IHK-Essen
MEO FI Anw. 1 (AP T2V1)
Abschlussprüfung Sommer 2022

Fachinformatiker/Fachinformatikerin (VO 2020)
Fachrichtung:Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Prototyp zur Überwachung der Luftqualität

Entwurf und Bau eines Prototyps zur Überwachung der Luftqualität in den Schulungsräumen der IT-Akademie.

Abgabedatum: Essen, den 04.05.2022

Prüfungsbewerber:

Norbert Heimsath

Mergelstraße 25

45478 Mülheim an der Ruhr

norbert@heimsath.org

0178 341 7997

Ausbildungsbetrieb:

IT-Akademie Dr. Heuer GmbH

Konrad-Zuse-Straße 2b

44801 Bochum

office@drheuer.de

0234 33855860

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

API
: Application Programming Interface

CSV
: Comma Separated Values

EPK
: Ereignisgesteuerte Prozesskette

ERM
: Entity Relationship Model

GUI
: Graphical User Interface

HTML
: Hypertext Markup Language

LAN
: Local Area Network

MVC
: Model View Controller

PHP
: PHP Hypertext Preprocessor

SMTP
: Simple Mail Transfer Protocol

SQL
: Structured Query Language

Verzeichnisse

SVN
: Subversion

VLAN
: Virtuelles LAN

XML
: Extensible Markup Language

Projektbericht

1. Einleitung

Die folgende Projektdokumentation schildert den Ablauf des IHK-Abschlussprojektes, welches der Autor im Rahmen ihrer Weiterbildung/Externenprüfung zum Fachinformatikerin Fachrichtung Anwendungsentwicklung durchgeführt hat.

Da es sich bei dem Projekt um einen Prototypen handelt war natürlich mit Problemen und Änderungen im Laufe des Projekts zu rechnen und natürlich sind diese auch reichlich aufgetreten. Am Ende steht aber wie ich finde ein Interessantes und Ausbaufähiges Ergebnis das aber noch wesentlich weiter entwickelt werden muss.

Das Projekt musste im Zeitraum vom 15.03.2022 – 05.05.2022 stattfinden, einem Zeitraum der sich für Externe Prüflinge mitten in der Vorbereitung der Prüfungen für Teil 1 (30.03.) und Teil 2 (04.05). Damit musste das Projekt immer um die Vorbereitungskurse herum ausgeführt werden oder Vorbereitungskurse mussten ausfallen. Alles in allem eine sehr schwierige Situation. Es bestand damit überhaupt keine Zeit eine geschönte Dokumentation zu erstellen, also wird das Desaster im vollen Umfang dokumentiert.

2. Projektplanung

2.1. Entwicklungsprozess

Der Ablauf des Entwicklungsprozesses war durch die Dr. Heuer GmbH vorgegeben. Die generelle Planung des Projektes und seiner Teilkomponenten sollte konventionell durchgeführt werden und die Implementierung sollte mit Hilfe des **Spiralmodells(nach Boehm)** stattfinden. In der Planung sollten als einzelne Schritte identifiziert und spezifiziert werden, wobei die Reihenfolge der Ausführung und weitere Details in der Implementierung Schritt für Schritt sich in den Iterationen der Spirale ergeben.

Die erste Schwierigkeit die sich hierbei ergab, war das es mehrere Definitionen des Spiralmodells gibt. Vor allem überraschend war, das die verbreitete deutsche Fassung, die auch in die Wikipedia Einzug gefunden hat leider nicht der original Definition von Boehm entspricht. So beginnt sie mit dem 2. Quadranten und verzichtet auf eine Definition der Ziele für die Erste Iteration.

Das Spiralmodell ist ein iteratives Vorgehensmodell in dem alle Entwicklungsphasen mehrfach spiralförmig durchlaufen werden. Die Phasen sind:

1. Beschreiben der Rahmenbedingungen, festlegen der Ziele und Alternativen.
2. Risikoanalyse, Evaluierung der Alternativen, beseitigen der Risiken such Analysen, Simulationen oder Prototyping.
3. Zwischenprodukt Realisieren und Überprüfen.
4. Analyse des jetzigen und Planung des nächsten Zyklus.

In der Implementierung werden Schritte 1-4 immer wieder durchgegangen und die Zwischenergebnisse festgehalten.

Die kontinuierliche Überprüfung macht abschließende Tests nicht überflüssig, aber reduziert deren Aufwand drastisch. Auch Probleme mit dem User Interface oder zusätzlich Wünsche können direkt einfließen.

Um Unklarheiten zu vermeiden habe ich die Originalgrafik noch einmal hier eingefügt:

Projektplanung

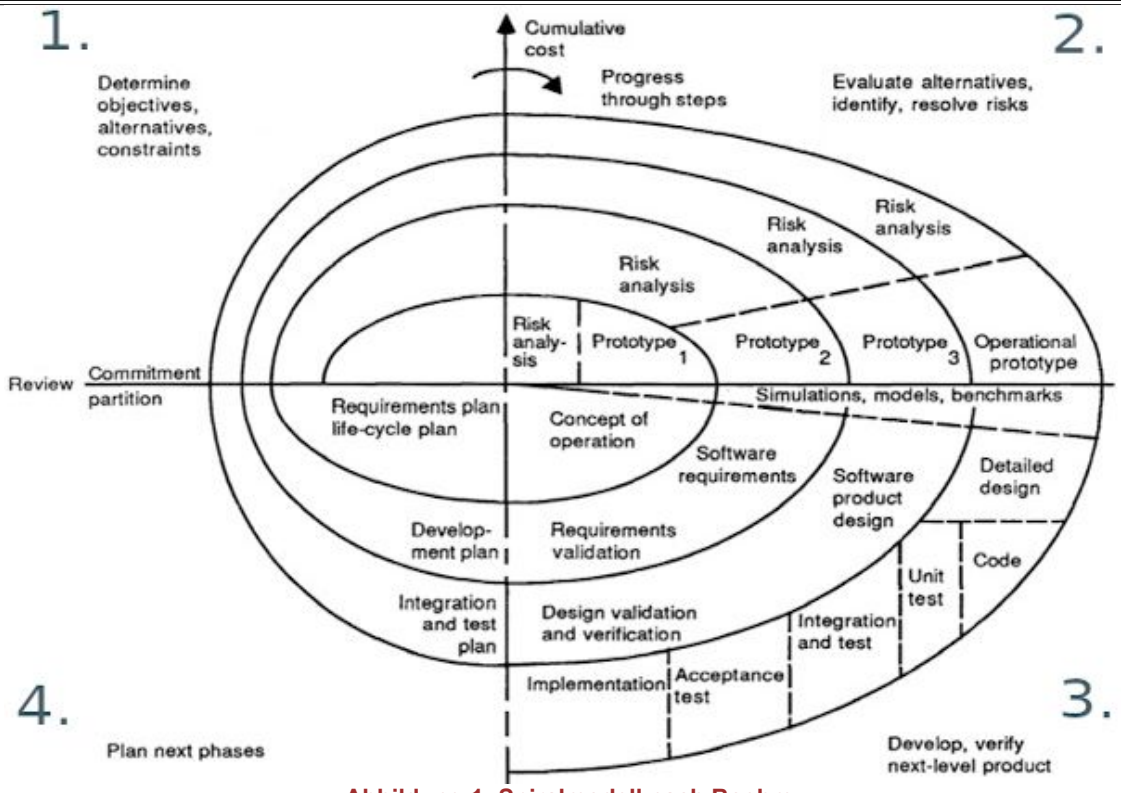


Abbildung 1: Spiralmodell nach Boehm

Eine vor weggehende Planung war mir in diesem Projekt trotzdem wichtig damit ich einen groben Überblick hatte wohin es überhaupt gehen soll. Da es sich bei dem Projekt um einen Prototypen handelt war es jedoch von Vorteil einen fast schon agilen Prozess für die Ausführung zu nutzen, da Probleme und Abweichungen absolut zu erwarten waren. Zusätzlich waren ja Pflichtenheft und DV Konzept ausdrücklich gefordert.

2.2. Zeitplanung

Sämtliche Identifizierte Einzelschritte wurden von mit Thematisch zusammengefasst um trotz Spiralmodell eine ungefähre Zeitplanung durchführen zu können.

Projektphase	Zeit
Analysephase	12 h
Entwurfsphase	12 h
Implementierungsphase	37 h
Abnahme und Deployment	3 h
Dokumentation	13 h
Pufferzeit	3 h
Summe	**80 h**

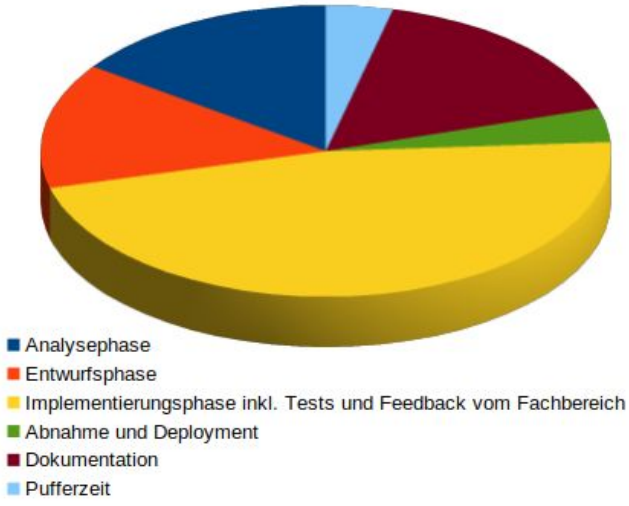


Tabelle 1: Grobe Zeitplanung

PROTOTYP ZUR ÜBERWACHUNG DER LUFTQUALITÄT
Entwurf und Bau eines Prototyps zur Überwachung der Luftqualität
in den Schulungsräumen der IT-Akademie.

Projektplanung

Analysephase	**12**
Durchführen der Ist-Analyse	2
Ermitteln von Use-Cases und Erstellung eines Anwendungsfall-Diagramms	2
Ermitteln der benötigten Sensordaten, nur CO^2 oder auch andere ?	1
Recherche über geeignete Sensoren und deren Kosten und Kompatibilität mit günstigen Development Boards und Verfügbarkeit.	1
Erstellen der Sensorkonzepte (Prototypen mit unterschiedlichen Preisen und Qualitäten)	2
Durchführen der Wirtschaftlichkeitsanalyse eventuell erstellen einer Break-Even- Analyse und Amortisationsrechnung	2
Unterstützung des Fachbereichs bei der Erstellung des Lastenheftes	2
Entwurfsphase	**12**
Auswahl eines Passenden Webframeworks und passender Datenbank	1
Erstellen eines Plans zur Kommunikation zwischen Sensoren und Server	1
Entwurf der Grundstruktur der Sensoranordnung und des Workflows mit dem Sensoren hinzugefügt und Entfernt werden	1
Entwerfen der Datenbankstruktur inkl. Erstellen eines ER-Modells	2
Ableitung des Tabellen- und Domänenmodells aus dem ER-Modell	1
Planung der Architektur inkl. Erstellen eines Komponentendiagramms	1
Entwerfen der Benutzeroberfläche inkl. Erstellen von Mock-Ups	2
Erstellen des Pflichtenheftes	3
Implementierungsphase inkl. Tests und Feedback vom Fachbereich	**37**
Webserver	
OS Installieren und Einrichten.	1
Apache Webserver , PHP, Datenbanksystem einrichten	1
Grafische Benutzeroberfläche, IDE und Development Tools , Fernwartung	1
Sensoren	
Zusammenbau mit Lötarbeiten und Fräsarbeiten für Gehäuse	2
Programmierung der verschiedenen Konfigurationen	6
Funktionstests	2
Webanwendung	
Framework/CMS einrichten und Konfigurieren inklusive benötigter Module	2
Datenbankstrukturen im Framework anlegen	3
Module und Hauptanwendung erstellen	16
Dashboard mit Abfragen und Modulen konfigurieren und einrichten	3
Abnahme und Deployment	**3**
Code-Review mit dem Ausbilder in mehreren Iterationen	2
Abnahme und Usability Tests durch die Fachabteilung in allen relevanten Iterationen	1
Dokumentation	**13**
Erstellen der Entwicklerdokumentation	1
Erstellen des Benutzerhandbuches	4
Projektdokumentation	8
Pufferzeit	**3**
Gesamtstunden:	**80**

Tabelle 2: Detailierte Zeitplanung

2.3. Abweichungen vom Projektantrag

Die Pufferzeit ist separat ausgewiesen, da sie nicht nur der Implementierung dient sondern generell zur Verfügung steht.

2.4. Ressourcenplanung

Bei der Auswahl der verwendeten Software wurde darauf geachtet, dass diese kostenfrei (z.B. als Open Source) zur Verfügung steht. Damit sollen die anfallende Projektkosten möglichst gering gehalten werden.

2.4.1. Hardware

- Bauteile für die Sensoren(Mikroprozessor Board, CO^2 Sensor, Platinen Gehäuse…) Eine detaillierte Auflistung befindet sich in der Wirtschaftlichen Betrachtung @todo link it
- Ein Raspberry PI Minirechner als Webserver
- Die Server und Netzwerkarchitektur der Dr. Heuer GmbH wird mit benutzt.

2.4.2. Software

- Arduino IDE (C++ Programmierung Sensoren)
- Raspian OS
  (Schlanke Linux Distribution, speziell für den Raspberry PI, basiert auf Debian)
- Apache 2 Webserver
  - Mit Unterstützung für Mod Rewrite, .htaccess
- MariaDB Datenbanksystem
- PHP Version 7+ Skriptsprache mit „PHP short tags“ aktiviert .
  - PHP Modul GD2 oder Image Magic Grafik Bibliothek
  - PHP Modul BCMATH
  - PHP Modul mbstring
- Real VNC und SSH zur Fernwartung des Servers.
- OpenSSL zur Verschlüsselung.
- Processwire CMS als Application Framework
  - ListerPro , verbeserte Darstellung von Tabellenübersichten.
  - AdminOnSteroids , verbesserte Backend Verwaltung.
  - AdminRestrictPageTree Erweiterte Einschänkung von Berechtigungen.
  - ConnectPageFields erleichtert Bidirektionale Verbindungen von Objekten.
  - Dashboard, Grundfunktionen und Javascript Bibliotheken für die Anzeige eines Dashboards
  - Verschiedene Module für Datenfelder(Button, FontIconPicker,RockAwesome, IconPicker)
  - Less Modul zur einfachen/dynamischen Anpassung des Templates(Less statt statischem CSS)
  - SymmetricEncryption zur verschlüsselung von Textfeldern.
  - Tracy Debugger
  - WireMailSmtp Klasse für SMTP Anbindung.
- VIM als Entwicklungsumgebung (VI Improved).

