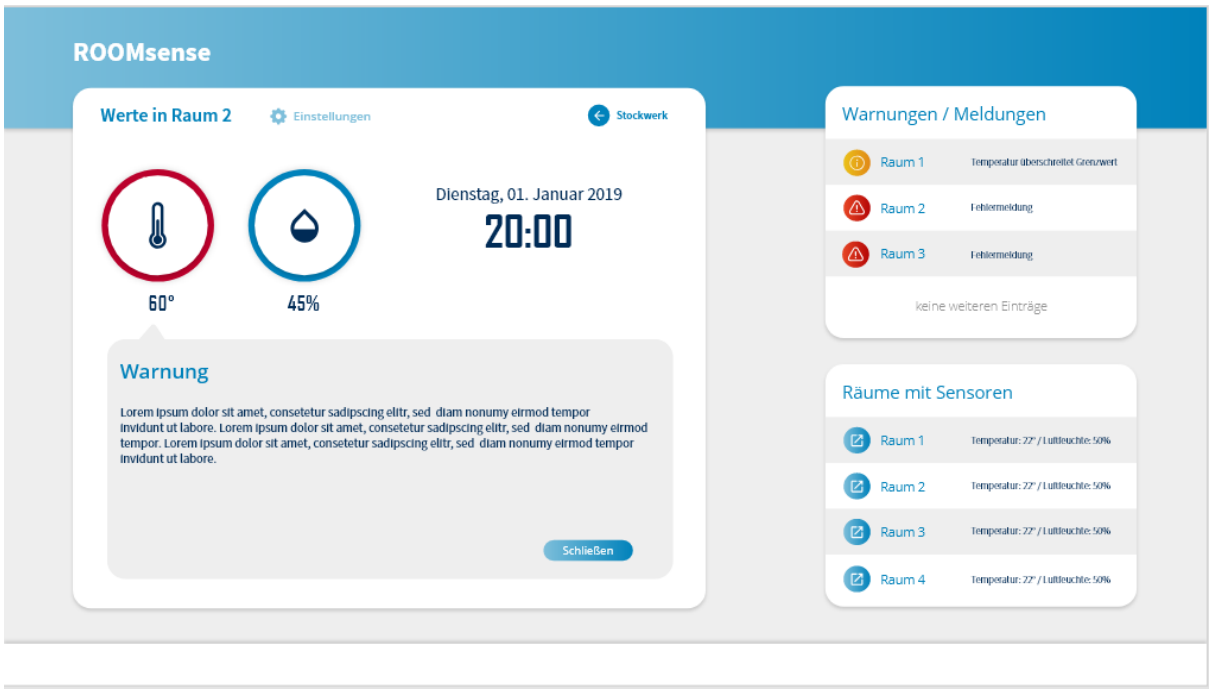


Abb. 14: Detailansicht Prototyp

3.7. Logo

ROOMsense

Das Logo ist in den Farben von SYSTECS gehalten. In der WebApp wird es allerdings aufgrund des Hintergrundes im Header weiß dargestellt.
Die Verlängerung am Buchstaben R soll ein WiFi-Symbol darstellen. Dieses Symbol steht für die Übertragung der Werte vom Raspberry in die Datenbank und von dort aus zur WebApp.

Da auch die gewählte Schrift für die WebApp (Open Sans) serifenlos ist und dennoch einen leicht geschwungenen Ansatz hat wurde dieser hier im Logo in Form von der Schrift Asap umgesetzt. Zudem ist eine serifenlose Schrift moderner und schafft somit eine Verbindung zum technischen Ansatz der Firma.

4. Raspberry Pi

Der Raspberry Pi dient als Hardware Schnittstelle für die Messdaten aus dem Temperatur- und Feuchtigkeitssensor Si7021 und der Datenbank. Er wird genutzt, um die Daten zu empfangen, zu verarbeiten und an die Azure Cosmos DB zu senden. Für diese Schritte wurde der Programmcode in drei wesentliche Teile aufgeteilt. Das Lesen der Daten, die Verarbeitung und schließlich das Senden an die Cosmos DB. Für diese Aufgaben wurde eine C# Windows Vordergrund App genutzt. Dabei handelt es sich um eine Anwendung, die eine grafische Oberfläche besitzt, die wir nutzen um die gelesenen Daten anzuzeigen, sodass der Nutzer ein visuelles Feedback bekommt. Dort wird Ihm die Möglichkeit geboten die aktuell gelesenen Daten zu sehen und die App durch einen Knopfdruck zu beenden.

4.1. Si7021 Sensor

Für das Lesen der Temperatur und Luftfeuchtigkeit wurde der Si7021 Sensor genutzt. Dieser benötigt eine 3,3V Stromversorgung sowie ein SDA und SCL Ein/Ausgang für den "I^2C Bus" der zum Datenaustausch genutzt wird.
Der Sensor hat im Code eine eigene Klasse bekommen, sodass man im Hauptprogramm nur die öffentlichen Funktionen "Setup_device", "get_temperature" und "get_humidity" aufrufen muss. Zum Auslesen der Temperatur/Luftfeuchtigkeit wird der entsprechende Hexcode[1] über den "I^2C Bus" gesendet um dadurch die entsprechenden Werte zu bekommen. Da es anfangs zu Problemen mit der richtigen Verkabelung zwischen Sensor und Raspberry Pi gab, wurde noch ein try-catch Block beim Senden der Instruktionen an den Sensor eingefügt, der im Falle eines Fehlers eine entsprechende Fehlermeldung in die Konsole ausgibt. Die gelesenen Werte müssen dann noch in °C/%RH umgewandelt werden[2]. Zuletzt wird der Wert auf zwei Nachkommastellen gerundet und zur Übergabe an die Cosmos DB von der Klasse zurückgegeben.

