|  |  |
| --- | --- |
| AVEC_LOGO_R_RGB.pdf | |
| IPA-Bericht 2020 | |
|  | |
| BBB Gebäude Management Pilot Versuch | |
| \\vfbdn571\home70$\REINEGGE\My Documents\AXI NEU\AVECTRIS_Benutzeranleitung_A4\Links\image001.png | |
|  | |
| Nils Egger  Projektleiter | |
| 23.03.2020 |  |
|  | |

Änderungsindex

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Beschreibung | Erstellt | Datum |
| 0.1 | Erstellung des Dokumentes | Nils Egger | 23.03.2020 |
|  |  |  |  |
|  |  |  |  |

Inhaltsverzeichnis

[1 Aufgabenstellung 6](#_Toc35948470)

[1.1 Titel der Arbeit 6](#_Toc35948471)

[1.2 Ausgangslage 6](#_Toc35948472)

[1.3 Detaillierte Aufgabenstellung 6](#_Toc35948473)

[1.3.1 Beschreibung 6](#_Toc35948474)

[1.3.2 Generelle Eigenschaften 6](#_Toc35948475)

[1.3.3 Anforderungen an die Webapplikation 6](#_Toc35948476)

[1.4 Mittel und Methoden 8](#_Toc35948477)

[1.5 Vorkenntnisse 8](#_Toc35948478)

[1.6 Vorarbeiten 8](#_Toc35948479)

[1.7 Neue Lerninhalte 8](#_Toc35948480)

[1.8 Arbeiten in den letzten 6 Monaten 8](#_Toc35948481)

[1.9 Individuelle Bewertungskriterien 8](#_Toc35948482)

[1.9.1 Responsive Design 8](#_Toc35948483)

[1.9.2 Browserkompatibilität 9](#_Toc35948484)

[1.9.3 Einhaltung Corporate Design 9](#_Toc35948485)

[1.9.4 Codingstyle – Dokumentation 9](#_Toc35948486)

[1.9.5 Validierung Eingaben 10](#_Toc35948487)

[1.9.6 Anzeige der Luftqualitäts-Messwerte 10](#_Toc35948488)

[1.9.7 Grafische Darstellung der Messwerte 11](#_Toc35948489)

[2 Projektorganisation 11](#_Toc35948490)

[2.1 Methode 11](#_Toc35948491)

[2.2 Auftraggeber 11](#_Toc35948492)

[2.3 Beteiligte Personen 11](#_Toc35948493)

[2.4 Durchführungsort 11](#_Toc35948494)

[3 Deklaration der Vorkentnisse 12](#_Toc35948495)

[4 Deklaration der Vorarbeiten 12](#_Toc35948496)

[5 Deklaration der benützten Firmenstandarts 13](#_Toc35948497)

[6 Zeitplan 13](#_Toc35948498)

[6.1 Bemerkungen 13](#_Toc35948499)

[6.2 Legende 13](#_Toc35948500)

[6.3 Meilensteine 13](#_Toc35948501)

[7 Organisation der Arbeitsergebnisse 15](#_Toc35948502)

[8 Informieren 15](#_Toc35948503)

[8.1 Ziel 15](#_Toc35948504)

[8.2 Anforderungen 15](#_Toc35948505)

[8.3 Deadline 15](#_Toc35948506)

[8.4 Herausforderungen 15](#_Toc35948507)

[8.4.1 JavaScript Diagramme 15](#_Toc35948508)

[8.4.2 Sensor Wert Beobachter 17](#_Toc35948509)

[8.4.3 Responsive Design 17](#_Toc35948510)

[8.4.4 Sphinx Dokumentation 18](#_Toc35948511)

[8.4.5 Python API Framework 18](#_Toc35948512)

[8.4.6 HTML Präprozessor 19](#_Toc35948513)

[8.4.7 Tür Schaltungssensor Wert 19](#_Toc35948514)

[9 Planen 20](#_Toc35948515)