2.4.3. Personal

- Ich selbst.
- Mein Projektbetreuer Herr Paul.
- Gelegentlich einen Mitarbeiter der Fachabteilung zum testen.

2.5. Wirtschaftlichkeit

2.5.1. Wie viele Messpunkte werden benötigt?

Die DHG besitzt an beiden Standorten zusammen insgesamt 12 Schulungsräume und 6 Büros. Pro Schulungsraum werden 2 Sensoren benötigt, pro Büro nur einer. Da in allen Gebäuden Betondecken installiert benötigen wir bei den Kommerziellen Systemen mindestens 4 Basisstationen(2 pro Etage), da diese Systeme alle ein eigenes WLAN aufspannen um mit Ihren Sensoren zu kommunizieren. Das heißt bei den Kommerziellen Systemen benötigen wir 30 Sensoren und mindestens 4 Basisstationen. Da unser Eigenbau normales WLAN nutzt benötigen wir dort nur die 30 Sensoren.

2.5.2. Vorgehen

Zuerst habe hat der Autor sich auf die suche nach käuflichen Raumluft Kontrollsystemen umgeschaut. Danach wurden die Prototypen für die Testsensoren festgelegt und zum Schluss ein Vergleich durchgeführt. Die Kosten für geleistete Arbeitsstunden sind von der Dr. Heuer GmbH mit 43,29 € / Stunde angegeben worden.

2.5.3. Kommerzielle Sensor Systeme

Bei den Kommerziellen Sensor Systemen war es Relativ schwierig, Systeme zu finden die zumindest im weisesten Sinne den Anforderungen des Pflichtenheftes entsprechen.

Die 3 Systeme die ausgemacht werden konnten habe in In Anhang Fehler: Verweis nicht gefunden Fehler: Verweis nicht gefunden Seite Fehler: Verweis nicht gefunden zusammengefasst.

Entschieden habe ich mich für das Modell Monitoring CA-M Kohlendioxid von Clean-air-engineering, das pro Sensor mit 547,40 €, pro Basisstation mit 458,15 € und für die Management Cloud mit mindestens 199,38 € / Jahr zu Buche schlägt. Die Geräte müssen weiterhin immer noch an der Wand befestigt werden, die Basistationen eingerichtet und mit den Sensoren verbunden werden. Auch muss die Firewall dazu eingerichtet werden den Datenverkehr zwischen den Basistationen und der Cloud zu erlauben. Der insgesamt von mir geschätzte Arbeitsaufwand beträgt etwa 10 Stunden. Die Kosten für die Cloud berechne ich auf 2 Jahre, das entspricht dem üblichen Garantiezeitraum für Elektronische Geräte

Tabelle 3: Kosten Sensorsystem Kauflösung

Posten	Preis	Anzahl	Gesamt
Sensor	547,40 €	30	16422,00 €
Basistation	458,15 €	min. 4	1832.60 €
Cloud	Min. 199,38 €	2	398.76 €
Arbeitsstunden	43,29 €	10	432,29 €
Summe			**19085,65 €**

Vorteile:

- Kaufen, Installieren und Fertig(hoffentlich)
- Gerätegarantie und möglicherweise ein wenig Support vom Anbieter

Nachteile:

- Defekte Geräte können nicht vor Ort repariert werden. Nach ende der Garantiefrist bleibt nur der Komplette Austausch der Einzelgeräte.
- Möglicherweise gibt es nach ein paar Jahren keine Austauschgeräte mehr , da der Hersteller ein neues System benutzt. Dann muss das ganze Netz ersetzt werden.
- Wenn man alle Funktionen nutzen möchte ist man von der Cloud abhängig und den Preis für die Vollversion(Enterprise Level) gibt es nur auf Anfrage.

2.5.4. Eigenbau System

Es sollten 3 Sensor Prototypen Zusammengestellt werden werden alle mit Unterschiedlichen Ausstattungen.

Die detaillierte Übersicht finden Sie im Anhang 10.1 Sensor Prototypen Eigenbau Seite xvii

Für die Berechnung nehmen wir hier die teuerste und die günstigste Variante. Und als Basis-station dient ein Raspberry PI Minicomputer der mit Gehäuse 81,90 € gekostet hat. Da sich auch Dieser genauso wie die NodeMCU boards nahtlos in die bestehende WLAN Architektur einfügen, wird nur eine Basisstation benötigt.

Tabelle 4: Kosten für Sensorensystem Eigenbau

Posten	Preis	Anzahl	Gesamt V1	Gesamt V2
Sensor	**29,18 € / 61,24 €**	30	875,40 €	1837,20
Basistation	81,90 €	1	81,90 €	81,90 €
Arbeitszeit	43,29 €	80	3463,20	3463,20 €
Summe			**4420,50 €**	**5382,30 €**

Vorteile:

- Alle Sensoren können repariert werden.
- Sensortypen können erweitert und Verbessert werden.
- Auch die Kommunikationsschnittelle kann angepasst werden.
- Die Applikation kann beliebig erweitert werden
- Die Einzelteile für Sensoren werden eher günstiger als teurer.

Nachteile:

- Mehr Arbeitszeit benötigt.
- Fachkräfte werden benötigt.

2.5.5. Fazit

Es ergibt sich sofort eine Beträchtliche Preisdifferenz:

Bei Variante 1: 19085,65 € - 4420,50 € = **14665,15 €**
und
Bei Variante 3: 19085,65 € - 5382,30 € = **13703,35 €**

Das entspricht **338** bzw. **316** weiteren Entwicklerstunden die im Vergleich zur Kommerziellen Lösung noch zur Verfügung stehen.

Weitere Vorteile:

- Reparaturen an den Sensoren sind problemlos möglich also max. 23,40 € statt 547,40 € für einen neuen Sensor.
- Die Cloud Kosten fallen komplett weg.

Die Entscheidung fällt eindeutig zu Gunsten der Eigenbaulösung aus!

2.6. Datenübertragung für die Sensoren

Bei den ersten Überlegungen wie denn die Daten am besten von den Sensoren zur Applikation übertragen werden sollten kamen in Grunde genommen nur 2 Varianten in Frage nämlich JSON und XML. Alte Formate wie SOAP kamen schon aufgrund der Sicherheitsprobleme(z.B. Programmsteuerung über SOAP Nachrichten) nicht in Frage. Der Transport sollte über HTTPS zu realisieren sein.

2.6.1. JSON

Die **J**ava**S**cript **O**bject **N**otation (JSON) ist ein kompaktes Datenformat in einer einfach lesbaren Textform für den Datenaustausch zwischen Anwendungen. JSON ist von Programmiersprachen unabhängig. Parser und Generatoren existieren in allen verbreiteten Sprachen. *Quelle: Wikipedia*

2.6.2. XML

Die Extensible Markup Language (dt. Erweiterbare Auszeichnungssprache), abgekürzt XML, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten im Format einer Textdatei, die sowohl von Menschen als auch von Maschinen lesbar ist. *Quelle: Wikipedia*

2.6.3. Entscheidungsmatrix

Feature	JSON	XML
Datenmenge	+ (kaum overhead)	- (deutlicher Overhead)
Geschwindigkeit	+ (schnell)	- (langsam)
Implementierung in PHP	+ (gut)	0 (kompliziert)
Komplexität	+ (einfach)	- (komplex)
Unterstützt Arrays	+ (ja)	0 (bedingt)
Namespace Support	- (nein)	+ (ja)
Leicht zu validieren via DTD	- (nein)	+ (ja)
Gesamt	**+++ (3)**	**- (-1)**

Tabelle 5: Entscheidungsmatrix Datenübertragung

2.6.4. Fazit

Eine Validierung über eine Document Type Definition (DTD) oder Namespace Support sind in dieser Anwendung nicht wirklich hilfreich. Damit negieren sich auch noch die wenigen Vorteile von XML. **Alles in allen entscheide ich mich deswegen für JSON.**

2.6.5. ! NACHTRAG !

Im Verlaufe des Projekts kristallisierte sich immer mehr heraus das mittlerweile MQTT zum quasi Standard für „Internet of Things“ (IoT) Geräte wird. MQTT basiert auf einem Server der allen zu ihm gehörenden IoT Geräten sozusagen eine Plattform bietet an die sie Ihre Mitteilungen Senden können. Diese Daten werden dann in bestimmten Kanälen zur Verfügung gestellt und andere Geräte müssen diese Kanäle abonnieren um dann dort die Daten mitzulesen.

Da der zeitliche Rahmen des Projekts es nicht zulässt sich tief in dieses Konzept einzuarbeiten werden wir weiterhin die Daten direkt per JSON übertragen. Ein MQTT Server/Broker kann später eingefügt werden zumal es sich beim Payload auch bei MQTT oft um JSON Daten handelt da diese die Möglichkeit bieten alle Arten von Werten in einer einfachen Textnachricht zu übertragen. Damit muss dann später nur eine weitere Ebene in die Kommunikation eingezogen werden.

2.7. Auswahl der Programmierumgebung

Da ich seit vielen Jahren Webanwendungen mit PHP schreibe viel die Entscheidung relativ leicht. Zumal es keine gute Idee ist eine Projektarbeit in Sprachen auszuführen mit denen man nur wenig vertraut ist. Weiterhin bin ich in PHP mit verschiedenen Frameworks vertraut. Vor allem da der Umfang des Projektes relativ groß ist, wäre es ziemlich sinnfrei alle Teilelemente vollständig selbst zu programmieren. Bleibt also noch die Auswahl eines passenden Frameworks als Grundlage.

2.7.1. Framework

Da das ganze Projekt auf einem Raspian Minirechner (Raspi) laufen soll ist die Menge die an Resourcen zur Verfügung steht eher gering. Damit fallen Umgebungen wie Zend Framework oder Synfony schon aus da der Bedarf an Rechenleistung und Speicher eher für größere Server geeignet ist. Leider ist es in Mode gekommen solche Anforderungen öffentlich freizugeben, so das ich mich da nur auf Erfahrungswerte verlassen kann.

Bleiben hier also noch leichtgewichtige Frameworks zur Auswahl. In meinem Fall sind das Laravel und Processwire. Beide bringen einen guten Funktionsumfang mit und sind nicht all zu schwerfällig. Eine Gegenüberstellung findet sich in Fehler: Verweis nicht gefunden Fehler: Verweis nicht gefunden Seite Fehler: Verweis nicht gefunden

2.7.1.1.Worauf legen wir bei der Auswahl besonderen Wert?

Da das Projekt trotz ziemlichen Umfangs in 80h nebst Planung und Dokumentation abgeschlossen sein muss ist Rapid Development sehr wichtig. Sparsame Ressourcennutzung ist ein weitere wichtiger Punkt. Bei auftretenden Problemen ist es wichtig schnell eine Lösung zu finden, deswegen irecht hoe Gewichtung für gute Dokumentation/Support. Testing ist grundsätzlich nicht unwichtig. Auch die Vertrautheit mit einem System ist als Projektüberlegung wichtig, da es den ganzen Prozess beschleunigt. Synchronisation und Rollout werden zwar erst wirklich wichtig wenn der Prototyp in Serie gehen sollte aber auch das sollte berücksichtigt werden. Den gesamt möglichen Funktionsumfang lasse ich hier außen vor, da die Applikation einen eng umgrenzten Einsatzbereich hat. Wenig Wartungsaufwand und problemlose Updates sind hier in einem Punkt zusammengefasst. Zu guter Letzt ist Sicherheit natürlich immer ein wichtiges Thema.

Eigenschaft	Gewichtung	*Processwire*		*Laravel*	
Ressourcenbedarf	**4**	**6**	24	**4**	16
Rapid Development	**6**	**5**	30	**4**	24
Basis Funktionalität	**3**	**5**	15	**2**	6
Testing	**2**	**2**	4	**5**	10
Vertrautheit	**3**	**4**	12	**3**	9
Sicherheit	**5**	**5**	25	**3**	15
Update/Wartung	**4**	**5**	20	**4**	16
Dokumentation/ Support	**4**	**5**	20	**4**	16
Sync und Rollout	**2**	**3**	6	**5**	10
Nutzwert			**156**		**122**

Tabelle 6: Entscheidungsmatrix Framework (Max. Wert = 6)

2.7.2. Fazit

Trotz der Mächtigkeit von Laravel halte ich Processwire für diese Projekt für das besser geeignete System.

2.7.3. Datenbanksystem

Da Processwire in Zusammenarbeit mit MariaDB oder MySQL die besten Resultate liefert wähle ich einfach aus persönlicher Vorliebe **MariaDB** aus. Zumal das auch die unter Debian/Raspian bevorzugte DB ist und sich damit auf den geplanten Minirechner gut einrichten lässt.

3. Entwurfsphase

3.1. Zielplattform

3.1.1. Server

Als Serverrechner ist ein Raspberry PI minirechner vorgegeben, dieser muss neu eingerichtet werden. Das standard Betriebssystem für diese Rechner ist Raspian ein Leichtgewichtiger Klon der Debian Linux Distribution. Im Zweifelsfalle können auch original Debian Pakete installiert werden. Folgede Features müssen installiert werden:

- Apache Webserver mit verschlüsselter Datenübertragung (HTTPS)
- Ein selbstsigniertes Zertifikat für HTTPS
- Ein Datenbanksystem
- PHP
- Remote Zugriff
- X-Window System
- Arduino IDE

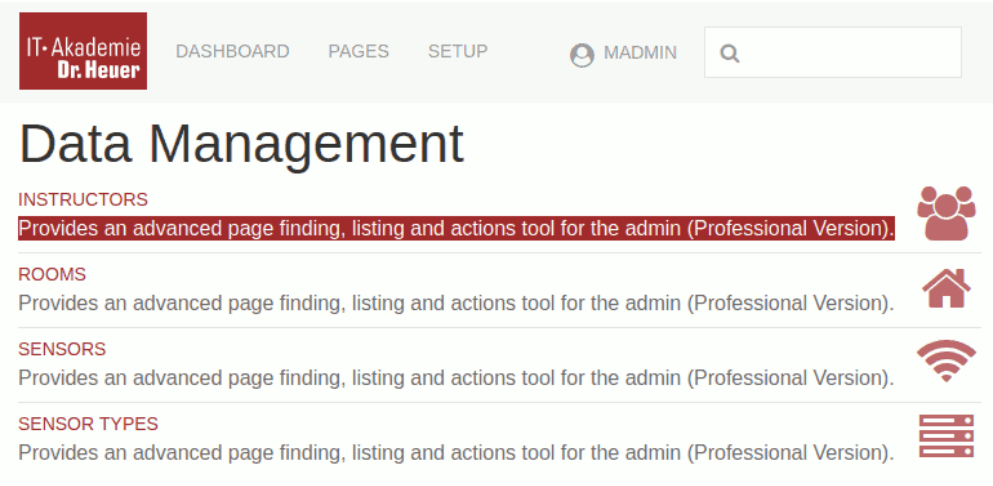
Es wird nur eine grundsätzliche Einrichtung erwartet, den Rest übernimmt der Fachbereich.