4.2. Verbindung zur Azure Cosmos DB

Über Klassen und Funktionen im Namensraum Microsoft.Azure.Devices.Client erfolgt das Senden der gemessenen Daten an den Azure IoT-Hub. Einer Instanz der Klasse DeviceClient wird neben der IoT-Hub-URI, der DeviceId und des Device-Keys auch noch das Transportprotokoll HTTP übergeben. Mit Hilfe der Methode
SendEventAsync(message)
werden die zuvor nach UTF-8 kodierten Messdaten an den IoT-Hub gesendet.

[1] https://www.silabs.com/documents/public/data-sheets/Si7021-A20.pdf Seite 18
[2] https://www.silabs.com/documents/public/data-sheets/Si7021-A20.pdf Seite 21/22

Jedes Mal, wenn im Iot-Hub neue Daten ankommen wird eine erstellte Azure Function ausgelöst, welche als Ausgabebindung die erstellte Azure Cosmos DB (SQL-API) nutzt. Damit werden die eingehenden Daten in der Datenbank gespeichert. Azure fügt zusätzlich weitere Informationen wie zum Beispiel eine ID oder den Zeitstempel in Sekunden (_ts) eigenständig hinzu, was die spätere Verarbeitung der Daten vereinfacht.

5. Allgemeine Architektur

Zur Erstellung einer Webanwendung zur Anzeige von Temperatur- und Luftfeuchtigkeitsdaten bedarf es einiger Entscheidungen bezüglich der Architektur beziehungsweise der Verwendung verschiedener Technologien.
Entschieden haben wir uns für Angular als Frontend-Framework und NodeJS / Express als Backend-Runtime-Environment.

5.1. NodeJS

NodeJS ist ein JavaScript Framework / Laufzeitumgebung basierend auf Googles V8-Engine, welche es ermöglicht, JavaScript auf einem Server auszuführen.
Besonders Node's Non-Blocking-I/O bringt enorme Vorteile mit sich, wenn es um die Abarbeitung von asynchronen Tasks, wie zum Beispiel Datenbankabfragen geht.
Das Ergebnis einer Datenbankabfrage steht selbstverständlich nicht direkt nach Ausführung der abfragenden Funktion zur Verfügung, was unter Verwendung von beispielsweise PHP unter Umständen dazu geführt hätte, dass erst auf das Ergebnis der ersten Abfrage hätte gewartet werden müssen, bevor eine zweite Abfrage ausgeführt werden kann.
Um dies zu vermeiden, verwendet Node nicht für jede Anfrage einen eigenen Thread, sondern führt alle Anfragen nacheinander auf einem Thread aus. Asynchrone Operationen wie Datenbankabfragen verwenden Callbacks um das Ergebnis dann zurück zu liefern, wenn die Daten zur Verfügung stehen.
Das führt dazu, dass Ladezeiten enorm reduziert werden, was der User-Experience sehr zugute kommt. Dadurch ist Node sehr skalierbar, was die Handhabung von mehreren Anfragen gleichzeitig betrifft.

In Kombination mit dem Node.js-Framework Express, dessen HTTP-Modul und Middleware-Funktionen lässt sich eine leistungsstarke backend API erstellen.

5.2. Angular

Das JavaScript Framework ist prädestiniert für die Entwicklung von Single-Page-Applications, bei denen die Seite nur einmal geladen wird und weitere Komponenten dynamisch nachgeladen werden.

Als Architektur wird dazu eine Hierarchie von Komponenten verwendet, die einen Teil des Views darstellen und sogar weitere Komponenten enthalten können.
Das Verwenden von sogenannten Direktiven macht besonders den HTML-Code übersichtlicher und leicht wiederverwendbar und verhindert die Verschachtelung von unzähligen HTML-Elementen wie , <ul> oder <li>. Durch Bindings werden Darstellung und Logik miteinander verbunden und es ist möglich, dynamisch Daten auf dem HTML-Template anzuzeigen.
Das Angular eigene HTTP-Client-Modul ermöglicht die Kommunikation mit einer entsprechenden Web-API. Unter Verwendung der rxJS Bibliothek könne Observables anstatt Promises verwendet werden, um asynchrone Daten zu verarbeiten und den Empfänger über neue Daten zu informieren.
Des Weiteren bietet Angular durch die Erstellung von Services die Möglichkeit an, den Zugriff auf Daten an einem Ort zu bündeln und von dort an alle Komponenten weiter zu reichen.

6. Sensor Data

Sensor Data Service

Komponenten in Angular sollten es vermeiden, einzeln für sich Daten abzufragen. Rufen mehrere Komponenten die gleichen Daten ab, kann es zu einer Verfälschung dieser kommen. Nach dem Design Pattern der Dependency Injection wird der Zugriff auf Daten einem Service zugeteilt. Die Komponenten können sich so auf die Darstellung der Information konzentrieren und Daten über den Service abfragen.
Der Service wird im Angular-Komponent im constructor injiziert, um überflüssige Abhängigkeiten zu vermeiden:

```
constructor(private sensorDataService: SensorDataService) { }
```

Der Service implementiert jegliche Http-Requests gegen den Server und verarbeitet jegliche Antwort des Servers. Des Weiteren stellt der Service ein Subject bereit, welches in Angular dafür verwendet werden kann, alle Komponenten die vom Service entsprechende Daten benötigen mit den neuesten Daten zu versorgen, damit diese angezeigt werden können. Dazu fragen die Komponenten diesen beim Service an, erhalten das Subject als Observable und können dieses abonnieren, um über eine Änderung der Daten im Datenstrom informiert zu werden und die neuesten Daten zu erhalten.