[9.1 Use Case 20](#_Toc35948516)

[9.2 System 20](#_Toc35948517)

[9.2.1 Allgemein 20](#_Toc35948518)

[9.2.2 Uvicorn Server 20](#_Toc35948519)

[9.2.3 Websockets 21](#_Toc35948520)

[9.2.4 PostgreSQL Datenbank 21](#_Toc35948521)

[9.2.5 Sensoren 21](#_Toc35948522)

[9.3 Wert Beobachter 22](#_Toc35948523)

[9.4 Authentifizierung 22](#_Toc35948524)

[9.5 Webseite 23](#_Toc35948525)

[9.5.1 Admin Dashboard 23](#_Toc35948526)

[9.5.2 Personal Verwaltung 24](#_Toc35948527)

[9.5.3 Räume Verwaltung 24](#_Toc35948528)

[9.5.4 Raum Ansicht 26](#_Toc35948529)

[9.5.5 Sensoren Verwaltung 26](#_Toc35948530)

[9.6 Sensor Ansicht 27](#_Toc35948531)

[9.6.1 Reinigungspersonal Dashboard 28](#_Toc35948532)

[9.7 Konzeptionelles Modell 30](#_Toc35948533)

[9.8 Logisches Modell 31](#_Toc35948534)

[9.9 API Struktur 31](#_Toc35948535)

[9.10 Sicherheit 34](#_Toc35948536)

[9.10.1 Feld Validierung 34](#_Toc35948537)

[9.10.2 SQL Injection 34](#_Toc35948538)

[9.10.3 Cross Site Scripting 34](#_Toc35948539)

[9.11 Testen 34](#_Toc35948540)

[9.11.1 Testkonzept 34](#_Toc35948541)

[9.11.2 Testfallspezifikationen 35](#_Toc35948542)

[10 Glossar 39](#_Toc35948543)

[11 Literaturverzeichnis 39](#_Toc35948544)

[12 Abbildungsverzeichnis 40](#_Toc35948545)

Teil 1: Umfeld und Ablauf

# Aufgabenstellung

## Titel der Arbeit

Auswertung und Visualisierung von IoT Sensor Daten

## Ausgangslage

Unsere Kundin die Berufsfachschule Baden (BBB) möchte in einem Pilotversuch erste Funktionen mittels IoT Sensoren für das Gebäudemanagement testen. Als erstes soll die Toiletten-Benutzung gemessen werden. Diese Informationen werden verwendet um den Reinigungsvorgang zu optimieren. Als zweite Messkategorie wird die Luftqualität in den Unterrichtszimmern überwacht. Die Pilotinstallation wird auf das Gebäude Bruggerstrasse der BBB begrenzt.

Für den Hausdienstleiter sollen die Daten der Sensoren in einem Dashboard angezeigt werden. Für das Reinigungspersonal sind Tablet-PCs vorgesehen, auf welchen die nötigen Informationen für die Reinigungstour angezeigt und Rückmeldungen erfasst werden können.

Die Applikation soll bezüglich Sicherheit den gängigen Standards entsprechen und eine unerlaubte Einsicht und Manipulation der Daten verhindern. Für die Umsetzung des GUI kommen die Corporate Design Vorgaben der BBB zum Einsatz.

## Detaillierte Aufgabenstellung

### Beschreibung

Die Aufgabe besteht im Groben in der Umsetzung der folgenden Funktionen:

* Interpretation und Auswertung der IoT Sensordaten aus der Datenbank.
* Programmierung eines Dashboards für das Gebäudemanagement (Reinigung, Luftqualität)
* Programmierung einer Detailansicht von Sensordaten
* Programmierung einer Übersicht für das Reinigungspersonal. Optimiert für die Benützung auf einem Tablet mit Touch-Bedienung.
* Programmierung von Notifikationen in der Anwendung bei der Erreichung von vordefinierten Grenzwerten der Toilettenbenutzung und bei der Luftqualität.