3.1.2. Sensoren

Für die Sensoren werden einzelne elektronische Sensor- und Anzeige- Bauteile verwendet die gemeinsam mit einem ESP8266 Mikroprozessorboard auf eine Platine gesteckt oder gelötet werden. Der ESP8266 ist günstig (ca. 4€) und wird in so gut wie allen Bastelprojekten zum Thema Sensorik verwendet Eine Alternative wäre ein original Arduino, der aber wesentlich teurer ist (ab ca. 20€) und auch keine bessere Funktionalität bietet. Ein ESP8266 lässt sich in verschiedenen Programmiersprachen programmieren (MicroPython, C++ , LUA), leider stellte sich schon bei der Recherche für die Sensor Komponenten heraus, das eine vollständige Treiberunterstützung für alle Komponenten nur mit der Arduino IDE (C++) möglich ist.

3.2. Entwurf der Benutzeroberfläche

Processwire kommt von Hause aus mit einer auf dem CSS Framework Uikit basierenden Oberfläche die sich mit Hilfe eines Modular integrierbaren Compilers für die Stylesheet-Sprache „Less“ vollständig konfigurieren und anpassen lässt.



Fürs Mockup habe ich eine Test Applikation farblich etwas angepasst und mit dem Grafikprogramm GIMP noch ein wenig nachbearbeitet. Weitere Mockups finden sich in Anhang Fehler: Verweis nicht gefunden Fehler: Verweis nicht gefunden Seite Fehler: Verweis nicht gefunden

4. Implementierungsphase

Da die Implrementierung im Spiralsystem durchgeführt wurde Werde ich die Iterationen hier Quadrantenweise durchgehen.

4.1. Implementierung des Servers (soll 3h / ist 6h)

4.1.1. Festlegen der Ziele und Alternativen

Raspberry, Raspbian OS installieren mit Apache Webserver mit HTTPs, MariaDB, PHP 8, Processwire CMS, Arduino IDE , X-Windows, VNC Server, SSH, VIM als lokale Entwicklungsumgebung, Setup mit Monitor Maus und Tastatur. (Geplant 3h)

4.1.2. Beurteilung von alternativen und Risikoanalyse

Bei Unverträglichkeiten oder Fehlenden Softwarepaketen ist mit zusätzlichem Zeitaufwand zu rechnen. Deswegen habe ich mich für ein Original Raspbian OS entschieden da Debian einer Recherche nach bei der Installation mehr Probleme bereitet. Hardware Probleme wären auch möglich aber es steht Reservehardware zur Verfügung

4.1.3. Entwicklung und Test

Zuerst Image auf Speicherkarte schreiben und Rechner starten. Die Paketinstallation verlief teilweise etwas zickig. Es musste PHP aus Fremdquellen installiert werden und einige Module(z.b. BcMath) mussten händisch nachinstalliert werden. Die Qualität von Raspbian OS im generellen entspricht nicht der Stabilität von Debian, alles verlief sehr mühsam.

Als abschließender Test wurde Processwire installiert, der Selbsttest bei der Installation und eine manuelle Überprüfung verliefen Problemlos.

Leider zeichnete sich der Rechner durch spontane Instabilitäten aus und blieb gelegentlich einfach stehen. Eine Überprüfung der Speicherkarte ergab einen Defekt, so das alles noch einmal gemacht werden musste.

4.1.4. Analyse und Planung des nächsten Zyklus

Habe Mich bei dem Zeitaufwand deutlich verschätzt so das hier schon 6h anfielen. Leider machte mein Internetprovider wirklich Probleme mit dem heimischen Serverbetrieb so das Ich mich entschlossen habe im nächsten Schritt auf einem Öffentlichen Webserver auszuweichen.

Im nächsten Schritt wird die Applikation erneut installiert, ans Corporate Design angepasst , alle Notwendigen Module werden eingespielt und konfiguriert die vorgesehenen Datenstrukturen in Processwire angelegt und die Formulare und Datentabellen konfiguriert.

4.2. Grundanwendung, Benutzeroberfläche (soll 5h / ist 7h)

4.2.1. Festlegen der Ziele und Alternativen

Installation von Processwire auf einem Webspace, Installlation aller Module Siehe @todo und Implementierung der Datenstrukturen Siehe @ todo in das Pages/Templates System von Processwire. Anpassung ans Corporate Design.

4.2.2. Beurteilung von alternativen und Risikoanalyse

Auf dem Webserver laufen auch weitere Proceswire Anwendungen mit Problemen ist hier eher nicht zu rechnen. Die Anpassung des Designs erfolgt über Less Variablen auch hier sind keine Überraschungen zu erwarten.

4.2.3. Entwicklung und Test

Keine Probleme hier. Schnell einen neuen Webspace eingerichtet. Module über das Admin Backend installieren, alle Einstelllungen Konfigurieren und in geplanten **(2 h)** war alles erledigt.

Übertragen des DB Entwurfs in Processwire war ebenfalls problemlos und völlig nach Plan **(3 h).**

Das war dann der Moment wo ich feststellen musste das für die Anpassung des Designs gar keine Zeit eingeplant ist, damit haben wir eine weitere Stunde Verzögerung. Das wurde direkt gemeinsam mit dem Fachbereich so das die Oberfläche auch gleich abgenommen war **(1 h)**.

Im Gespräch/Test mit dem Fachbereich ergab sich dann auch eine Reihe von Attributen die noch wünschenswert bis notwendig waren. Zum Beispiel kristallisierte sich heraus da ja das Löschen von Sensoren oder Sensor Typen normalerweise nicht möglich ist es zumindest eine Möglichkeit bestehen muss diese zu deaktivieren und in den Auswahlen auszublenden. Auch wäre es toll den C++ Quellcode der für die Verschiedenen Sensorentypen gleich abspeichern zu können, oder sogar automatisiert mit Sensorspezifischen Daten zu befüllen. Dies ist im Entwurf dokumentiert, dort habe ich den ursprünglichen Entwurf und auch das erweiterte Ergebnis eingefügt. Da sich Solche Felder in Processwire sehr schnell erstellen lassen hat mich das ganze inklusive Gespräch nur zusätzliche **1 h** gekostet.

@todo Link zu ER Diags

4.2.4. Analyse und Planung des nächsten Zyklus

Eigentlich nutzen wir ja das Spiralmodell um so etwas zu ermöglichen, aber wenn das öfter passiert explodiert der Zeitplan.

Mittlerweile waren alle Bauteile für die angekommen und wir würden auch Testdaten benötigen um den Empfang zu testen. Also im nächsten Schritt sollen die Prototypen gefertigt und getestet werden.

4.3. Sensoren Bau, Programmierung und Tests (soll 10h / ist 14h)

4.3.1. Festlegen der Ziele und Alternativen

Es sollen die Sensoren anhand der Prototypen zusammengesetzt werden. Die Grundfunktionalität der verschiedenen Sensoren wird über die Anzeige in der Seriellen Konsole der Arduino IDE getestet. Treiber werden gesucht und ausprobiert. Auf Hardware Seite wird ein Prototyp im Gehäuse aufgebaut um zu testen ob alle Komponenten passen.

4.3.2. Beurteilung von alternativen und Risikoanalyse

Hier kann schon einiges schief gehen. Komponenten könnten nicht ins Gehäuse passen, allerdings haben wir zum vorbeugen 2 Größen bestellt. Komponenten könnten Kaputt sein, aber auch hier haben wir dann Ersatz. Defekte Komponenten können unklare Ergebnisse Liefern und auf diese Art Zeit kosten. Zur Vorbeugung war die Recherche zwar richtig gründlich aber letztendlich bleibt das ein Glücksspiel.

4.3.3. Entwicklung und Test

Die Sensoren die Technisch geprüft werden sollten wurden auf Breadboards gesteckt, so das jederzeit Änderungen möglich waren. Gleichzeitig wurde ein Prototyp 3 vollständig zusammengebaut um bei Bedarf noch andere Gehäuse ordern zu können. Auch das Einrichten des Arbeitplatzes war problemfrei und passte noch in die geplanten **(2 h)**

Bilder Siehe Anhang 10.3 Sensoren Bilder @todo Bildbezeichnungen Seite xx

Zuerst wurden die Grundfunktionen des NodeMCU Boards getestet was zuerst überhaupt nicht funktionierte da von 5 Boards 2 Defekt waren und sich einfach nicht beschreiben ließen, bzw. ein defektes WLAN Modul besaßen. Nachdem ich ein funktionierendes Board gefunden hatte ließen sich WLAN und ein HTTP Client recht einfach einrichten.

Zum Testen und erproben der Sensoren und Treiber wurde die Sensoren und auch alle anderen Komponenten separat an einem Node MCU getestet um Interaktionen ausschließen zu können

An dieser Stelle begannen dann die Probleme. Die MQ-135 Luftqualitäts- Sensoren benötigen 48 Stunden Einbrennzeit um danach einigermaßen stabile Werte zu liefern wenn Sie 1 Stunde Zeit hatten sich aufzuheizen . Zusätzlich scheint die ‚Manuelle Kalibrierung nicht sonderlich stabil zu sein . Obendrein unterstützen 2 der 3 Treiber die ich finden Konnte entweder nicht die Analoge oder alternativ nicht die Digitale Funktionalität .

Bei den wesentlich hochwertigeren MH-Z19C Co2 Sensoren schien nur einer von 5 dem Board überhaupt zu antworten und das auch nur instabil. Nach langer Fehlersuche stellte sich heraus das der Sensor mit den 4,5 Volt des Stromanschlusses des Raspi nicht zufrieden ist und erst wenn er an einen USB Hub mit 5,3 V angeschlossen wird zuverlässig funktioniert.

Die Probleme setzten sich fort als sich beim Programmieren der Prototypen auch noch herausstellte das die Treiber zum Teil nur auf bestimmten Pins einwandfrei funktionierten, natürlich gab es Überschneidungen. Dummerweise waren die Pins hart in die Treiber einprogrammiert. Zusätzlich schienen manche Treiber abzustürzen wenn andere ebenfalls installiert waren.

Insgesamt war ich an dieser Stelle weit über die geplanten 8 Stunden und noch keine funktionsfähigen Sensoren in Sicht. **(12 h)**

4.3.4. Analyse und Planung des nächsten Zyklus

An dieser Stelle musste ich abbrechen da auch kein Ziel in Sicht war. Also wurde von mir beschlossen einfach simulierte Sensoren zu nutzen.

4.4. Simulierte Sensoren (soll 0h / ist 3h)

4.4.1. Festlegen der Ziele und Alternativen

Schreiben eines Scripts das zufällige Sensordaten im JSON Format an eine URL sendet.

4.4.2. Beurteilung von alternativen und Risikoanalyse
Simples Handwerk, kein Risiko.

4.4.3. Entwicklung und Test
Zum Testen musste noch eine Klasse erstellt werden die das JSON Payload empfängt und wieder in ein Datenobjekt wandelt. Das ganze kann auf der Console getestet werden und per Cronjob aufgerufen werden.

4.4.4. Analyse und Planung des nächsten Zyklus
Als nächstes Muss die Anwendung weiter fertiggestellt werden.

4.5. Anwendung Programmieren/Code-Review (soll 18 h / ist 16h)

4.5.1. Festlegen der Ziele und Alternativen
Endlich soll es an die Anwendung gehen. Für alle Artefakte werden Module benötigt , sie sich um die Regeln und Logik beim Speichern der Datensätze Kümmern. Ein Modul für Jedes. Module und Hooks für den Empfang der Daten müssen erstellt werden. Module die sich um den Export der Daten kümmern.

4.5.2. Beurteilung von alternativen und Risikoanalyse
Auch hier erwarte ich keine Schwierigkeiten außer das die Zeit knapp wird.

4.5.3. Entwicklung und Test
Alle Module wurden Schrittweise entwickelt und direkt getestet. Leider hat an dieser stelle der Projektbetreuer die Notbremse gezogen und mir empfohlen das Projekt als Teilweisen Fehlschlag auch so zu dokumentieren .

4.5.4. Analyse und Planung des nächsten Zyklus
Projekt Abbruch und ich muss gestehen die Verbleibenden 12 Stunden haben kaum gereicht die Dokumentation völlig abzuschließen.

5. Abnahmephase
Ist wegen des Abbruchs ausgefallen

6. Fazit

6.1. Soll-/Ist-Vergleich

6.1.1. Wurde das Projektziel erreicht
Ja und nein. Ein Anwendungsprototyp haben wir jetzt. Sensoren, Dozenten und alle andere Artefakte können Verwaltet werden, Daten können empfangen werden. Der Webserver läuft und ist bereit zur Aufnahme der Applikation allerdings befindet sich die Anwendung immer noch auf dem Webspace. Für die Sensorelemente liegt jetzt ein klarer Bauplan vor, aber noch keine Programmierung, nur einige Tests. Dafür gibt es jetzt eine Klasse zum Simulieren von Sensoren die zuverlässig Daten liefert. Das Dashboard ist zwar implementiert aber noch nicht fertiggestellt. Warnungen können Versand werden.In Code Dokumentation ist wirklich ausführlich aber die Zeit zum Erstellen mit PHP Dokumentor fehlte dann doch.

Fazit

Der Hauptgrund das es so Chaotisch geworden ist, dürfte der sein, das der Author sich auf etwas eingelassen hat was eigentlich 3 Projekte beinhaltet und zusätzlich mit Dokumentation total unerfahren ist. Nach etlichen Problemen und Verzögerungen hat er sich dann fast völlig verzettelt. Auch die Unsicherheit wie denn nun das Spiralmodell in der Dokumentation zu handhaben ist hat natürlich dazu beigetragen.