Service ruft auf, wenn Server mit neuen Daten antwortet:

```
this.fullDataUpdated.next([...this.fullData]);
```

[...this.fullData] schreibt Kopie der Originalen Daten in Subject damit die Empfänger nicht die Originaldaten verändern können.

Komponent abonniert Subject bei Initialisierung und speichert Abonnement in lokaler Variable um Subscription bei Zerstörung wieder aufzulösen → verhindert Memory-Leaks.

```
this.fullDataSubscription =
this.sensorDataService.getFullDataUpdateListener().subscribe((upda
tedFullData: any[]) => {
     this.fullData = updatedFullData;
   });
```

Daten alle 5 Sekunden aktualisieren

Neue Daten werden in Zeitabständen von 5 Sekunden abgerufen. Bei Initialisierung wird in der Sensor Box ein Intervall gestartet, welches regelmäßig neue Daten über den Service abfragt. Die Sensor-Box-Komponente wird nie zerstört, wodurch das Intervall nie gestoppt und neu gestartet werden muss. Sicherheitshalber wird die ID des Timers, die setInterval() zurück gibt dennoch in einer lokalen Variable gespeichert, um das Intervall im Fehlerfall zu stoppen.
Liegt kein Fehler vor, werden die Daten in der Komponente alle 5 Sekunden aktualisiert und über das Subject wie oben erwähnt an alle weiteren Komponenten weitergegeben, die neue Daten benötigen.
Im ungünstigsten Fall konnte es vorkommen, dass erst kurz vor dem Aufruf einer neuen Komponente durch den Timer die Daten aktualisiert wurden. Um die hieraus resultierende Wartezeit bis zum Anzeigen der Daten von über dann über vier Sekunden (Zeit bis zur nächsten Aktualisierung) zu verhindern,
ruft jede Komponente bei Initialisierung einmal die neuesten Daten für sich ab, zeigt diese an und richtet sich dann an den Aktualisierungen des Intervalls. So wird überflüssige Wartezeit verhindert, was unter anderem auch die User Experience verbessert.
Des Weiteren wird dadurch auch die Anzahl an Datenbankabfragen stark reduziert.

Bei Ansicht der Main-Page, also des Floorplans wird als nur alle fünf Sekunden eine Datenbankabfrage ausgeführt, dies reicht jedoch aus, um auch die Daten in der Warnungen-Box und der Sensor-Box zu aktualisieren.

Sensordaten und Rauminformationen kombinieren

Um möglichst wenig verschiedene Datensätze über die gesamte Anwendung verteilen zu müssen, erhält jede Komponente die Daten benötigt Zugriff auf einen Datensatz, der die Daten aus zwei verschiedenen Collections innerhalb der Datenbank vereint. Dazu müssen die gemessenen Sensordaten inklusive Timestamp mit den gespeicherten Raumdaten inklusive Höchsttemperatur, Mindesttemperatur, etc. kombiniert werden, um einen vollen Datensatz zu erhalten, aus dem jede Komponente nur die Informationen anzeigen kann, die sie soll.

Dazu werden aus der Collection mit den gemessenen Sensordaten alle Daten der letzten drei Tage (aus Performance- und Kostengründen) abgerufen.

Aus der Collection mit den gespeicherten Raumdaten werden alle Einträge abgerufen. Da diese gelöscht und überschrieben werden können, sollte sich die Anzahl an Einträgen hier in Grenzen halten.

Über den Datensatz, der die Sensordaten der letzten drei tage enthält wird iteriert und alle Sensornamen in ein dafür vorgesehenes Array geschrieben. JavaScript verfügt seit ES6 über ein Set-Objekt, mit dessen Hilfe eindeutige Werte in der Reihenfolge ihres Hinzufügens in eine Sammlung geschrieben werden können. Entsprechend der verwendeten Query zur Datenbankabfrage sind die Daten nach absteigendem Timestamp sortiert:

```
`SELECT c.deviceId, c.temperature, c.humidity , c._ts FROM c c
WHERE c._ts >  ${timestamp} ORDER BY c._ts DESC`
```

`${timestamp}` definiert das aktuelle Datum in Sekunden abzüglich von 260 000 Sekunden, was einer Zeitspanne von drei Tagen entspricht. Dieser Wert wurde auch aus Test- und Demozwecken so gewählt, damit nicht dauerhaft Daten gesendet werden müssen, um diese anzeigen zu können.

Doppelte deviceId's werden entfernt:

```
let distinctNameArray = [...new Set(nameArray)];
```

Übrig bleibt ein Array mit den Namen der Sensoren, die in den letzten drei Tagen Daten an die Datenbank gesendet haben.

Die eindeutigen Namen können dann dazu verwendet werden, um mit Hilfe des Indexes des ersten Auftretens im kompletten Datensatz, jeweils den neuesten Eintrag des jeweiligen Sensors zu extrahieren.

```
for (let i = 0; i < distLen; i++) {
distinctIndexes.push(nameArray.indexOf(distinctNameArray[i]));
 }

for (let i = 0; i < distInLen; i++) {
   newDataArray.push(a[distinctIndexes[i]]);
 }
```

Ergebnis: Datensatz mit den neuesten Temperaturmessungen der letzten drei Tage.