### Generelle Eigenschaften

* Die Anwendung muss als Webapplikation in der Programmiersprache Python (Backend) und HTML/Javascript/JQuery (Frontend) umgesetzt werden.
* Das GUI soll mit Responsive Design für Desktop, Tablet und Mobile umgesetzt werden. Für Mobile sollen die Standard-Browser der Android und iOS Geräte verwendet werden, respektive während der IPA können entsprechende Emulatoren verwendet werden.
* Die Anwendung soll flexibel und modular umgesetzt werden, so dass weitere durch Sensoren messbare Indikatoren in das Dashboard aufgenommen werden können.
* Die Anwendung soll die folgenden Browser in der aktuellen Version unterstützen: Chrome, Firefox und Edge.

### Anforderungen an die Webapplikation

#### Login Funktion

Der Zugang zur Applikation soll mit Benutzer/Passwort geschützt werden.

#### Dashboard Hausdienst

Das Dashboard dient dem Hausdienst als Übersicht über den aktuellen Zustand bezüglich Luftqualität und Reinigungsbedarf im Gebäude.

**Überblick aktuelle Luftqualität in den einzelnen Räumen**

In einer Übersicht sollen die aktuellen Messwerte (Temperatur, Luftfeuchtigkeit und CO2) angezeigt werden. Die Darstellung soll übersichtlich z.B. gemäss Aufteilung des Gebäudes angezeigt werden.

**Zeitlicher Verlauf der Luftqualität**

In einer Detailansicht eines Messpunktes sollen die Messwerte in grafischer Form angezeigt werden. Z.B. als Liniengrafik mit der Möglichkeit über einen wählbaren Zeitraum.

**Anzeige Reinigungsbedarf**

In einer separaten Seite sollen die Zählerdaten der Toilettenbenutzung angezeigt werden. Der Reinigungsbedarf soll grafisch sichtbar sein mit einem speziellen Symbol oder mit farblicher Unterscheidung.

**Administrationsseite**

Die Grenzwerte für die Luftqualität wie auch für die Toiletten-Reinigung sollen auf einer Administrationsseite angepasst werden können.  
Ebenfalls soll eine Reinigung unabhängig des Benutzungs-Zählers ausgelöst werden können.  
Alternativ kann diese Funktion auch in die Detailansicht der Sensoren eingebaut werden.  
Einfache Benutzerverwaltung (Benutzer erstellen, Rollen zuweisen, Passwort reset)

#### Übersicht für Reinigungspersonal

**Liste zu reinigender Stellen**

Die zu Reinigenden Toiletten sollen sortiert nach Stockwerk und Gebäude Seite angezeigt werden, so dass eine Reinigungstour optimiert durchgeführt werden kann.

**Rapportierung nach erfolgter Reinigung**

Pro Toilette soll eine Funktion zur Rapportierung der Reinigung mit Kommentarfunktion eingebaut werden. Zum Beispiel, wenn die Reinigungsperson einen Defekt feststellt.  
z.B. Checkbox «Reinigung erledigt», Textfeld «Möchten Sie etwas melden»  
Nach der Rapportierung startet der Zähler der Toilette automatisch einen neuen Zyklus.

**Materialliste zur Vorbereitung der Reinigungstour**

Aufgrund der Anzahl Benutzungen der zu reinigenden Toiletten soll eine Materialliste (WC-Papier, Handtücher, Seife etc.) erstellt werden, so dass das Reinigungspersonal die richtige Menge auf die Reinigungstour mitnehmen kann. Diese Funktion kann mit einer relativ einfachen Formel umgesetzt werden:

Anzahl Benutzung x Anzahl Toiletten x Erfahrungswert Material

Der Erfahrungswert des benötigten Materials im Bezug zur Benutzung muss noch definiert werden und ist nicht Bestandteil dieser IPA.