6.1.2. Projekt Planung

6.1.2.1.AnalysePhase (Soll 12 h / Ist 13 h)

Hier verlief alles nach Plan bis auf die Tatsache, das der Fachbereich das Pflichtenheft aufgrund von Corona nicht erstellten konnte, was mich eine extra Stunde gekostet hat.

6.1.2.2.Entwurfsphase (Soll 12 h / Ist 12 h)

Eigentlich soweit gut verlaufen, nur das ich mich mit der Trennung von DV-Konzept und Pflichtehheft wirklich schwer getan habe.

6.1.2.3.Implementierungsphase

An dieser Stelle ging dann fast alle schief was schief gehen konnte, der Puffer war schon bei der Serverinstallation aufgebraucht, und danach wurde es nicht besser. Ab hier ist dann wie man in der Implementierung sehen kann alles vollends aus dem Ruder gelaufen. So das Dann abgebrochen werden musste um überhaupt noch Dokumentieren zu können, und auch die Zeit war schon extrem knapp

6.1.3. Projekt Nutzen

Wenn man nun fragt ob das Projekt nützlich Ergebnisse geliefert hat muss man das eindeutig bejahen.

- Schlüssiges Konzept für die Sensoren
- Die Erkenntnis das ein MQTT Broker benötigt wird um Kompatibilität mit anderen IoT Geräten zu herzustellen. Damit könnte man dann letztendlich die gesamte Haustechnik Kontrollieren
- Eine Anwedung die zwar nicht fertig ist , aber Ausbaufähig. Etwa noch 20 Stunden zur funktionalen Fertigstellung.
- Es wurden jede Menge unbekannter Problemstellen gefunden die jetzt nicht mehr unbekannt sind . (Inkompatibilitäten oder fehlende Funktionalitäten in Treibern für Arduino, Sensoren die nicht wirklich taugen MQ-135, Probleme mit Raspian OS)

6.2. Lessons Learned

Murphy hat recht ;-) Es ist wirklich viel daneben gegangen bzw. hat viel zu lange gedauert. Der Autor hat sich völlig verzettelt.

Es war sehr aufschlussreich sich einmal wirklich mit der Planung und dem Umgebungen eines Projektes zu befassen, leider hatte ich trotz meiner Berufserfahrung bisher eher nur mit der puren Programmierung und weniger mit einer wirklichen Projektplanung zu tun. Aber gelernt habe ich hier wirklich viel, deutlich mehr als wenn ich es mit einem Bilderbuch Projekt zu tun gehabt hätte !

- Beim nächsten mal würde ich mich definitiv auf die Webanwendung beschränken, mit simulierten Testsensoren. An mehreren Baustellen(Server, Hardware und Software) zu arbeiten ist einfach zu viel.
- Beim Planen der Zeiten ist wesentlich mehr Vorbereitung nötig eventuell auch einige kleinere Tests.

Fazit

- Das Spiralmodell ist im Rahmen einer üblichen Projektplanung wirklich schwierig zu handhaben, die Iterationen bilden eigentlich vollständige Miniprojekte und genauso müssten sie auch behandelt werden. Auch ist hier alles weit weniger Planbar. Wobei ich völlig unsicher bin wie eine solche Projektarbeit dann bewertet würde ?
- Besser kleinere Projekte aber dafür gründlicher Planen.
- Beim Spiralmodell unbedingt mehr Zeit zum Testen einbauen, da das Testen der einzelnen Iterationen wesentlich mehr Zeit benötigt. Mitunter auch weil dabei Ideen aufkommen, die im nächsten Schritt umgesetzt werden.

6.3. Ausblick

Die Webanwendung wird mit hoher Wahrscheinlichkeit innerhalb eines Nebenjobs ausgebaut werden. Daraus soll letztendlich ein vermarktbares Produkt entstehen. Dafür müssen noch viele kleine Erweiterungen vorgenommen werden z.B Code Verwaltung für die einzelnen Sensoren/Sensortypen, Anbindung an einen MQTT Broker und vieles mehr.

Aus diesem Projekt werden wahrscheinlich weitere Projekte hervorgehen.

- Erweiterung der Sensoren mit einem ESP32 NodeMCU der kaum teurer ist aber viel mehr Speicher und Leistung mitbringt, so das der Sensor ein vollständiges Projekt werden kann. Verwaltung und Einrichtung über Bluetooth oder eigenen Webserver/Webseite. Kommunikation mit verschiedenen IoT Schnittstellen. Das Ganze muss natürlich auch abgesichert werden, vielleicht eigenes Logging, ein passendes gedrucktes Gehäuse. Ich bin mir sicher da geht einiges.
- Genauso ist auf der Serverseite einiges zu tun. Der Webserver kann Leistungsorientiert eingerichtet werden. Die Architektur muss weiter abgesichert werden. Eine Firewall sollte eingerichtet werden. Ein MQTT Broker muss eingerichtet werden, die Einstellungen für einen Broker wie Eclipse Mosquitto können auch über die Webanwendung verwaltet werden und auch hier ist ein riesiges Potential vorhanden.

ANHANG

7. Lastenheft

7.1. Allgemeines

7.1.1. Zweck des Dokuments

Das vorliegende Lastenheft dient als Basis für die Ausschreibung und Vertragsgestaltung und bildet somit die Vorgabe für die Angebotserstellung. Es enthält alle an das zu entwickelnde Produkt gestellten Anforderungen, funktionale sowie nicht-funktionale. Mit den Anforderungen werden die Rahmenbedingungen für die Entwicklung festgelegt, die vom Auftragnehmer im Pflichtenheft detailliert ausgestaltet werden. Kommt es zwischen Auftragnehmer und Auftraggeber zu einem Vertragsabschluss, ist das bestehende Lastenheft rechtlich bindend.

7.1.2. Ausgangssituation

Die Dr. Heuer IT-Akademie ist eine Schulungseinrichtung für IT-Berufe mit ca. 160 Schülern in 12 Schulungsräumen. Spätestens seit Corona ist eine Überwachung der Raumluftqualität wünschenswert. Auch wäre es aus Energiespargründen durchaus sinnvoll, diese Überwachung auch auf die Temperatur und möglicherweise andere Parameter auszuweiten.

Zurzeit ist überhaupt kein Sensornetz vorhanden. Es werden einfach nur periodische Messungen mit Handgeräten durchgeführt. Aus den Messwerten werden unter Berücksichtigung der Raumgröße und aktueller Belegung mit Hilfe von statistischen Verfahren, die Zeit Intervalle festgelegt, die für eine ausreichende Belüftung notwendig sind. Dieses Verfahren ist zwar grundsätzlich funktional, aber es erfordert regelmäßigen Personalaufwand für Messungen, die Übernahme der Messwerte und Berechnungen.

Vorhanden sind die Option, ein akademieweites WLAN aufzuspannen, Kabelschächte unter den Fensterbänken (LAN und Strom) und einige bodennahe Steckdosen in den Räumen. Zu beachten ist, dass die Sensoren auf eine Stromversorgung angewiesen sind und die Netzwerkswitches nicht über Power over Ethernet (POE) verfügen.

Als zentraler Server soll ein Raspberry Pi dienen, der auch schon vorhanden, aber nicht eingerichtet ist. Alle weiteren Hardware- Komponenten müssen neu angeschafft werden. Die Netzwerk-Infrastruktur, die Konfiguration des Routers sowie das Zur- Verfügung-Stellen der Kommunikationsmittel (Webhook, SMS Portal, SMTP) und auch die Beschaffung der Hardware nach Festlegung der Komponenten obliegen der Fachabteilung und sind nicht Teil dieses Projekts.

Nach Vorgaben der IT-Akademie soll sich die Umsetzung des Projektes an den Phasen des Spiralmodells (nach Boehm) orientieren. Aufgrund der Vielzahl der Komponenten (Sensoren, Server, Netzwerk, Webanwendung,...) ist mit Problemen zu rechnen, die sich im Wasserfallmodell nur sehr schwer im Voraus abbilden lassen. Auch im Bereich der User Interfaces ist mit mehreren iterativen Schritten zu rechnen. Als Versionsverwaltung ist GIT vorgegeben.

7.1.3. Was soll am Ende des Projektes erreicht sein?

Am Ende des Projektes steht der Prototyp eines Sensornetzes sowie eine Datenbank, die automatisiert alle Daten der Sensoren ausliest bzw. empfängt und speichert. Die Anzeige der Daten sowie die Verwaltung des Sensornetzes soll über eine Webanwendung gelöst werden, die darüber hinaus die Möglichkeit bietet, den Verlauf der Sensordaten grafisch darzustellen und Datensätze zu exportieren.

7.2. Funktionale Anforderungen

7.2.1. Generelle Anforderungen

1. Drei Sensor-Prototypen in verschiedenen Preisklassen müssen zusammengestellt werden.
2. Die Sensoren müssen den aktuellen CO^2-Wert in den Räumen messen.
3. Die Daten müssen zeitgesteuert an einen zentralen Server übertragen werden. Die Art der Übertragung ist noch auszuarbeiten.
4. Das Konzept der Sensoren sollte erweiterbar sein, um weitere Messdaten erfassen zu können.
5. Die Daten müssen zusammen mit einem Zeitstempel in einer Datenbank gespeichert werden. Die Art des Datenbanksystems muss festgelegt werden.
6. Wenn möglich, können neue Sensoren automatisch im Netz gefunden werden.
7. Es sollen automatische Vorgänge wie Archivierung oder regelmäßiger Export/Report der Daten ermöglicht werden.
8. Weiterhin muss bei kritischen Werten ein Alarm ausgelöst werden der direkt an den Dozenten im Schulungsraum gesendet wird, z.B. per SMS, Mail oder Microsoft Teams.
9. Es muss ein Alarmsystem und Konzept erstellt werden, welches bei Überschreiten der Toleranzschwellen automatisiert Alarme an die zuständigen Stellen versendet. (Email, SMS, eventuell MS Teams)

7.2.2. Anforderungen an die Webanwendung

1. Es muss eine vollständige Authentifizierung implementiert sein.
2. Sie sollte sich um die Überwachung des Sensornetzes kümmern.
3. Sie muss die Daten grafisch (Kurven und Diagramme) und statistisch (Reports) aufbereiten und anzeigen können.
4. Weiterhin muss sie den Export der Daten und der Reports ermöglichen.
5. Auch muss sie eine Möglichkeit bieten, neue Sensoren, Mitarbeiter und Räume einzupflegen und zu verwalten.
6. Es können eventuell noch weitere Autorisierungsstufen implementiert werden.

7.2.3. Administrationsfunktionen

Es sind 3 Benutzerstufen vorgesehen.

1. Jeder Dozent sollte in der Lage sein Warnungen zu empfangen und eine generelle Übersicht einzusehen. Dazu ist kein Login erforderlich.
2. Die Managementstufe sind Mitarbeiter die damit beauftragt sind die Applikation zu Verwalten. Diese Mitarbeiter dürfen Räume, Dozenten, Sensoren verwalten und anlegen. Weiterhin dürfen sie alle Verlaufsdaten einsehen und exportieren und haben ein Dashboard das aktuelle Zustände, Warnungen und Verläufe Grafisch darstellt.
3. Der Administrator verwaltet die gesamte Webanwendung und hat damit Zugriff auf z.B. die Datenstrukturen, installierte Module und einfach Alles.

7.2.4. Konnektivität

Die gesamte Kommunikation der Sensoren mit dem Web-Server der Akademie muss nicht unbedingt Verschlüsselt erfolgen, da die Daten nicht personenbezogen sind und generell keiner Geheimhaltung unterliegen. Viele generell verfügbare IoT Geräte verfügen ebenfallls über keine Verschlüsselung.

Bei späteren Produkteinsatz, wäre es von Vorteil die Option zu haben die Daten auch verschlüsselt zu übertragen, für den Prototypen wäre dies schön, aber ist nicht erforderlich.

7.3. Nicht funktionale Anforderungen

7.3.1. Datenschutz

1. Sämtliche Zugangsdaten und Schlüssel müssen verschlüsselt gespeichert werden.
2. Die Menge an personenbezogenen Daten beschränkt sich auf Namen und Kommunikationskanäle(Teams), die sowieso schon von der Dr. Heuer GmbH zur Verfügung gestellt werden und auch Rechtlich schon geregelt sind. Dementsprechend besteht hier kein weitere Bedarf an Regelung.

7.3.2. Zu berücksichtigende Einschränkungen.

1. Die Anwendung sollte möglichst der Corporate Identity der Institution angepasst werden.
2. Die Sensoren sollen unter wirtschaftlichen Gesichtspunkten ausgewählt werden. Teure Lizenzgebühren, hohe Hardwarekosten sollen vermieden werden. Die Geräte sollen gut zu warten und leicht zu reparieren sein.
3. Als Basis ist ein fertiges Webframework zu verwenden, welches noch ausgewählt werden muss. Die Funktionalität wird dann als Modul/Anwendung in diesem Framework erstellt. Die Auswahl des Datenbanksystems wird sich dann auch nach diesem Framework richten.
4. Die räumliche Situation ist nicht ganz einfach, da die Sensoren natürlich nicht direkt ans Fenster gesetzt werden können. Hier muss eine Lösung erarbeitet werden.

7.3.3. Qualitätsanforderungen

Bei der Anwendung sollten die folgenden Kriterien besonders in Vordergrund stehen: Alle Daten sollen Übersichtlich dargestellt werden.. Die Anwendung soll zuverlässig über lange Zeit funktionieren Sie soll robust sein und benutzerfreundlich

8. Pflichtenheft (Anhang)

8.1. Projektumfeld

Die Dr. Heuer IT-Akademie ist eine Schulungseinrichtung für IT-Berufe mit ca. 160 Schülern in 12 Schulungsräumen an 2 Standorten. Die IT-Akademie bietet seit 2005 an Standort Bochum und seit 2013 am Standort Düsseldorf, Umschulungen und Weiterbildungen im IT-Bereich an.

Die Dr. Heuer GmbH ist auch gleichzeitig Auftraggeber da diese Arbeit als Weiterbildung/Externenprüfung durchgeführt wird.

8.2. Projektziel

Spätestens seit Corona hat sich herausgestellt das eine Überwachung der Raumluftqualität in Schulungsräumen und Büros angebracht ist, auch wäre es aus Energiespargründen durchaus sinnvoll, diese Überwachung möglicherweise auf andere Parameter wie Temperatur auszuweiten. Das Ziel ist dementsprechend der Entwurf und Bau eines Prototyps zur

Überwachung der Luftqualität in den Schulungsräumen. Das Kernstück soll hierbei eine Webanwendung darstellen, die eine Verwaltung des WLAN-basierten Sensornetzes und der zugehörigen Messdaten anbietet.