Als nächsten Schritt müssen die vorhandenen Sensordaten mit den Raumdaten vereint werden. Um es etwas kürzer zu fassen:
Grundidee ist es, den ebenfalls mit den Rauminformationen gespeicherten Sensornamen dazu zu verwenden, um den Index des Raumes in den Rauminfo-Daten zu erhalten, zu dem der aktuelle Eintrag im Messdaten-Datensatz gehört, über welchen iteriert wird:

```
let index = roomNames.indexOf(dataNames[q]);
// Index des Raumes, zu dem Daten an Stelle q gehören
```

Anschließend können über die ermittelten Indizes die entsprechenden Daten aus beiden Datensätzen zusammengefügt werden und als Objekt an die aufrufende Funktion zurückgegeben werden.

Auszug aus backend/database/utils/modify.js:

```
if (index >= 0) {
     newData.push({
       deviceId: roomData[index].deviceId, //datennames[q]
       room: roomData[index].roomName,
       temp: sensorData[q].temperature,
       hum: sensorData[q].humidity,
       upperTemp: roomData[index].upperTempLimit,
       lowerTemp: roomData[index].lowerTempLimit,
       upperHumi: roomData[index].upperHumiLimit,
       lowerHumi: roomData[index].lowerHumiLimit,
       message: roomData[index].message,
       _ts: sensorData[q]._ts,
     });
```

Ergebnis: Array aus Datensätzen der in den letzten drei Tagen sendenden Sensoren mit entsprechenden Rauminformationen (falls vorhanden).

Suchfunktion

Um eine funktionierende Suche nach Räumen in der Übersicht zu ermöglichen, müssen die angezeigten Daten mit Hilfe einer Pipe entsprechend der Eingabe in der Suchleiste gefiltert werden. Dazu wird eine entsprechende Pipe erstellt, die die Eingaben filtert.

Auszug aus utils/pipes/search-filter.pipe.ts:

```
return fullData.filter(dataEntry =>
     dataEntry.room != null && dataEntry.room != undefined &&
dataEntry.room.toLowerCase().indexOf(searchBarInput.toLowerCase()) != -1);
```

An die Transform Methode, die eine Pipe implementieren muss, werden diejenigen Dateneinträge zurückgegeben, für die zur aktuellen Eingabe in der Suchleiste auch ein Index im Daten-Array gefunden wird.

7. Datenbank

7.1. Planung

Bei der Planung der Datenbank wurde darauf geachtet, Stammdaten und Bewegungsdaten zu trennen. Das Ziel dabei war die Datenbankgröße zu reduzieren, um nicht bei jedem neuen Messwert auch die Grenzwerte neu zu speichern.

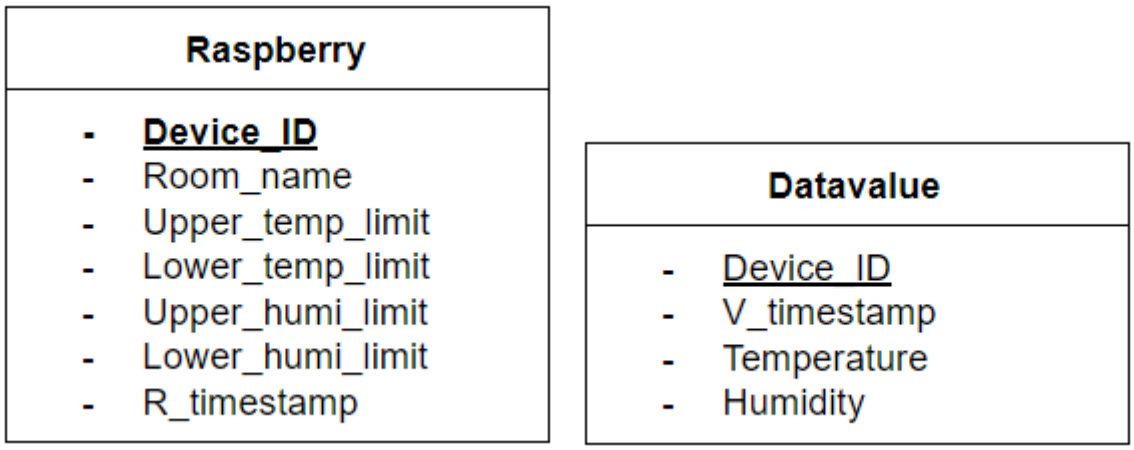


Abb. 14: Raum hinzufügen

Es wurde auch ein Primary-Key und ein Foreign-Key bereits festgelegt, um später durch Unterabfragen spezifische Daten zu erlangen. Leider unterstützt dies die Cosmos DB nicht. Dadurch wurde die Tabellenübergreifende Datenabfrage in Typescript oder NodeJs vorgenommen.

7.2. Umsetzung

Bei der Umsetzung der Datenbank wurde die SQL-Core API für die Cosmos DB gewählt, da bereits in Vorlesungen darüber gesprochen wurde. Der Aufbau der Cosmos DB besteht aus einer Datenbank, in der Collections und diesen wiederum Items liegen. Vergleichbar mit einem Eintrag in eine Tabelle einer Datenbank. Die einzelnen Items in der Cosmos DB benötigen jeweils eine eindeutige ID, damit sie eindeutig voneinander unterscheidbar sind. Aufgebaut sind die einzelnen Items wie ein JSON-Objekt, dadurch wird das Verarbeiten, Senden und Empfangen vereinfacht.
Von der Planung der Datenbank bis hin zur fertig aufgesetzten Datenbank wurde nicht mehr viel geändert. Die zwei unterschiedlichen Tabellen so wie die einzelnen Attribute wurden übernommen. Nur das Attribut Timestamp was die Zeit des Sendens (in Sekunden seit 1970) speichert, wird von Azure automatisch hinzugefügt.