## Mittel und Methoden

* Python
* Javascript + JQuery
* IDE: Pycharm & Datalore
* Windows Server
* GIT

## Vorkenntnisse

* HTML/CSS/Javascript (3 Jahre)
* Python Programmierung (1 Jahr)
* JQuery

## Vorarbeiten

* Planen und Einrichten der Datenbank für die Sensordaten.
* Sensordaten erheben und/oder Testdaten erstellen damit genügend Daten für die Validierung der Funktionen vorhanden sind.
* Web Server mit Python einrichten.
* Wireframes für die verschiedenen Screens erstellen.
* HTML Templates vorbereiten.
* Token Basierte Authentifizierung
* GIT Repository
* Klassen Diagramm

## Neue Lerninhalte

* Web Grafiken
* Token Basierte Authentifizierung
* Sensor Daten auswerten und darstellen
* Darstellung von Echtzeitdaten in einem Web basierten Dashboard

## Arbeiten in den letzten 6 Monaten

* Python API mit Firebase
* Webseite mit PHP/Wordpress für BBBaden realisieren
* Wordpress Plugins programmieren
* HTML/CSS/Javascript Templates umsetzen

## Individuelle Bewertungskriterien

### Responsive Design

**Leitfrage**

Wird die Webanwendung dynamisch auf verschiedene Bildschirmgrössen für Desktop, Tablet und Mobile angepasst?

|  |  |
| --- | --- |
| Gütestufe | Beschreibung |
| 3 | Die Webanwendung wird auf allen drei Bildschirmgrössen angepasst angezeigt. |
| 2 | Zwei Bildschirmgrössen werden angepasst angezeigt. |
| 1 | Eine Bildschirmgrösse wird angepasst angezeigt. |
| 0 | Bei keiner Bildschirmgrösse wird die Anwendung angepasst angezeigt |

### Browserkompatibilität

**Leitfrage**

Unterstützt die Anwendung die drei in der Aufgabenstellung geforderten Browser Edge, Firefox und Chrome bezüglich Funktionen und Design.

|  |  |
| --- | --- |
| Gütestufe | Beschreibung |
| 3 | Alle drei Browser werden bezüglich Funktionen und Design vollständig unterstützt. Es sind höchstens minimale Unterschiede im Design zu erkennen. |
| 2 | Zwei der drei in der Leitfrage erwähnten Browser werden vollumfänglich unterstützt. |
| 1 | Einer der drei in der Leitfrage erwähnten Browser wird vollumfänglich unterstützt. |
| 0 | Keiner der drei in der Leitfrage erwähnten Browser wird vollumfänglich unterstützt. |

### Einhaltung Corporate Design

**Leitfrage**

Ist bei der Webanwendung das Corporate Design der BBB gemäss BBB\_CD\_Manual.pdf eingehalten?

|  |  |
| --- | --- |
| Gütestufe | Beschreibung |
| 3 | Die Grafik-Elemente Schrift, Farben & Logo werden gemäss obenstehenden Vorgaben korrekt verwendet. |
| 2 | Zwei der Grafik-Elemente werden richtig verwendet. |
| 1 | Ein Grafik-Element wird korrekt eingesetzt. |
| 0 | Keines der Grafik Elemente wird korrekt eingesetzt. |

### Codingstyle – Dokumentation

**Leitfrage**

Ist der Quellcode dokumentiert? Hilft der Text, die Funktionalität zu verstehen und nachzuvollziehen?

|  |  |
| --- | --- |
| Gütestufe | Beschreibung |
| 3 | Der Text erklärt was notwendig ist, um die Funktion/Methode besser zu verstehen. Allfällige Richtlinien sind eingehalten. |
| 2 | Der Text hilft nur zum Teil weiter. Allfällige Richtlinien sind teilweise berücksichtigt. |
| 1 | Der Text hilft selten weiter. Allfällige Richtlinien sind offensichtlich verletzt. |
| 0 | Es ist wenig bis gar nichts dokumentiert. |

### Validierung Eingaben

**Leitfrage**

Werden die eingegebenen Daten aller Felder in allen Formularen validiert und allgemein verständliche Fehlermeldungen angezeigt?