8.3. Projektbegründung

Derzeit finden die Messungen stichprobenartig und manuell statt, das ist zwar grundsätzlich funktional, aber es erfordert regelmäßigen Personalaufwand und deckt auch nicht wirklich den Tagesverlauf ab. Eine gekaufte Lösung ist nach ersten Recherchen sehr kostenintensiv, so das eine eigene Lösung praktikabel erschient.

8.4. Projektschnittstellen

Grundsätzlich sind folgende Schnittstellen zu beachten

- Die als Auftraggeber genehmigt das Projekt und stellt die Mittel zur Verfügung und nimmt die Anwendung ab.
- Einige privilegierte Mitarbeiter der IT-Akademie verwalten das Sensornetz und werden benachrichtigt wenn Sollwerte deutlich abweichen.
- Die Dozenten der Dr. Heuer GmbH werden vom System benachrichtigt wenn Sensorwerte nicht im soll Bereich liegen
- Die Hauptanwendung empfängt Daten von den einzelnen Raumsensoren und speichert diese in einer Datenbank

8.5. Projektabgrenzung

Folgende Punkte sind nicht Teil des Projektes. Da sie völlig den rahmen von 80 Stunden sprengen würden.

- Die detaillierte Absicherung des Webservers.
- Zusammenbau weiterer Sensoren außer der 3 Testgeräte.
- Einrichten eines vom normalen Betrieb der Akademie abgegrenzten WLAN
- Konfiguration des Routers (DHCP und dergleichen) und der Switches zu aufspannen eines VLAN
- Zur Verfügung stellen von Webhooks für MS Teams, SMS Portalen und SMTP Server zum Mailversand
- Beschaffung der Hardware obliegt der Fachabteilung.

8.6. Ist-Analyse

Zurzeit ist überhaupt kein Sensornetz vorhanden. Es werden einfach nur periodische Messungen mit Handgeräten durchgeführt. Aus den Messwerten werden unter Berücksichtigung der Raumgröße und aktueller Belegung mit Hilfe von statistischen Verfahren, die Zeit Intervalle festgelegt, die für eine ausreichende Belüftung notwendig sind. Dieses Verfahren ist zwar grundsätzlich funktional, aber es erfordert regelmäßigen Personalaufwand für Messungen, die Übernahme der Messwerte und Berechnungen.

Vorhanden sind die Option, ein akademieweites WLAN aufzuspannen, Kabelschächte unter den Fensterbänken (LAN und Strom) und einige bodennahe Steckdosen in den Räumen. Zu beachten ist, dass die Sensoren auf eine Stromversorgung angewiesen sind und die Netzwerkswitches nicht über Power over Ethernet (POE) verfügen.

Als zentraler Server soll ein Raspberry Pi dienen, der auch schon vorhanden, aber nicht eingerichtet ist. Alle weiteren Hardware- Komponenten müssen neu angeschafft werden.

8.7. Anwendungsfälle

Dieses Projekt beinhaltet 3 Haupt Anwendungsfälle

1. Ein jeder einem Klassenraum zugeordneter Dozent wird gewarnt wenn der Co2-gehalt der Atemluft im Raum über die festgelegten Grenzen steigt. Des Weiteren hat er die Möglichkeit die allgemeine Übersicht (Dashboard) über alle Räume ohne Login aufzurufen.
2. Ein als Benutzer eingetragener „Manager“ wird benachrichtigt wenn der zuständige Dozent die Warnung ignoriert und der Co2 Gehalt über den 2. Grenzwert steigt. Auch kann er die Übersicht aufrufen. Weiterhin kann ein Manager sich anmelden und sämtliche Objekte (Dozenten,Sensoren, Räume…) erstellen und verwalten. Löschen kann er nicht, nur deaktivieren. Darüber hinaus kann er Daten exportieren.
3. Der Administrator der Anwendung wird nicht benachrichtigt. Er kann allerdings ebenfalls alles was der Manager kann und darüber hinaus die gesamte Anwendung verwalten, erweitern und verändern.

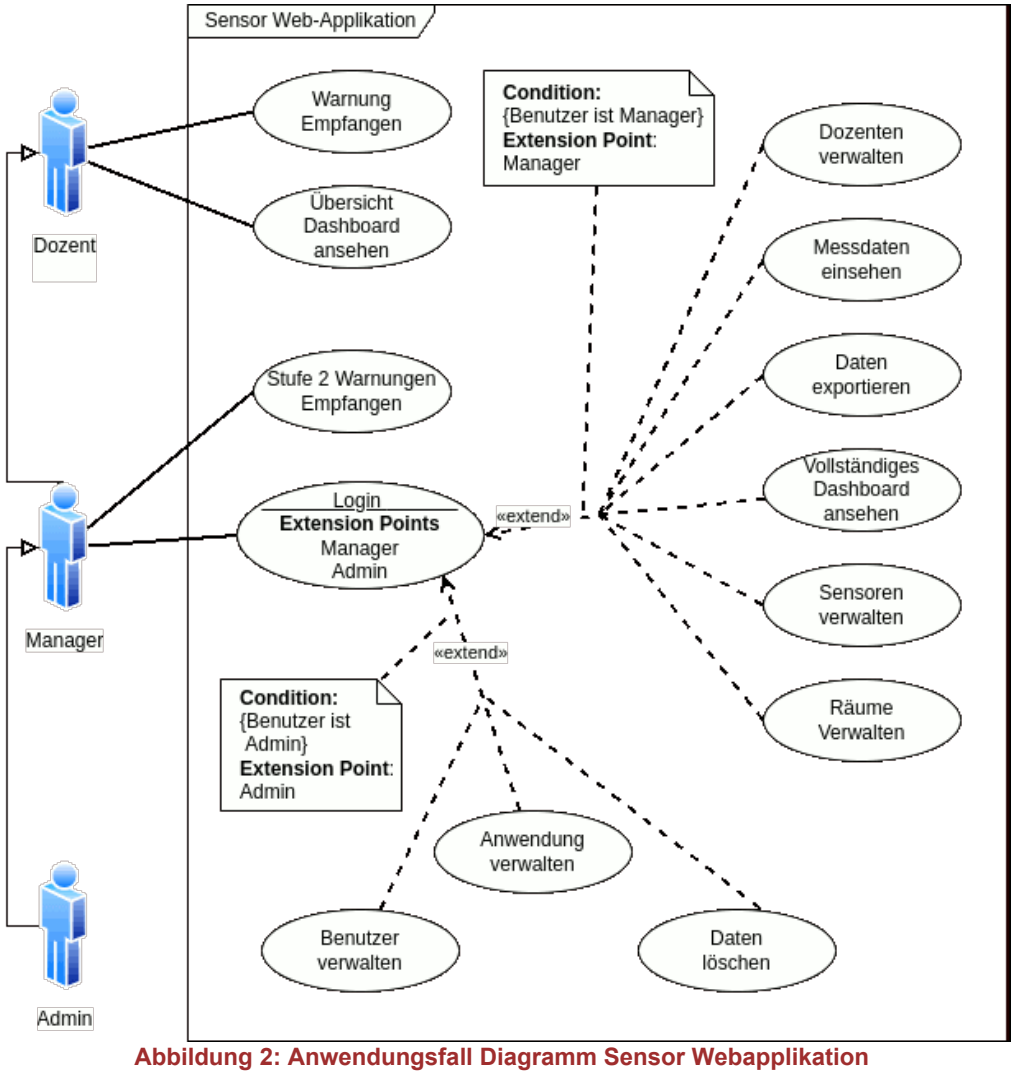


Abbildung 2: Anwendungsfall Diagramm Sensor Webapplikation

8.8. Erweiterte Qualitätsanforderungen

Da es sich um einen Prototypen handelt sind die Qualitätsanforderungen nicht übermäßig hoch.

- Übersichtliche Darstellung der Daten.
- Datentabellen sollten sortierbar und durchsuchbar sein.
- Die Anwendung soll zuverlässig über lange Zeit funktionieren und robust sein.
- Die Optik sollte dem Corporate Design entsprechen
- Programmcode sollte unbedingt Objektorientiert sein
- Der Code sollte den Vorgaben des Frameworks entsprechen.
- In Code Kommentare werden in ausreichender Form erwartet.
- Die Sensoren werden in einem fortgesetzten Projekt noch weiterentwickelt, insofern sollen diese nur generell funktionieren und Testdaten liefern.
- Die Datenübertragung sollte wenn möglich verschlüsselt erfolgen.

8.9. Sensoren

Grundsätzlich sollen die Sensoren auf einem einfachen Entwicklerboard z.b. NodeMCU (ESP8266) diese sind günstig, haben analoge und Digitale Ausgänge. Welche genauen Boards sowie Sensorkomponenten muss noch in einer Recherche geklärt werden. Auch muss das ganze dann unter wirtschaftlichen Gesichtspunkten mit käuflichen Systemen verglichen werden

9. DV – Konzept / Entwurf

9.1. Websserver

Beim Webserver handelt es sich um einen Raspberry PI Minirechner mit 4 GB RAM und einem ARM Prozessor, dieser ist Vorgegeben weil schon vorhanden.

9.1.1. Zu installierende Features

- Raspian OS
  (Schlanke Linux Distribution, speziell für den Raspberry PI, basiert auf Debian)
- X-Window System
- Arduino IDE (C++ Programmierung Sensoren)
- Apache 2 Webserver
  - Mit Unterstützung für Mod Rewrite, .htaccess
- PHP Version 7+ Skriptsprache mit „PHP short tags“ aktiviert .
  - PHP Modul GD2 oder Image Magic Grafik Bibliothek
  - PHP Modul BCMATH
  - PHP Modul mbstring
- Real VNC und SSH zur Fernwartung des Servers.
- OpenSSL zur Verschlüsselung.
- VIM (VI Improved) Als Entwicklungsumgebung

Das Betriebssystem kann frei heruntergeladen werden und wird dann auf eine SSD Speicherkarte geschrieben, die als Systempartition für den Rechner dient.

Die Pakete werden über den Debian Paketmanager installiert was dann z.b. so aussieht.

```
sudo apt update
sudo apt install apache2
```

Die Pakete werden danach über Ihre jeweiligen Konfigurationsdateien konfiguriert und die Dienste neu gestartet. z.B.

```
sudo vim /etc/apache2/sites-available/example.com.conf
sudo systemctl restart apache2
```

Das ganze sollte eigentlich relativ schnell und schmerzlos gehen zumal keine Sonderanforderungen wie z.B. Firewall Konfiguration gefordert sind.

9.1.2. Testen

Zum überprüfen der Grundfunktionalität wird einfach der Installer von Processwire gestartet und das CMS/Framwork installiert.

9.2. Sensoren

Die Sensoren basieren auf einem NodeMCU board. Im Grunde ist ein NodeMCU-Entwicklungsboard nichts anderes als ein ESP8266 Mikrocontroller der um eine Usb-Schnittstelle (Uart/USB-Wandler auf dem Board) und eine Spannungsstabilisierung (ESP-Module arbeiten mit 3,3V) erweitert wurde. Zudem wurde die Pinbelegung des NodeMCU vereinheitlicht und ähnlich wie bei den Arduinos neu durchnummeriert. Weiterhin wurde die Programmierung erheblich vereinfacht, da jetzt auch die zwei Handshake-Leitungen per USB genutzt werden um das Board jederzeit per Arduino programmieren zu können.

Aufgrund zahlreicher Digitaler Ein und Ausgänge eignet sich das Board hervorragend um diverse Sensoren und Anzeigeelemente anzuschließen. Zusätzlich verfügt das Board über eine WLAN Antenne.

Das Board wird über die Arduino IDE in C++ programmiert. Und für die allermeisten Sensoren sind auch vorgefertigte Klassen(Treiber) verfügbar.

- Sensoren müssen recherchiert und ausprobiert werden.
- Grundsätzlich Verbindung zum WLAN muss aufgebaut werden.
- Das Board muss Https Requests mit einem Json Payload an einen Server Senden können.

Leider ist das alles im freien Fall da unklar ist welche Sensoren letztendlich genommen werden. Folgende Daten sollen Übertragen werden. Wobei **tempValue und temperatur optional sind**

- **title(string)**: Name des Sensors
- **hash(sting)**: Ein Hash zur Überprüfung der Identität des Sensors
- **tempValue(double)**: Der roh Messwert den der Sensor ausgibt, für mögliche Korrektur Berechnungen
- **temperature(double)**: Temperatur in Grad Celsius

- **co2Ppm(integer)**: CO^2 in ppm(Parts per Million)
- **co2Value(double)**: CO^2 roh Messwert für spätere Korrekturen
- ipAddresse(String): IP Adresse des Sensors

Was auch schon bekannt ist wie das JSON Payload ausschauen soll .

```
{"title": "Sensor 1",
 "hash": "Checkmich ich bin der hash",
 "tempValue": 668.32999999999993,
 "temperature": 37,
 "co2Ppm": 425,
 "co2Value": 1277.49,
 "ipAddress": "127.0.0.33"}
```

9.2.1.

9.2.2. Sensor Platzierung

Zuerst musste einmal geklärt werden wo die Sensoren überhaupt Platziert werden konnten damit festgelegt werden konnte ob die Datenübertragung per Lan oder Wlan stattfinden konnte.

Büros und Klassenräume in der DHG sind praktisch immer gleich aufgebaut. Eine große Stromleiste unter den Fenstern versorgt alle Arbeitsplätze mit Strom und LAN Anschlüssen. Wireless Access Points versorgen weiterhin alle Räume mit einem Akademieweiten WLAN. Die LAN Switches sind leider nicht in der Lage POE(Power over Ethernet zu liefern).

- Wie wir schon vorher festgestellt haben ist die optimale Höhe irgendwo zwischen Knie und Gesichtshöhe einer sitzenden Person. Damit fällt die Decke als Platz auch schon aus.
- Eine Positionierung am Fenster ist nicht sinnvoll, da beim Lüften sofort wieder optimale Luftqualität gegeben ist obwohl die Luft noch nicht den Rest des Raumes erreicht hat.
- An den Innenseiten der Räume befinden sich mehrere Steckdosen auf Kniehöhe, die als Stromversorgung dienen könnten allerdings keine Anschlüsse für LAN. Die Steckdosen dürfen allerdings nicht vollständig blockiert werden, da Sie vom Reinigungspersonal benötigt werden.