8. Raumerstellung

In der Planung der Raumerstellung gab es anfangs sehr viele unterschiedliche Ansätze, wie diese umsetzbar ist. Dabei sind Ideen wie die Erstellung eines Zeichentools, Drag & Drop Raumerstellung oder ein Zuweisen durch ein Bild/Floorplan entstanden. Da diese Features aber den Zeitplan übersteigen, haben wir uns dafür entschieden, die Räume in einer Modularen Liste anzuzeigen. Somit hat sich die Raumerstellung auf einige Eingabefelder beschränkt. Dabei wurde besonders darauf geachtet, dem Anwender genug Informationen bereitzustellen, um mögliche Fehleingaben zu verhindern. Aus diesem Grund haben alle Eingabefelder einen Platzhalter sowie Informationen zu den maximal eingebbaren Zeichen oder Suffixe wie °C und % bei Temperatur oder Luftfeuchtigkeit.

8.1. Fehlerüberprüfung / Validation

Alle Eingabefelder haben eine Verifikation und/oder eine Eingabebeschränkung. Der Raumname ist ein Pflichtfeld und besitzt zusätzlich aus Design Gründen eine maximale Zeichenlänge von 10 Zeichen. Bei Nichteingabe oder Löschung des Raumnamens wird zusätzlich eine Fehlermeldung unterhalb des Eingabefeldes angezeigt.

Abb. 15: Inputfeld

Das zweite Pflichteingabefeld ist der Name bzw. die DeviceId des Sensors. Das Eingabefeld ist als Dropdown-Liste realisiert, welche alle verfügbaren Sensoren anzeigt. Auch hier wird eine Fehlermeldung ausgegeben, falls kein Raspberry ausgewählt wird.
Die Grenzwerte von Temperatur und Luftfeuchtigkeit sind Wahlfelder, in welchen nur Zahlen eingegeben werden dürfen.
Das letzte Eingabefeld ist eine Checkbox, ob dem Nutzer Meldungen über- bzw. unterschrittene Grenzwerte angezeigt werden sollen oder nicht. Hier gibt es keine Verifikation, da der Nutzer keine Fehlerhafte Eingabe tätigen kann.
Sobald alle Verifikationen erfüllt sind, ist der “Speichern“ Knopf aktiviert und der Raum kann gespeichert werden.

8.2. Freie Raspberry's

Um zu erkennen, welchem Raspberry noch keinem Raum zugewiesen ist, werden beim Seitenaufruf zwei queries an die Cosmos DB über die NodeJs API gesendet um zuerst jeweils alle Geräte herauszufinden, die Daten gesendet haben und welche Geräte einen Eintrag in der Raumtabelle (Collection)[3] haben. Nach Datenempfang werden dann die bereits verwendeten Raspberries von den sendenden abgezogen und das Ergebnis an die Dropdown-Liste übergeben.

8.3. Speichern

Beim Speichern wird zuerst der Raumname auf 10 Zeichen mit Leerzeichen aufgefüllt, sodass das Design einheitlich ist. Danach werden alle Eingabefelder ausgelesen in einen JSON-String umgewandelt und über ein http-POST an das Backend übergeben und dort in der Datenbank gespeichert. Zum Schluss wird noch eine Meldung "Raum <name> erfolgreich erstellt" dem Nutzer angezeigt, sowie die Eingabefelder geleert und der verwendete Raspberry aus der Dropdown-Liste gelöscht.

9. Einstellungen

Bei der Raumeinstellung soll der Nutzer die Möglichkeit haben, den Raum zu löschen oder einzelne Eingaben wie Raumname, Temperatur, Luftfeuchtigkeit und Benachrichtigungen zu ändern.

9.1. Aufruf

Der Aufruf der Unterseite erfolgt über die Detailansicht des jeweiligen Raumes. Deshalb wird bereits beim Aufruf der Raspberry Name mitgegeben, da dieser eindeutig ist, um die Information zu erhalten, welcher Raum bearbeitet werden soll. Beim Erzeugen der Unterseite wird dann eine Query an die Datenbank gesendet um die Rauminformationen aller Räume zu bekommen. Sobald die Daten verfügbar sind, wird der richtige Raum durch die beim Aufruf übertragene DeviceId gefunden. Die Benutzeroberfläche bietet die gleiche Funktion wie bei der Raumerstellung. Nur dass die Raumwerte bereits in die dafür vorgesehenen Felder eingetragen sind. Der Raspberry kann dabei nicht geändert werden.

9.2. Löschen / Ändern

Das Ändern von Werten des Raumes ist auch hier erst möglich, sobald die vorgegebenen Validierungen[4] erfüllt sind. Beim Ändern werden alle Daten der Eingabefelder an das Backend übergeben und in der Datenbank werden die gewünschten Felder ersetzt

[3] Genaueres zur Struktur der Daten unter dem Punkt Datenbanken
[4] Genaueres zur Validierung bei Raumerstellung

Auf ähnliche Art und Weise können Daten auch gelöscht werden.
Nach dem Löschen oder Ändern des Raumes wird eine Nachricht als Bestätigung an den Nutzer gesendet und er wird automatisch auf die Startseite/den Floorplan weitergeleitet.

10. Meldungs-Box

Das Meldungsfenster dient der Anzeige für Abweichungen bei einer Grenzwertüberschreitung. Durch ihren statischen Platz und nicht verändernden Inhalt kann der Nutzer direkt auf den ersten Blick erkennen, welche Räume Probleme verursachen. Wenn der Nutzer bei der Raumerstellung oder durch die Raumeinstellungen die Benachrichtigungen aktiviert hat, wird zusätzlich eine Benachrichtigung in Form eines Fensters am Unteren Bildschirmrand ausgegeben.