|  |  |
| --- | --- |
| Gütestufe | Beschreibung |
| 3 | Alle drei Kriterien (Felder, Formulare, Fehlermeldungen) sind erfüllt |
| 2 | Zwei Kriterien sind erfüllt |
| 1 | Ein Kriterium ist erfüllt |
| 0 | Kein Kriterium ist erfüllt |

### Anzeige der Luftqualitäts-Messwerte

**Leitfrage**

Erfolgt eine korrekte Auswertung der Messwerte der Sensoren und werden diese richtig im Dashboard angezeigt? Beispielsweise soll die relative Luftfeuchtigkeit in Prozent angezeigt werden.

|  |  |
| --- | --- |
| Gütestufe | Beschreibung |
| 3 | Alle drei Messwert-Kategorien werden richtig interpretiert und angezeigt. |
| 2 | Zwei Messwert-Kategorien werden richtig interpretiert und angezeigt. |
| 1 | Eine Messwert-Kategorie wird richtig interpretiert und angezeigt. |
| 0 | Keine Messwert-Kategorie wird richtig interpretiert und angezeigt. |

### Grafische Darstellung der Messwerte

**Leitfrage**

Kann der Verlauf der drei Luftqualitätsmesswerte grafisch und mit wählbarem Zeitraum angezeigt werden?

|  |  |
| --- | --- |
| Gütestufe | Beschreibung |
| 3 | Alle drei Messwerte werden als Grafik und mit wählbarem Zeitraum angezeigt. |
| 2 | Alle drei Messwerte werden als Grafik angezeigt, jedoch ohne wählbaren Zeitraum. |
| 1 | Nur ein Teil der Messwerte werden als Grafik angezeigt. |
| 0 | Kein Messwert kann als Grafik angezeigt werden. |

# Projektaufbauorganisation

## Methode

Die gesamte IPA wird mit IPERKA (Informieren, Planen, Entscheiden, Realisieren, Kontrollieren und Auswerten) durchgeführt. IPERKA ist einfach anzuwenden und wurde oft während meiner Lehre benutzt, daher ist diese Projektmethode die einzige welche für mich in Frage kommt.

## Auftraggeber

Die Berufsbildung Baden ist der Auftraggeber für diesen Pilot Versuch.

## Beteiligte Personen

**Kandidat**

Nils Egger  
[nils.egger@avectris.ch](mailto:nils.egger@avectris.ch)

**Verantwortliche Fachkraft**

Daniel Säuberli  
[daniel.saeuberli@avectris.ch](mailto:daniel.saeuberli@avectris.ch)

**Berufsbildner**

Mark Grabner  
[mark.grabner@avectris.ch](mailto:mark.grabner@avectris.ch)

## Durchführungsort

Die IPA wird in der AVECTRIS AG an der unten aufgeführten Adresse durchgeführt.

Avectris AG  
Bruggerstrasse 68  
5401 Baden

# Deklaration der Vorkentnisse

|  |  |
| --- | --- |
| Kenntnis | Bemerkung |
| Python | Mit Python automatisiere ich meist kurze wiederholende Tasks. Vor knapp einem Jahr habe ich damit mit einem Framework für Backend API’s begonnen. |
| HTML, CSS | HTML und CSS verwende ich seit Beginn meiner Lehre, dies ist definitiv kein Neuland mehr. |
| Bootstrap | Ich kann Bootstrap ohne Probleme anwenden, jedoch ist es für mich neu die SCSS Dateien anzupassen. |
| JavaScript, JQuery | Bei den meisten Webseiten welche ich schreibe verwende ich JavaScript und JQuery. Daher kenne ich diese gut. |
| PostgreSQL | Ich beherrsche allgemeine SQL Fähigkeiten aber noch keine vertieften Kenntnisse. Ich arbeite erste seit Beginn des Jahres wieder mit SQL. |
| Jekyll | Jekyll ist ein HTML-Preprocessor bei welchem ich die Basics verstehe. |