Daraus ergibt sich eigentlich zwangsläufig folgende Konstellation:
Für die Stromversorgung werden die Steckdosen an den Rauminnenseiten genutzt und die Datenübertragung findet über WLAN statt. Die Sensoren werden neben oder an den Steckdosen montiert und sind damit auch ungefähr in der richtigen Höhe für die Messungen.

9.2.3. Welche Grenzwerte sind sinnvoll?

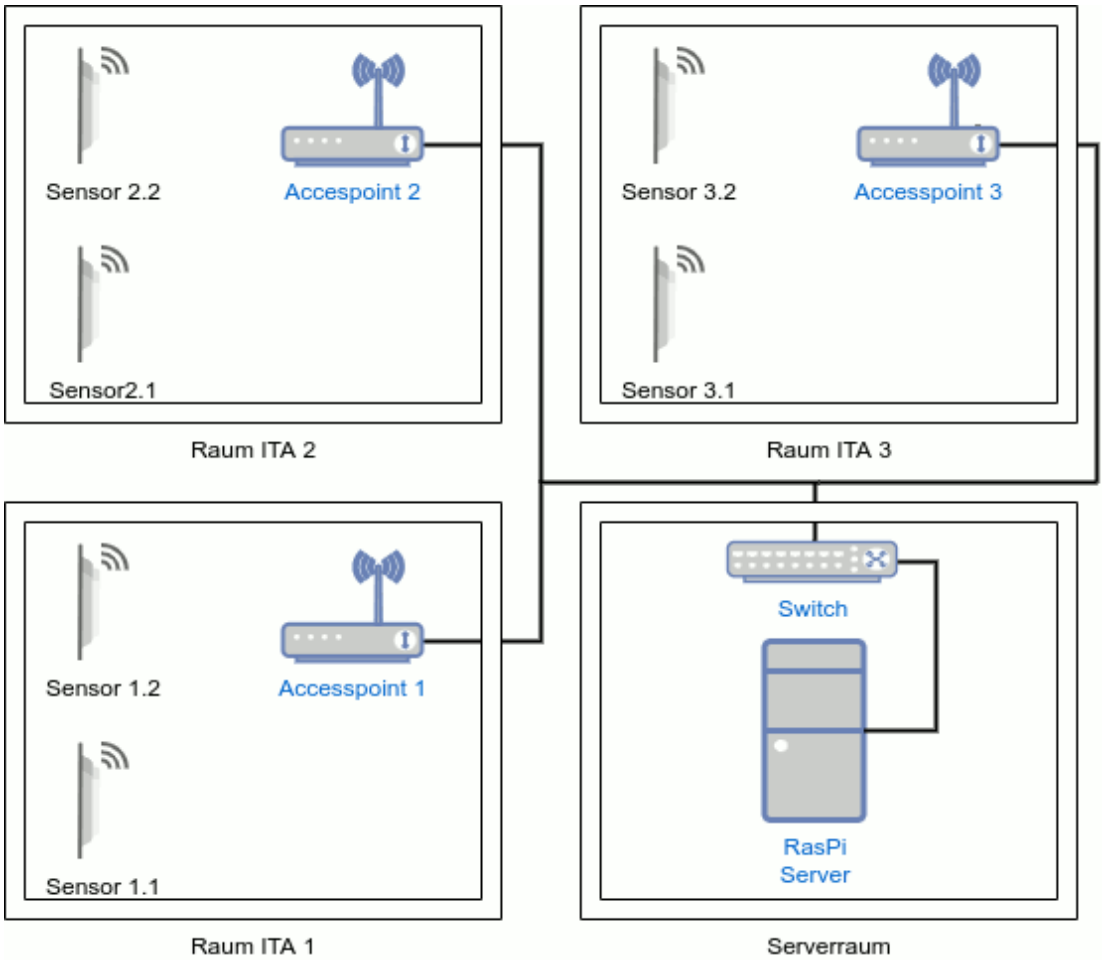
Das Bundesamt für Umweltschutz hat in seiner Fortbildung für den Öffentlichen Gesundheitsdienst 2019 (Anforderungen an Lüftungskonzeptionen in Bildungsgebäuden – Aktuelle Empfehlungen https://www.bfr.bund.de/cm/343/anforderungen-an-lueftungskonzeptionen-in-bildungsgebaeuden-aktuelle-empfehlungen.pdf) Empfohlen in Innenräumen mit dem Co2 Wert unter 1000 ppm zu bleiben. Kurzfristig wären auch Werte bis 1500 ppm vertretbar, dann müsse aber unbedingt gehandelt werden. Dementsprechend wurden die Grenzwerte für diese Projekt mit 1000 ppm und 1500 ppm festgelegt.

9.2.4. Testen

Da die Arduino IDE eine eingebaute Serielle Konsole besitzt kann die Funktionalität der Sensoren und deren Treiber problemlos über die IDE getestet werden. Für das testen der Datenübertragung sollte erst einmal die Webanwendung zur Verfügung stehen.

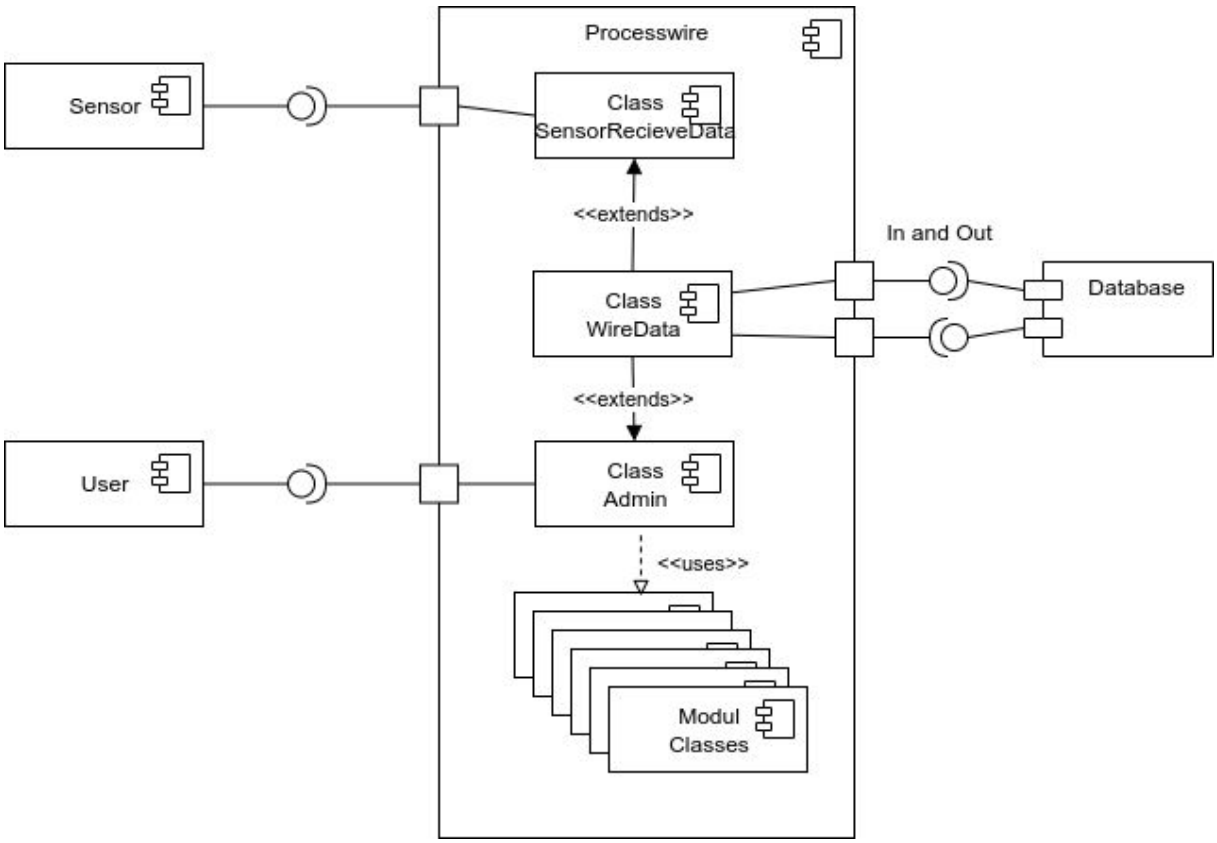
9.3. Physischer Aufbau

Nachdem die Übertragungswege und die Positionierung der Sensoren geklärt waren , ergab sich ein relativ einfacher Physischer Aufbau.



Der Server wird im Rechnerraum an die Zentrale Netzarchitektur angebunden und die Sensoren kommunizieren über die vorhandenen AccessPoints mit dem Server.

9.3.1. Komponenten



9.4. Wie sind die Anforderungen zu lösen

Wie in den Projektentscheidunge zu Lesen ist wurde sich für Processwire und MariaDB entschieden . Allle Lösungsansätze beziehen sich darauf.

9.4.1. Generelle Anforderungen

9.4.1.1.Sensoren

Die Bauteile für die Sensoren werden im Internet Recherchiert und zusammengestellt. Darüber wird eine Aufstellung und Kostenrechnung gemacht. In der Testphase werden die Komponenten auf „Breadboards“ zusammengesteckt. Bei den allermeisten Komponenten werden Treiber gleich mitgeliefert. Die Programmierung findet in C++ über eine Arduino IDE statt.

Durch die Verwendung von Programmierbaren Entwicklungsboards ist das Konzept immer und beliebig erweiterbar.

9.4.1.2.Datenübertragung

Ein Konzept ist hier @todo Link

Zeitstempel werden in der Datenbank automatisch gesetzt.

9.4.1.3.Alarmkonzept

Das Alarm Konzept sieht folgender Maßen aus

Es gibt 2 Grenzwerte die in der Applikation eingestellt werden können. In der Stufe „Warnung wird eine Nachricht an den für diesen Raum zuständigen Dozenten Versendet, in der Stufe Alarm erden zusätzlich Warnungen an alle Systemverwalter(Manager) Versand.

Der Versand wird im Speicher Modul für die einzelnen Datensätze getriggert, dazu ist ein weiteres Modul zu erstellen, das in der Lage ist diesen Versand zu Übernehmen. Dabei sollte sich an Bestehen den Klasse/Modulen wie z.b. „WireMail“ orientiert werden.

Am besten wäre ein Modul das die Anfragen bündelt und über die festgelegten Kanäle versendet. Die Kanäle sollte jeweils wieder ein Modul sein.

9.4.1.4.Weiteres

Zum Export von Daten muss ein kleines Modul geschrieben werden, das dann in die Tabellenansicht einhängen kann. Dafür ist eine Schnittstelle vorhanden.

9.4.2. Anforderungen an die Webanwendung

9.4.2.1.Authentifizierung

Die Authentifizierung wird durch die von Processwire mitgebrachten Module gelöst. Diese sind leicht konfigurierbar und bringen auch Funktionen wie „Passwort vergessen“ mit.

9.4.2.2.Grafische Darstellung

Die Grafische Darstellung und Abbildung der Daten wird mit dem Dashboard Modul realisiert, diese stellt einen konfigurierbaren Rahmen und eine Javascript Anzeige Bibliothek zur Verfügung. Alle Anzeige Elemente werden dann als einzelne kleine Mini Module definiert.

9.4.2.3.Sensornetz überwachen

Beim Speichern eines Datensatzes wird automatisch auch der Sensor mit seinen aktuellsten Werten aktualisiert, dadurch ist immer ein Zeitstempel der Letzten Aktualisierung vorhanden. Wir können dann per Cronscript alle Sensoren alle Sensoren deaktivieren die zu lange Offline waren. Genauso können sie wieder aktiviert werden wenn erneut Daten eingehen.

9.4.2.4.Verwaltung von Sensoren räumen Dozenten ….

Für diese Basis CRUD + Suche Funktionalität bieter Processwire eine Vielzahl von Modulen und Erweiterungen die bei der Erstellung behilflich sind . Alle dieser Erweiterungen sind selbst auch wieder Modular zu erweitern so jedwede Benutzerwünsche abgedeckt werden können.

9.4.3. Administrationsfunktionen

Der Administrator verwaltet die gesamte Webanwendung und hat damit Zugriff auf z.B. die Datenstrukturen, installierte Module und einfach Alles.

Jede weitere Benutzerebene kann durch Hooks(Module) realisiert werden die dich zum Beispiel in die anzeige des Backends einhängen und dort bestimmte Bereiche nicht anzeigen.

Darüber hinaus kann der Zugriff auf die Daten bis hin zum einzelnen Feld konfiguriert werden.

Mit der Kombination dieser beiden Methoden können beliebig viele Benutzerebenen implementiert werden

9.4.4. Konnektivität

Die gesamte Kommunikation der Sensoren mit dem Web-Server der Akademie muss nicht unbedingt Verschlüsselt erfolgen, da die Daten nicht personenbezogen sind und generell kei-

ner Geheimhaltung unterliegen. Viele generell verfügbare IoT Geräte verfügen ebenfallls über keine Verschlüsselung.

Bei späteren Produkteinsatz, wäre es von Vorteil die Option zu haben die Daten auch verschlüsselt zu übertragen, für den Prototypen wäre dies schön, aber ist nicht erforderlich.

9.5. Nicht funktionale Anforderungen

9.5.1. Datenschutz

Einzelne Felder können ebenfalls per Hook/Modul beim speichern und Laden verändert werden damit ist eine Ver- und Entschlüsselung möglich.

Was die Datenmenge betrifft ist das eine einfache Sache der Festlegung.

Processwire ist eines der sichersten QMS/Framewors mit unter deswegen weil es sich um viele Sicherheitsprobleme selbständig kümmert (SQL Injections, CSRF Attacken, Session fixation uvm.) Der Programmierer darf natürlich diese Mechanismen nicht umgehen.

Weiterhin unterstützt der Webserver Https als werden Daten verschlüsselt übertragen.

9.5.2. Maßnahmen zur Qualitätssicherung

9.5.2.1.Sicherheit

Durch Auswahl des Frameworks und der technischen Komponenten. Dadurch das der Programmierer die vom System vorgegebenen Sicherheitsmaßnahmen nicht umgeht. Aber auch durch gut Wartbarkeit weil das dazu verleitet Sicherheitsaktualisierungen auch einzuspielen.

9.5.2.2.Zuverlässigkeit

Durch Auswahl des Frameworks und der technischen Komponenten.

9.5.2.3.Robustheit

Durch Auswahl des Frameworks und der technischen Komponenten.