10.1. Meldungen

Bei Initialisierung der Hauptseite wird auch die Meldungsbox geladen und abonniert dann auch die “FullDataSubscription”, um immer die neuesten Daten für Räume und gemessene Werte zu erhalten. Durch eine Dynamische Liste werden dieser dann für die vier Fälle, Temperatur und/oder Luftfeuchtigkeit zu hoch oder zu niedrig mit dem entsprechenden Raum ausgegeben. Dazu wird für jeweils den entsprechenden Fall eine Funktion aufgerufen, die das Listenelement entsprechend sichtbar macht.

10.2. Benachrichtigungen

Die Benachrichtigungen nutzen dieselbe Funktion wie die Meldungen. Nur überprüfen sie zusätzlich, ob bei der Raumerstellung das gegebene Feld Benachrichtigungen ausgewählt wurde oder nicht. Bei jedem Empfang neuer Daten wird ein Array für alle Räume erstellt, falls dies noch nicht für den entsprechenden Raum erstellt wurde, das den Raumnamen sowie Boolesche Variablen für Temperatur/Luftfeuchtigkeit über/unter dem Grenzwert speichert. Dies wird genutzt um zu erkennen, ob Benachrichtigungen bereits gesendet wurden oder nicht. Bei Normalisierung werden die Variablen für den Wert dann zurückgesetzt und eine erneute Meldung dem Nutzer ausgegeben.

Hier als Beispiel der obere Temperaturgrenzwert.
Übergeben wird ein Raum (entry), der zuerst geprüft wird ob, Temperatur und Grenzwert-Informationen vorliegen. Gleichzeitig wird auch überprüft, ob der Temperaturwert den Grenzwert überschreitet. Ist einer dieser Bedingungen nicht erfüllt, wird der Raum nicht angezeigt und übergibt dem HTML ein logisches false zurück. Wenn die Bedingung jedoch erfüllt wird, wird noch überprüft ob eine Nachricht gesendet werden soll. Ist dies der Fall,

wird die Meldung gesendet und gespeichert und zuletzt ein logisches True zurückgegeben, um den Raum in der Warnungsbox anzuzeigen.

```
uppertemp(entry){
   if(entry.temp != null && entry.upperTemp!= null && entry.temp > entry.upperTemp){
     for (let entry1 of this.message)
       if(entry.message && entry1.room == entry.room && !entry1.upperTemp){
         entry1.upperTemp = true;
         let snackBarRef = this.snackbar.open("Übertemperatur in Raum: " + entry.room ,
'close', {duration: 20000});
       }
     return true;
   }
   return false;
 }
```

11. Detailansicht

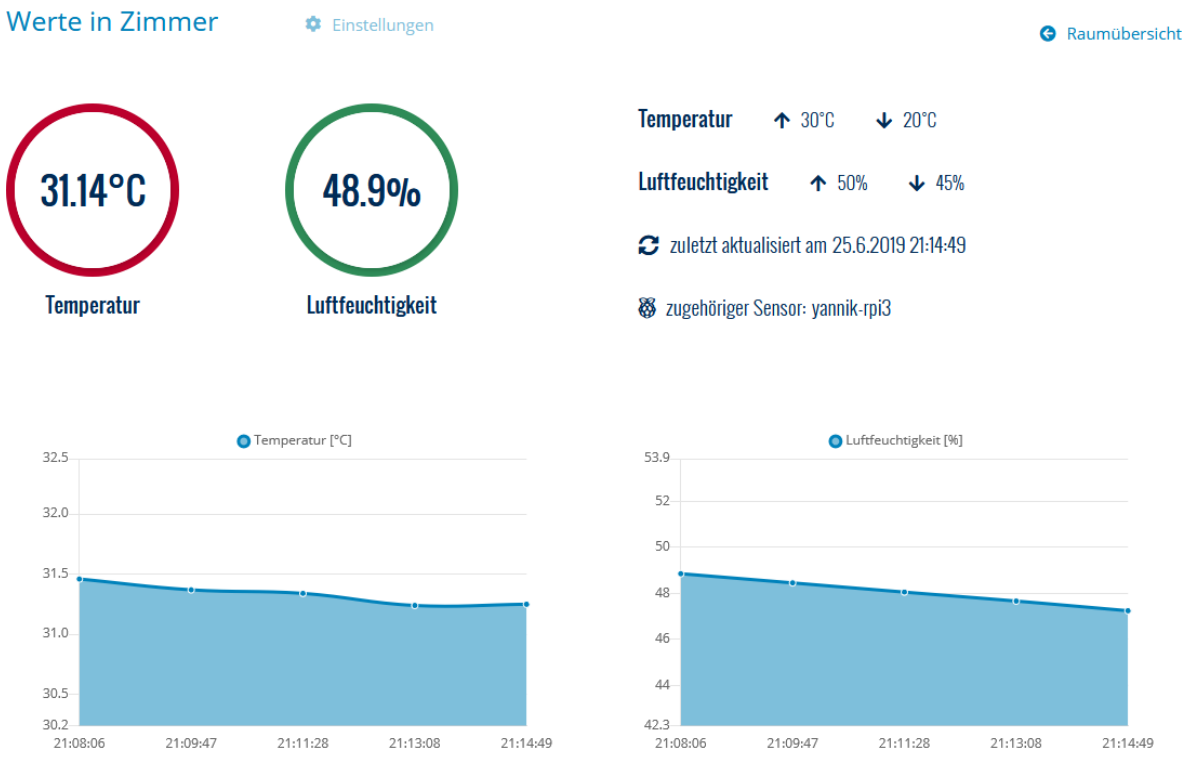


Abb. 16: Detailansicht

11.1. Temperatur- und Luftfeuchtigkeitsanzeige

Die Detailansicht gibt einen genauen Überblick über die gemessenen Werte eines Sensor. Die aktuellen Messwerte werden abhängig davon, ob ihre Grenzwerte über- oder unterschritten wurden rot, bzw blau umrandet. Liegt der Wert zwischen den zugeteilten Werten, werden diese grün umrandet.
Neben den jeweiligen Grenzwerten wird zusätzlich der dem Raum zugewiesene Sensor sowie der Zeitpunkt der letzten Aktualisierung angezeigt.