# Deklaration der Vorarbeiten

* Mockups der Webseite
* Sensor Wert Dekodierung vorbereitet für die Adeunis RF und Elsys CO2 Sensoren
* Sensor Wert Abhörung von Loriot
* Installation der Entwicklungsumgebung
* Vorbereitung des Servers
* Konzeptionelles und Logisches Modell
* SQL Abfragen für den Aufbau der Datenbank
* Sphinx Dokumentation für das Backend eingerichtet
* Jekyll mit Bootstrap eingerichtet für das Frontend
* Token Authentifizierung
* Framework für die Erstellung von Model Controllers für Models
* Python Client für PostgreSQL

# Deklaration der benützten Firmenstandarts

Für dieses Dokument wurde eine Word Vorlage für Projekt Berichte verwendet.

# Zeitplan

## Bemerkungen

Der Zeitplan ist in ein 2 Stunden Raster aufgeteilt. Somit besteht ein voller Tag aus 8 Stunden. Am 25. März und 01. April bin ich ein halber Tag in der Schule und daher bestehen diese 2 Tage nur aus 4 Stunden. Auch der Zeitplan ist nach IPERKA aufgeteilt, die Phasen sind mit einem grauen Balken aufgeteilt.

Nach Vorschlag des FArbeit\_2020.pdf Dokumentes, sind mehr als 40% der Stunden für dokumentieren eingeteilt.

## Legende

|  |  |
| --- | --- |
| Zustand | Kennzeichnung |
| Ist | Grün |
| Soll | Orange |
| Ist-Meilenstein | Güner Richtungspfeil |
| Soll-Meilenstein | Oranger Richtungspfeil |

## Meilensteine

Die Meilensteine sind nach den IPERKA Phasen gesetzt. Zudem ist die Phase Realisierung in zwei Meilensteine aufgeteilt. Die Erste nach dem Ende der Realisierung des Backend und die Zweite nach der Fertigung des Frontend.



# Organisation der Arbeitsergebnisse

Die Arbeitsergebnisse werden mit dem mit GIT versioniert und auf github hochgeladen. So kann auch bei lokalem Datenverlust auf jede Version zurückgegriffen werden.

Jeder Commit ist mit einem Kommentar versehen, so ist es einfach jede Version wiederzufinden.

Alle Arbeitsergebnisse werden mindestens einmal pro Stunde hochgeladen.

Die Wiederstellung der Dokumente ist durch das Hochladen auf eine Cloud, in meinem Fall github, sichergestellt.

Die Arbeitsergebnisse werden die ganze IPA hindurch dieselbe Qualität haben.

Der IPA Bericht wird in Word verfasst und letztendlich zu einer PDF Datei konvertiert. Mein Python Code wird mit Inline Kommentaren und [Sphinx](https://www.sphinx-doc.org/en/master/) dokumentiert, und schlussendlich als Webseite im Anhang vorhanden sein.

Teil 2: Projekt-Dokumentation

# Informieren

## Ziel

Das Ziel ist es für die BBB eine Webseite zu erstellen, welche dem Reinigungspersonal hilft zu wissen, wo und wann geputzt werden muss. Beim Pilotenversuch geht es primär darum Sensor Werte darzustellen und auf diese zu reagieren. Zum Beispiel soll eine Meldung im Dashboard erscheinen, wenn ein gewisser CO2 Wert überschritten wird. Diese Meldung soll dann vom Reinigungspersonal abgearbeitet werden. Hierbei wäre damit Fenster öffnen gemeint.

## Absicht

Die Webapplikation soll dabei helfen, dass weniger unnötig geputzt werden muss. So könnte man im Ideal fall erreichen, dass die Berufsbildung Baden weniger Reinigungspersonal anstellen muss.