9.5.2.4.Wartbarkeit

Durch Auswahl des Frameworks und der technischen Komponenten. Aber auch durch gute Dokumentation z.b. im Sourcecode. Verwendung von Steckern statt Lötstellen im Sensorbau. In Code Dokumentation sollte mit PHP Documentor kompatibel sein da das auch im Framwork genutzt wird.

9.5.2.5.Oberfläche

Eine leicht anpassbare und flexible Oberfläche, ermöglicht es bei Problemen der Benutzer sofort Anpassungen vorzunehmen

Bei der Anwendung sollten die folgenden Kriterien besonders in Vordergrund stehen: Alle Daten sollen Übersichtlich dargestellt werden.. Die Anwendung soll zuverlässig über lange Zeit funktionieren Sie soll robust sein und benutzerfreundlich

9.6. Maßnahmen zur Qualitätssicherung

Sicherheit @Todo hübsch machen
Das Framework kümmert sich u viele Sicherheitsprobleme Selbständig (SQL Injections,

CSRF Attacken, Session fixation uvm.)
Die Programmierung umgeht an keiner Stelle diese Mechanismen.
Sicherheits aktualisierungen sind wirklich einfach einzuspielen
Der Webserver unterstützt HTTPs zur verschlüsselten übertragung der ‚Daten

Oberfläche , die Oberfläche bietet vielfältige weitere funktionalitäten , ist einfach an neue anforderungen anzupassen optisch sowie technisch. Damit kann sie jederzeit den Benutzerwünschen angepasst werden.

Des weiteren ist der Code ausreichend Dokumentiert um Jederzeit mit PHP dokumentor eine Api Dokumentation zu erzeugen, zumal das Standard Dokumentationssystem von Processwire ebenfalls PHP Documentor ist

9.7. Webanwendung

9.7.1. Datenbank

Aufgrund des Spiralmodells und vieler Überlegungen die beim erstellen der Applikation aufgekommen sind sind ungünstiger weise jede Menge Datenfelder hinzugekommen. Um den Umfang zu verdeutlichen,habe ich das zuerst geplante ER Diagramm noch mit eingefügt.

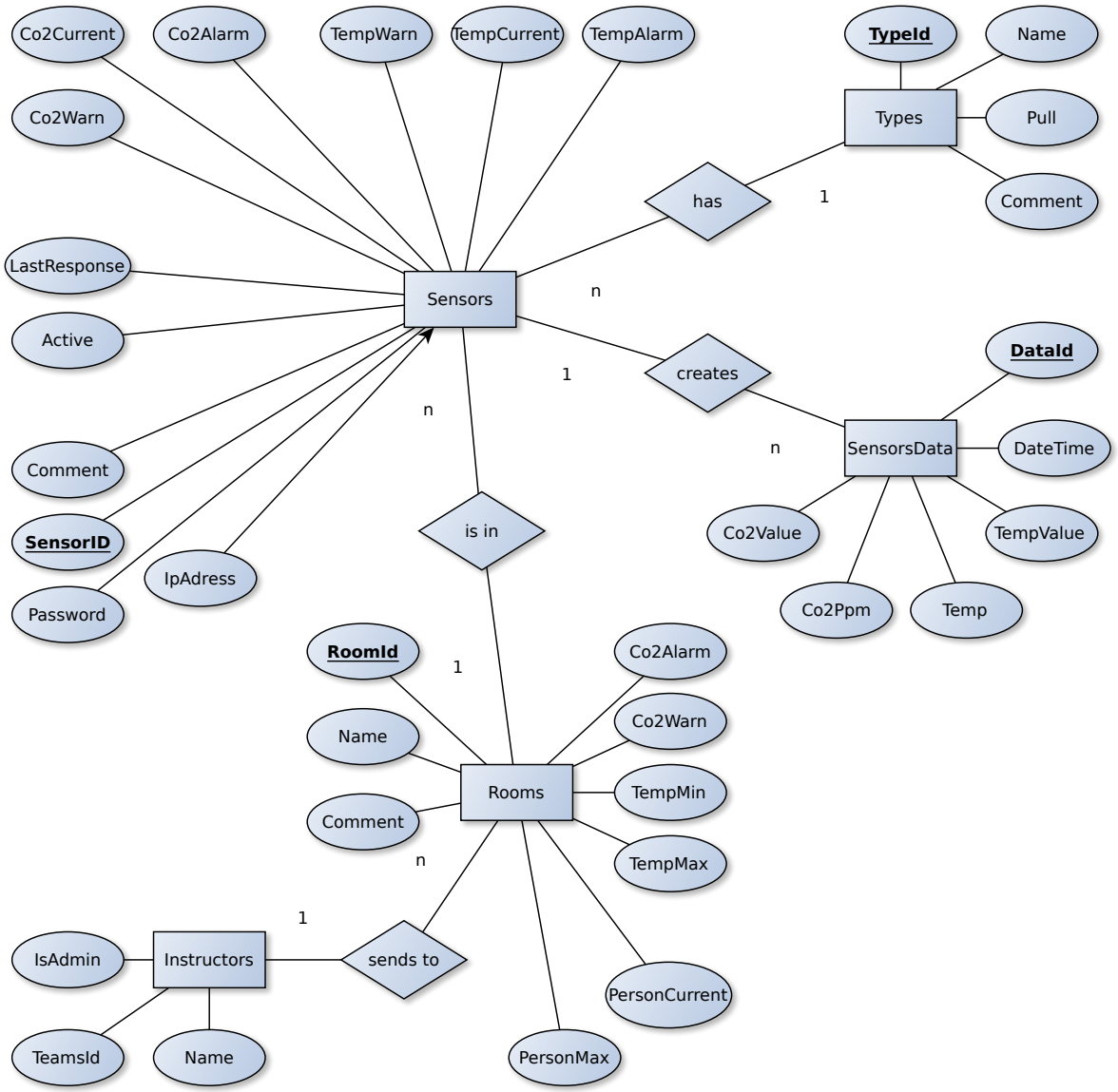


Abbildung 3: ER-Diagramm ursprünglicher Entwurf

Hier betrachte ich aber den Aktuellen Stand:

Anhang

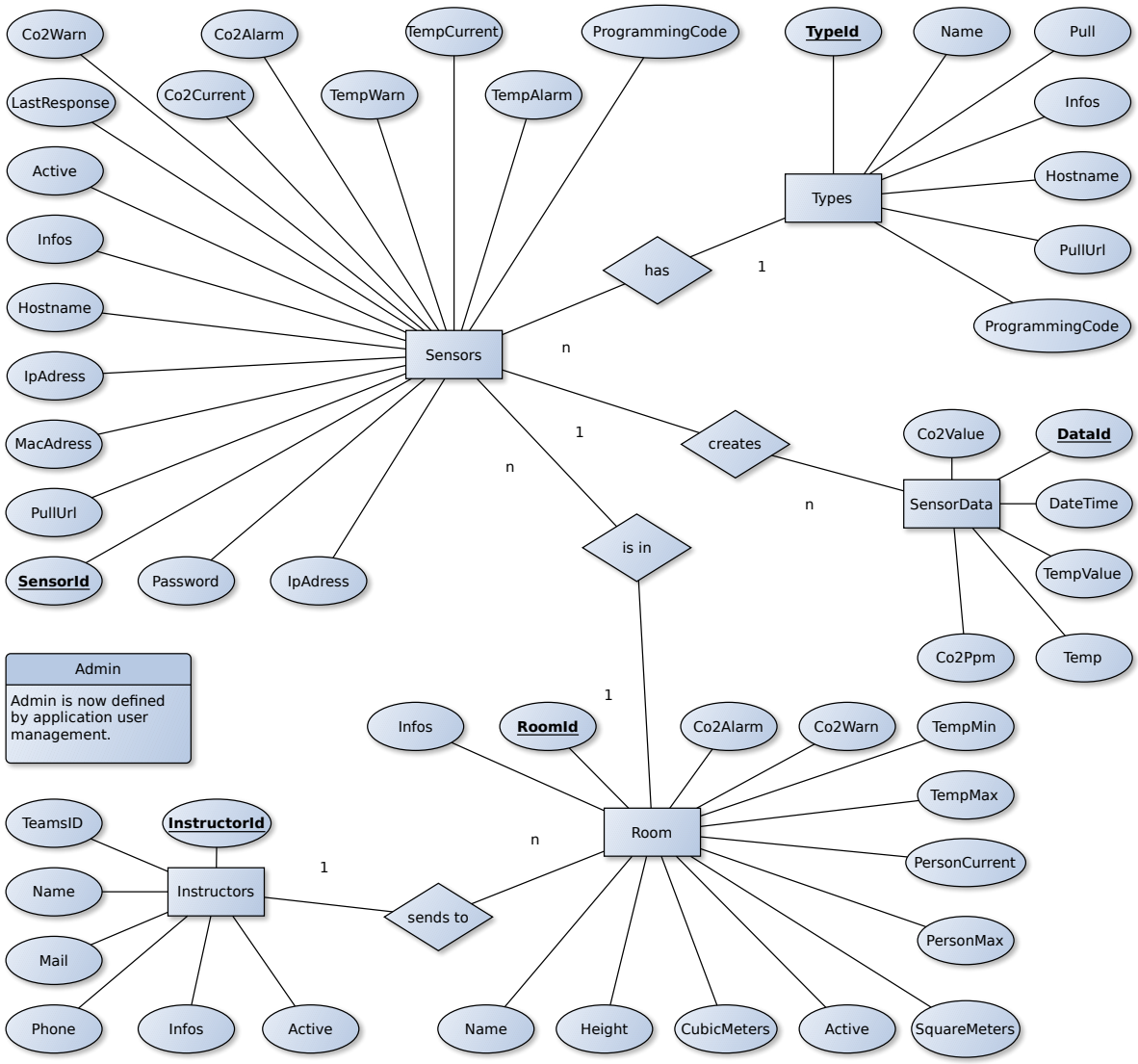
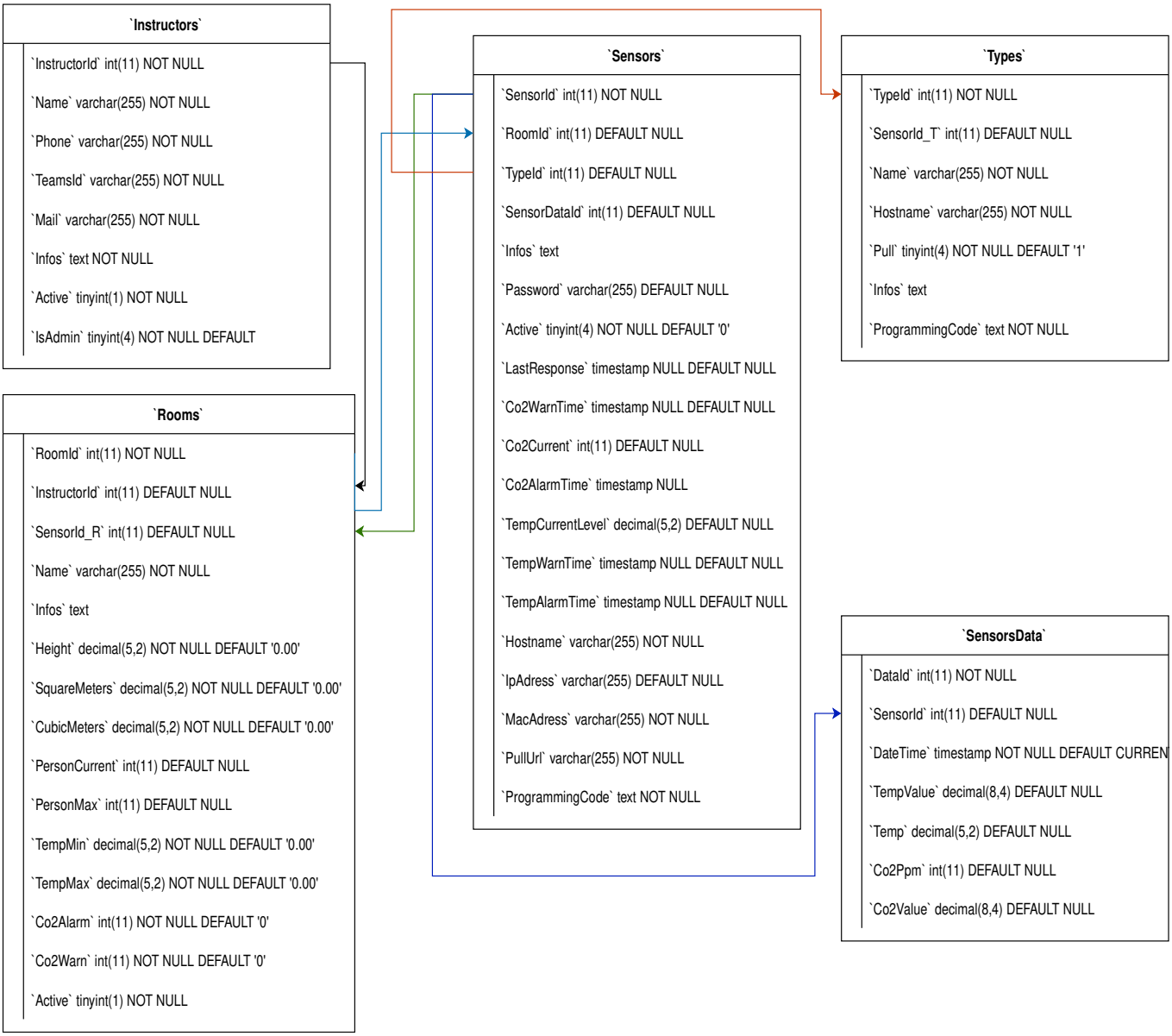


Abbildung 4: ER-Diagramm Aktuelle Fassung

9.7.1.1.Datenbank Struktur



9.7.2. Oberfläche

Processwire besitzt mehrere Module, die wirklich viel Arbeit übernehmen wenn es um die Erstellung von Formularen geht. Darüber hinaus lässt sich die Standard Oberfläche hervorragend mit dem Modularen Less Parser anpassen.

Es werden grundsätzlich mehrere gewöhnliche CRUD (**C**reate, **R**ead, **U**pdate and **D**elete) Oberflächen für die Haupt Artefakte der Datenbank benötigt(Sensors, SensorData, Rooms, Instructors und Types) Diese sollte alle auch passende/konfigurierbare Suchfelder besitzen.