11.2. Statistiken

Um Temperatur- und Luftfeuchtigkeitsverläufe darstellen zu können wurde Chart.js eingesetzt. Chart.js ist eine JavaScript-Bibliothek, die einem erlaubt mit Hilfe von Canvas verschiedenste Diagramme zu zeichnen und zu gestalten. Zur Abfrage der Daten wurde eine eigene Query verwendet. Um das Result-Set dieser Abfrage im Rahmen zu halten werden aktuell die 100 neuesten Messwerte abgerufen und jeder zwanzigste Wert zur Darstellung verwendet. Das entspricht einem Abstand von ca. einer Minute und 40 Sekunden. Dieser Wert hätte beliebig hoch gewählt werden können, um beispielsweise Daten der letzten zwei Tage anzeigen zu können, haben uns aber aus Kostengründen dagegen entschieden. Um mehr Zeitraum abzudecken wäre es auch möglich die Senderate zu verringern (z.B. von 5s auf 1min).
Beide Diagramme verwenden jeweils den Maximalwert der angezeigten fünf Messwerte als Maximum und den Minimalwert der Messwerte als Minimum auf der y-Achse. Dies resultiert darin, dass in jeder Darstellung des Diagramms der maximale, sowie minimale Ausschlag dargestellt wird. Bei einer Zeitspanne von ca. acht Minuten und Temperaturunterschieden von eventuell 0.2 Grad ist das jedoch wenig sinnvoll. Um diese Ausschläge zu reduzieren wird auf das Maximum der Temperatur ein Puffer von einem Grad addiert bzw. dem Minimum ein Grad abgezogen. Der Luftfeuchtigkeit werden jeweils 5% Puffer hinzugefügt bzw. abgezogen.
Die Statistiken werden nur beim Aufruf der Komponente aktualisiert, da sich eine Datenbankabfrage alle 5 Sekunden besonders bei einer größeren dargestellten Zeitspanne nicht lohnen würde.

12. Review-/Testphase und Auswertung

12.1. Welche Requirements wurden umgesetzt?

Must

Die Daten sollen in einem Datenarchiv gespeichert werden.
Zur Erstellung von Statistiken und zur Nachverfolgung werden die Daten für einen bestimmten Zeitraum gespeichert.

Die Daten werden in einer Datenbank (CosmosDB) für fünf Tage gespeichert und die letzten fünf gemessenen Werte werden dabei jeweils in einem Liniendiagramm dargestellt.

Das System soll Meldungen und Warnungen ausgeben und darstellen.
Meldungen und Warnungen entstehen wenn Grenzwerte überschritten oder unterschritten werden oder das Gerät einen Fehler aufweist. Bei Bedarf können Benachrichtigungen als Pop-Up oder E-Mail gesendet werden.

Die Meldungen werden rechts ausgegeben und sind somit immer sichtbar. Im Browser werden zudem kleine PopUp Meldungen am Ende der Seite angezeigt. Eine Benachrichtigung per E-Mail wurde nicht umgesetzt.

Die WebApp soll eine Funktion zur Erstellung einer Gebäudedarstellung besitzen.
Um Räume und Stockwerke darzustellen in denen gemessen wird soll das System eine Möglichkeit bieten das Gebäude zu rekonstruieren.

Das System bietet die Möglichkeit Räume darzustellen und hinzuzufügen. Auf eine Darstellung des Gebäudes bzw. von Stockwerken wurde verzichtet, da der gewählte Weg einfache und kurze Wege ermöglicht.

Das System soll über Einstellungen verfügen.
Es soll möglich sein Grenzwerte für die gemessenen Werte pro Raum festzulegen um dem Nutzer eine genaue Überwachung zu ermöglichen.

In der Detailansicht kann man auf die Einstellungen zugreifen und seine eingegebenen Werte verändern. Zudem ist es möglich die Grenzwerte für Temperatur und Luftfeuchtigkeit schon in der Raumerstellung festzulegen.

Should

Zur besseren Übersicht und als Rückblick sollen Statistiken eingeblendet werden.
Statistiken sollen immer in Form eines Liniendiagramms für einen bestimmten Zeitraum angezeigt werden. Über einen Button kann man zwischen Temperatur und Luftfeuchte wechseln.

Beide Statistiken werden gleichzeitig eingeblendet, dabei werden die letzten fünf gemessenen Werte in Form eines Liniendiagramms angezeigt.

Der Nutzer soll die Möglichkeit haben Sensoren/Räumen und Stockwerken Namen zu geben und zu bearbeiten.
Für eine bessere Übersicht und eine mögliche Suche soll der Nutzer bei der Raumerstellung dazu in der Lage sein z.B. Räumen Namen zu geben. Diese Namen soll er nachträglich ändern können.

Sowohl bei der Raumerstellung, als auch in den Einstellung muss der Nutzer dem Raum einen Namen vergeben bzw. kann er diesen bearbeiten.

Das System soll eine Suchfunktion besitzen.
Diese Funktion geht mit der Namensgebung einher. Die Suchfunktion soll dem Nutzer eine schnellere Möglichkeit bieten Räume oder Sensoren zu finden.