## Anforderungen

* Responsive Design
* Browserkompatibilität
* Einhaltung Corporate Design
* Quellcode Dokumentation
* Validierung der Eingaben von Benutzern
* Anzeige der Luftqualitäts-Messwerte
* Grafische Darstellung der Messwerte

## Ressourcen

Die Entwicklung geschieht auf einem normalen Windows-10 PC und für die endgültige Applikation steht ein interner Windows Server zur Verfügung.

Zusätzlich sind für die Messung von Daten drei verschiedene Arten von Sensoren in Betrieb.

## Deadline

Die Applikation soll bis und mit dem 08.04.2020 fertig sein.

## Herausforderungen

### JavaScript Diagramme

Die Sensordaten müssen auf eine Art Visuell dargestellt werden. Ich stelle mir dies anhand einer JavaScript Bibliothek vor. Ich kenne noch keine und muss mir eine Liste mit Google beschaffen.

Zwei Bibliotheken welche mir sympathisch vorkommen sind folgend aufgelistet. Beide Bibliotheken sind gratis und open source.

* [Chart.js](https://www.chartjs.org/)

Als ich ein einfaches Beispiel für Chart.js erstellte, fiel mir den Anfang sehr schwer, da es schwierig zu sein scheinte, die Diagramm Grösse zu verändern. Die Lösung hierzu war es, das Diagramm in einem div zu verpacken und die Grösse des divs zu setzen. Somit lässt sich mit folgendem Code ein Diagramm für einen Sensor erstellen. Der Code wurde von der [offiziellen Dokumentation](https://www.chartjs.org/docs/latest/charts/line.html) entnommen.



Abbildung 1 Chart.js Resultat

Abbildung 2 Chart.js Code

* [Chartist.js](https://gionkunz.github.io/chartist-js/)

Chartist konnte ich ohne weitere Probleme sehr schnell ein Beispiel mit den gleichen Daten wie oben implementieren. Es scheint minimiert auf das wichtigste zu sein.



Abbildung 3 Chartist.js Resultat

Abbildung 4 Chartis.js Code

### Sensor Wert Beobachter

Ein Sensor kann mehrere Werte messen, auf eine Weise müssen Benutzer definieren können, auf welche Werte geachtet werden müssen und wann ein Richtwert oder Zählerstand überschritten wurde. Dies möchte ich möglichst erweiterbar lösen. Jedoch komme ich vermutlich nicht darum, dass bei jeder neuen Art von Sensor ein neuer Wert Dekodierung Algorithmus von einem Programmierer geschrieben werden muss. Die Werte eines Sensors werden meist in Hex übertragen und müssen dann Anhand der Datenblätter ausgelesen werden.

Im Prinzip stelle ich es mir das so vor:

Ein Sensor Wert wird erhalten und anhand des passenden Sensor Dekodierers in die verschiedenen Werte umgewandelt. Danach wird bei jedem der einzelnen Werte geprüft, ob dieser seinen Richtwert oder Zählerstand überschritten hat. Wenn ja, wird eine spezifische Meldung für jenen Wert ausgelöst.



Abbildung 5 Sensor Wert Ablauf

### Responsive Design

Auch mit Bootstrap ist ein Responsive Design schwierig Korrekt umzusetzen. Zudem werde ich die SCSS Dateien von Bootstrap so ändern müssen, dass sie dem Corporate Design der BBB entsprechen. Das Reinigungspersonal wird sich mit Tablets im Gebäude herumgeben, daher ist es für mich aus sehr wichtig, dass das Frontend für Touch-Steuerung optimiert ist. Dies ist für mich auch eher Neuland und werde vermutlich meine Browser Ansicht auf die Grösse eines Tablets ändern damit ich ständig das Resultat sehe, welches schlussendlich auch wirklich benutzt wird. In Chromium basierenden Browsers kann man dies mit der F12 Taste erreichen. 