10. Weitere Anhänge

Alle weiteren Anhänge die sonst nicht wirklich passten

10.1. Sensor Prototypen Eigenbau

Übersicht über die 3 geplanten Prototypen mit Preisen und Komponenten. Es Sollte 3 verschiedene Preisstufen berücksichtigt werden.Bei der Recherche hatte ich überdies den Eindruck das auch die teuren Kommerziellen Systeme zur Co2 Messung ebenfalls eins der beiden Sensormodelle benutzen. Zu diesem Schluss bin ich gekommen weil sich die Eichverfahren, Einbrennzeiten und Benutzungshinweise zu 100% mit denen der Dokumentationen der einzelnen Sensoren decken.

Tabelle 7: Sensor Eigenbau 3 Prototypen

System	Preis	Variante 1	Variante 2	Variante 3
NodeMCU Mainboard	4,66 €	4,66 €	4,66 €	4,66 €
Platine	1,60 €	1,60 €	1,60 €	1,60 €
revolt USB Netzteile & Smartphone-Halterung	11,50 €	11,50 €	11,50 €	11,50 €
Gehäuse mit Klarsicht-deckel	4,50 €	4,50 €	4,50 €	4,50 €
Diverse Kleinteile	2,50 €	2,50 €	2,50 €	2,50 €
MH-Z19C Co2 Sensor	23,40 €		23,40 €	23,40 €
MQ-135 C02/Luftqualität Sensor	2,66 €	2,66 €		

HT22 Temperatur-Feuchtigkeitssensor	7,99 €			7,99 €
LED- Ampel	1,83 €		1,83 €	
Beeper	1,76 €	1,76 €	1,76 €	1,76 €
LCD Anzeige	3,33 €			3,33 €
Summe		**29,18 €**	**51,75 €**	**61,24 €**

10.2. Kommerzielle Sensoren

Generell kranken die kommerziellen Systeme immer wieder unter unklaren Spezifikationen. API‘s werden nicht offengelegt oder stehen erst zur Verfügung wenn man gekauft hat. Features werden nur vage beschrieben und Kosten für Clouddienste müssen separat recherchiert und natürlich auch bezahlt werden.

Übertragungswege für Warnungen außerhalb von Email und SMS sind nicht vorgesehen, meist werden nicht alle SMS Portale unterstützt. Oft werden Produktserien einfach eingestellt so dass man bei teilweisem Ausfall ein komplett neues System benötigt.

Wenn die Verfügbarkeit besser wäre wäre die Entscheidung auf den BMC gefallen, leider ist diese so schlecht das ich mich für den **Clean-air-engineering Monitoring CA-M** entschieden habe.

Tabelle 8: Übersicht Kommerzielle Sensoren

	Virtenio Preon	BMC	Clean-air-engineering
Name	Cube Co2 Ampel	RTR-576 Funk Datenlogger	Monitoring CA-M Kohlendioxid
Kosten	179,00€ / Stück	528,00 € / Stück	547,40 € / Stück
Weitere Kosten	Wandhalterung 25,00€ / Stück	Netzteil 22,80 € / Stück Wandhalterung 27,28 € / Stück	
Cloud	Cloud für 24 Monate 45,60 € / Pro Gerät ?	Derzeit kostenlos.	Basis Version 199,38 € / Jahr Enterprise Version nur auf Anfrage.
Link	Virtenio.com	BMC.de	Clean-air-engineering.de

Anhang

Bemerk.	Da Kalibrierung notwendig handelt es sich wahrscheinlich um den billigsten am Markt verfügbaren Sensor, Kosten ca. 3,50€. Alarmwerte können nicht eingestellt werden. Reparatur ist nicht vorgesehen.	Schlechte Verfügbarkeit in Europa. Reparatur ist nicht vorgesehen.	Sensor Takte von kürzer als 2 Stunden erfordern Enterprise Version der Cloud. Reparatur ist nicht vorgesehen.
Eignung	Nur bedingt geeignet, da Features der Cloud unklar und ohne Kommerziellen Cloud Anbieter keine zentrale Datenerfassung. Kalibrierung ist notwendig also ist in regelmäßigen Abständen mit manuelle Wartungsarbeiten zu rechnen.	In Europa ist es im Moment wirklich schwierig die Basisstationen zu erhalten damit ist dieses System für den Einsatz hier nur bedingt brauchbar.	Generell gut , nur sehr teuer. Leider haben die Sensoren keine Ampel oder Anzeige. Für lokale Datenerfassung ist immer die Enterprise version der Cloud Software erforderlich, diese kann dann aber auch lokal installiert werden. Preis leider nur auf Anfrage.
Basis	keine	ca. 350 € (Dollarkurs)	458,15 €

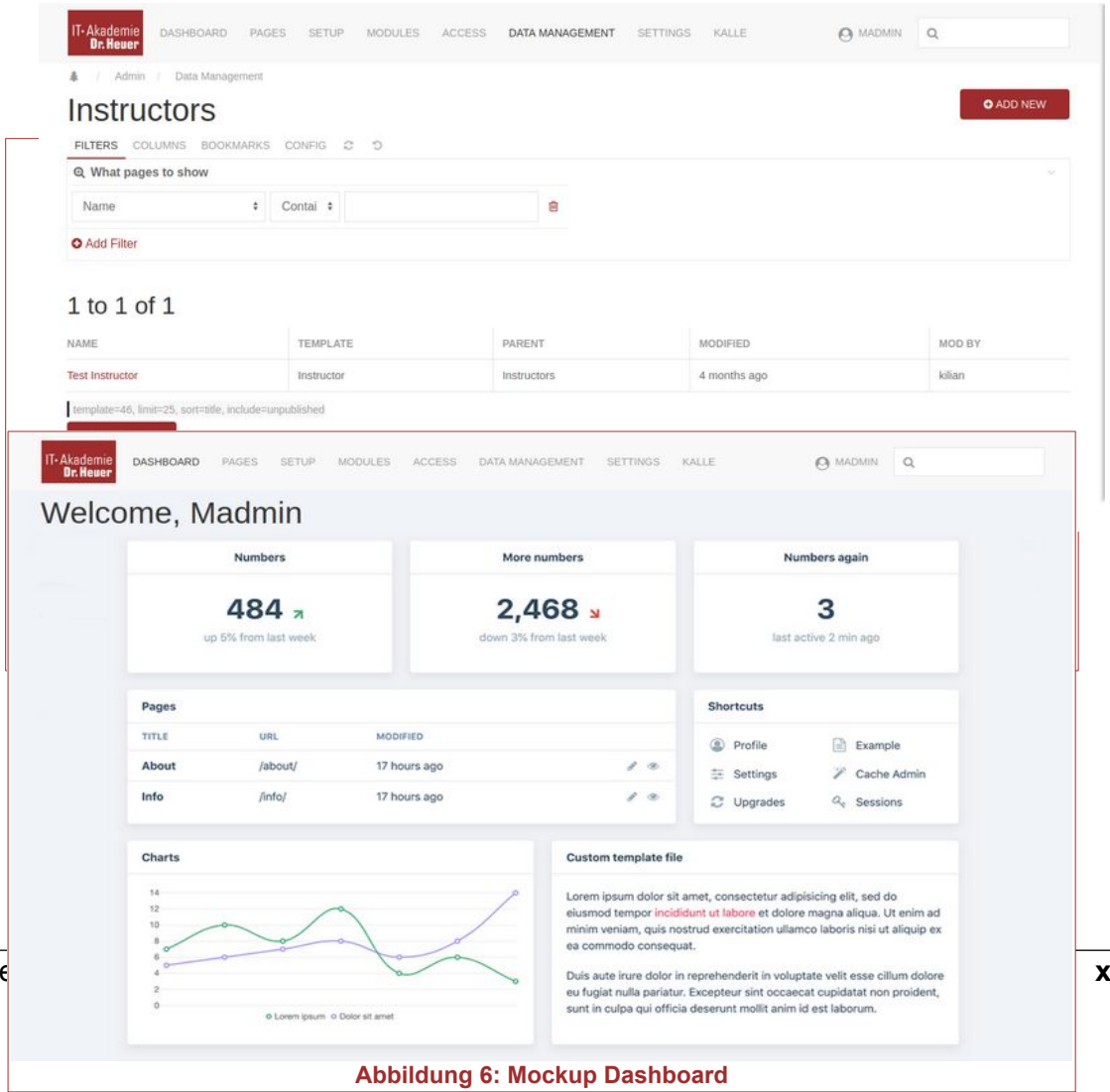
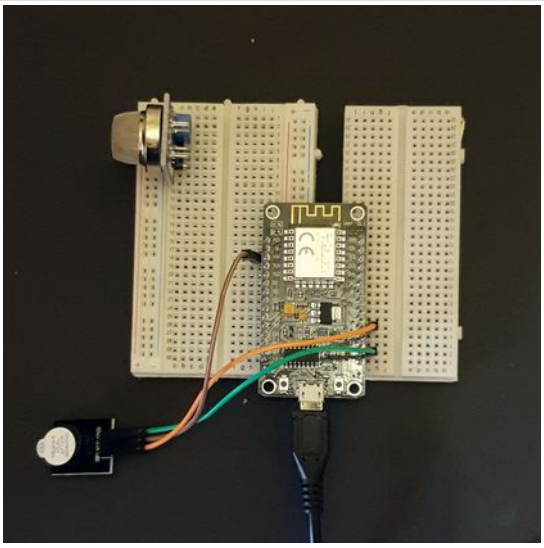
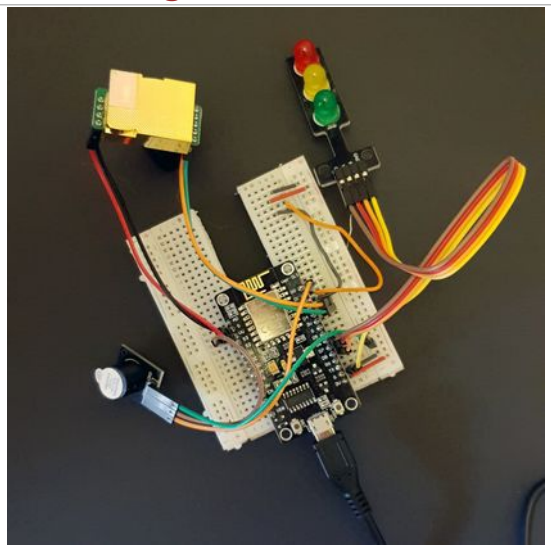


Abbildung 6: Mockup Dashboard

10.3. Sensoren Bilder @todo Bildbezeichnungen





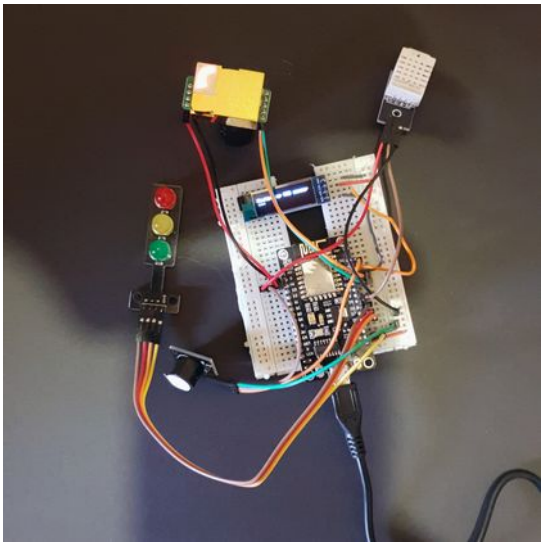


Abbildung 7: Arbeitsplatz mit Raspi

10.4. Codebeispiel Class dataObject

Die Klasse empfängt über die Statische Methode Deserialize eine JSON Datensatz und gibt sich selbst als Datenobjekt zurück. Diese Klasse wird auch im Empfangsmodul wiederverwendet

```
<?php
/**
 * Transform sensor data into a Data Object and do some basic
    validation and sanitation
 *
 * Still the application needs to do some checks, but a lot of crap is
    filtered out.
 * In addition this class is used for testing the test Sensor class in
    recieve.php
 */
class dataObject {

    /** @var string $title */
    public $title;

/************ Snip ***********/
/… gekürzt
/************ Snap ***********/

    /** @var string $ipAddress*/
    public $ipAddress;

    /**
     * Deserialize static method
     *
     * This is more or less a self factory that creates an instance of
    the dataclass and fills
     * it whith the attributes aqired from JSON string. If there are
    inconsintencies whith
     * those attributes it will fail and return an empty object.
     *
     *  @return object/false
     */
    public static function Deserialize($jsonString)
    {
        $classInstance = new DataObject();

        // needed for check if all are filled
        $cProperties = count((array)$classInstance);

        $jsonArray = json_decode($jsonString);

        $cFilled = 0;

        foreach ($jsonArray as $key => $value) {
            if (!property_exists($classInstance, $key)) continue;

            $classInstance->{$key} = $value;
            $cFilled++;
        }
```

```
        // All properties have benn filled?
        if ($cFilled == $cProperties ){
            // Validation ok so return Object
            if ($classInstance->basicValidation()){
                // but some basic sanitation first
                $classInstance->basicSanitation();
                return $classInstance;
            }
            else {
                echo"Basic Validation failed!\n";
            }

        }
        else {
            echo"Invalid properties filled: $cFilled  / Properties:
    $cProperties\n";
        }
        // concerning PHP manual its better to use both
        $classInstance = NULL;
        unset($classInstance);
        return (object)[];   // Returns empty Object
    }

    /**
     * Basic Validation method
     *
     * Returns True if validation is OK , false otherwise
     *
     * @return bool
     */
    protected function basicValidation() {

        if (!preg_match("/^(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9][0-9]|
    [0-9])(.(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9][0-9]|[0-9])){3}$/",
    $this->ipAddress)) {
            echo("IPAdress Validation Failed{$this->ipAddress}\n");
            return false;
        }

        if (!is_string($this->title) OR strlen($this->title) > 250 ){
            echo("Title Validation Failed: {$this->title}\n");
            return false;
        }
/************ Snip ***********/
/… gekürzt
/************ Snap ***********/

        if (!is_numeric($this->co2Value)){
            echo("co2Value Validation Failed: {$this->co2Value} \n");
            return false;
        }
        return true;
    }
```

```
    /**
     * Just some really basic sanitation.
     *
     * @return void
     */
    protected function basicSanitation() {
        $this->tempValue = (float)$this->tempValue;
        $this->temperature= (float)$this->temperature;
        $this->co2Ppm= (int)$this->co2Ppm;
        $this->co2Value= (float)$this->co2Value;
    }
}
```