Die Suchfunktion ermöglicht dem Nutzer schneller Räume zu finden.

Could

Bei Inputfeldern soll der Nutzer Unterstützung durch Hilfetexte bekommen.
Wenn der Nutzer einen falschen Begriff eingibt soll das System dies erkennen und ihn darauf Hinweisen bzw. angeben, was eigentlich zu tun ist.

Would

Aufgrund von Zeitmangel wurde keines der Would-Requirements umgesetzt.

12.2. Welche Features wurden ergänzt?

In der Detailansicht wurde die Uhrzeit und das Datum durch zuletzt aktualisiert, einer Darstellung der Grenzwerte und dem zugehörigen Sensor ersetzt.

Zudem wird das Unter- oder Überschreiten von Grenzwerten durch eine farbliche Darstellung erkenntlich gemacht. Rot bedeutet überschritten, Grün, dass alle Werte im Bereich sind und Blau, dass der untere Grenzwert unterschritten wurde.

In der Raumübersicht und auch in der Detailansicht wird der Sensor für den jeweiligen Raum angezeigt, um so einen besseren Überblick zu erhalten.

12.3. Probleme

12.3.1. Styling (CSS)

Browserkompatibilität
Was z.B. die Scrollbars und Schriften betrifft werden diese oft unterschiedlich angezeigt und dargestellt. Daher muss z.B. für Chrome eine eigene Scrollbar gebaut werden.

```
ul::-webkit-scrollbar{
 background-color: #0082BB;
 width: 7px;
}
```

Mit -webkit kann man speziell für Chrome Elemente anpassen, die sonst durch vorhandene Browsereinstellungen nicht übernommen werden.

Responsiveness
Aufgrund der Nutzung wurde die WebApp nicht direkt als Mobile First entworfen, da die Übersicht und Bearbeitung auf dem Desktop sehr viel angenehmer ist.

Ausgehend von einer einfachen Entwicklung wurde das Design darauf angepasst, dass nur das große Fenster entfernt wird und man die kleinen Boxen rechts als Hauptansicht auf dem Handy hat.

Das Arbeiten mit den Viewport-Größen und Media-Queries ist allerdings etwas verzwickt und führt so oft zu Verwirrungen. Bis jeder Bereich responsive wurde, hat es einiges an Versuchen gekostet. Im Endeffekt ist jeder Bereich auf einer Bildschirmgröße von 15 Zoll bis 27 Zoll (gängige Größen) in der Desktopansicht responsive.

```
@media only screen and (max-width: 600px) { /*verschwindet bei Mobile-
Version*/
   .room {
     display: none;
   }
 }


@media only screen and (max-width: 600px) {
 .warning {
   margin-top: -1vh;
   width: 90vw;
   height: 38vh;
   margin-left: -67vw;
 }

 ul {
   height: 26.5vh;
   margin-left: -40px;
 }

 p {
   font-size: 4.5vw;
 }
```

12.3.2. Datenbank

Das Verbinden, lesen und schreiben von Daten in die Datenbank gingen mehrere Probleme mit sich. Zuerst musste eine Verbindung durch C# mit dem Raspberry hergestellt werden. Eine direkte Verbindung zur Datenbank ist nicht verfügbar, so mussten die Daten erst über den IOT-Hub von Azure und dann mit einer Azure Funktion in die Datenbank schreiben. Da es bei diesem Schreibvorgang drei unterschiedliche Programme betroffen hatte war das Testen der einzelnen Komponenten auf Funktionalität erschwert.

Auch das Lesen der Daten in der WebApp mit Angular hat seine Zeit beansprucht, da zuerst das Backend über NodeJs verfügbar sein musste. Auch Unterabfragen durch eine Query zwischen zwei Collections der Datenbank war über die Cosmos DB nicht machbar, was mehr Arbeit in der Datenverarbeitung erforderte. Das Löschen und Updaten von Elementen hat anfangs Probleme verursacht. Mit Hilfe der Microsoft Docs zur Azure Cosmos DB SQL API hat sich dieses Problem beheben lassen.

12.3.3. Angular und Node/Express Backend

Natürlich traten auch beim Aufsetzen des gesamten Backens und Angular-Frontends einige Fehler bzw. Schwierigkeiten; ob CORS-Errors oder schlichtweg einfache Implementierungsfehler, auf welche wir erst nach ausgiebigem Testen aufmerksam wurden. Des Weiteren ließen sich einige Fehler durch Konsolenausgaben sehr gut lokalisieren, was sehr zur Behebung dieser beigetragen hat. Vieles dauerte länger als erwartet, eher weniger funktionierte auf Anhieb. Von einer Aufzählung aller Fehler/Schwierigkeiten sehen wir hier einmal ab.

12.4. Fazit

Aufgrund von Zeitmangel und dem Versuch die WebApp so einfach und nutzerfreundlich wie möglich zu gestalten, wurde der ursprüngliche Plan oft ein wenig abgewandelt. Dennoch ist am Ende ein für die Gruppe zufriedenstellendes Ergebnis entstanden, welches viele der geplanten Elemente enthält oder sogar durch einige Features ergänzt wurde.

Die Entscheidung das Projekt zu wählen wurde getroffen, da jedes Gruppenmitglied etwas dazulernen wollte und sein im Studium gewonnenes Wissen anwenden konnte.
Dieses Ziel wurde erreicht. Das Ganze bot die Möglichkeit in einer Gruppe mit Projektplanung, Rollenverteilung und Versionsmanagement zu arbeiten und so neben neuen Technologien auch Techniken kennenzulernen, die in der Industrie angewandt werden.