Abbildung 6 Chromium Entwickler Modus



Abbildung 7 Chromium Responsive Modus

### Sphinx Dokumentation

Python lässt sich anhand von sogenannten Docstrings dokumentieren. Diese kann man dann einfach mit Sphinx zu einer Webseite generieren lassen. Meine Herausforderung hier ist, dass ich diese gründlich nachführe und gleich nach der Erstellung einer Funktion diese auch wirklich schreibe. $

Hierbei gilt, dass meine Docstrings dem Standard von Google folgen, und folgenderweise aussehen sollten.



Abbildung 8 Google Style Docstrings

### HTML Präprozessor

Damit ich nicht mehrmals das gleiche HTML, wie der Header, schreiben werde, muss ich mir einen HTML Präprozessor beibringen, welcher fähig ist Templates in Seiten einzubinden und daraus eine statische Seite zu erstellen. Vor der IPA habe ich mir deswegen noch [Jekyll](https://jekyllrb.com/) angeschaut. Dieses Tool bietet genau was ich brauche und wird mich beim Aufbau der Seite unterstützen.

### Tür Schaltungssensor Wert

Die Werte des Adeunis RF und der Elsys CO2 Sensoren konnte ich bereits als Vorarbeit dekodieren. Für den Tabs Sensor reichte die Zeit nicht mehr.

Gemäss diesem [Datenblatt](https://iot-shop.de/wp-content/uploads/2020/03/RM_Door-_-Window-Sensor_20200205_v2.pdf) besteht der Wert eines Tabs Sensors aus 8 bytes.

|  |  |  |  |
| --- | --- | --- | --- |
| Wert | Byte | Bits | Beschreibung |
| Status | 0 | [0] | 1 – offen, 0 – geschlossen |
| Batterie | 1 | [3:0] | Bereich 1 – 14  Batterie Spannung in V = (25 + ν) ÷ 10. |
| Temperatur (PCB) | 2 | [6:0] | Bereich 1-127  Für Grad noch -32 rechnen. |
| Zeit | 3, 4 | [15:0] | Bereich 0 - 65535 |
| Zählstand | 5, 6, 7 | [23:0] | Bereich 0 - 16777215 |

# Planen

## Use Case



Abbildung 9 Use Case

## System

### Allgemein

Das System ist so aufgebaut, dass ein Benutzer zuerst die statische Webseite von dem IIS Server ladet und danach alle Daten von der API, welche von [Uvicorn](https://www.uvicorn.org/) gehostet wird, holt. Beide Dienste werden auf dem internen Server SI1010023 gehostet. Ein solcher Vorgang erlaubt eine angenehmere Benutzung der Webseite, denn man muss nicht warten bis zum Beispiel PHP alle Daten in das HTML gerendert hat, sondern man bekommt direkt die Webseite mit einer kleinen Ladeanimation und weiss bereits, dass etwas passiert, danach geht es meistens schnell bis die Daten von der API geladen und dargestellt werden. Zudem muss man nicht jedes Mal eine Seite neu laden um Daten abzuspeichern, denn alle Zugriffe auf die API werden mit Ajax Aufrufe ausgeführt.

Die wichtigsten Komponenten des Systems ist der Uvicorn Server, die PostgreSQL Datenbank und das Skript, welches von Loriot die Sensor Daten erhält. Der IIS könnte man durch einen Apache oder Nginx Server austauschen. IIS ist jedoch Standard auf Windows Servern.

### Uvicorn Server

Der Uvicorn Server erhält alle Anfragen eines Benutzers und leitet diese weiter an mein Backend, dieses verarbeitet die Anfrage, lädt vielleicht ein paar Daten von der Datenbank und gibt letztendlich eine Antwort zurück. Diese Antwort wird dann vom Uvicorn Server dem Benutzer zurückgeschickt.

### Websockets

Loriot ist eine Webseite, welche alle Daten der angehängten Sensoren sammelt. Man kann sich per Websockets mit diesem Verbinden und mithören, wenn neue Sensordaten eingefangen werden. Diese werden dann gleich in die Datenbank geschrieben.

### PostgreSQL Datenbank

PostgreSQL ist ein leistungsstarkes, objektrelationales Open-Source-Datenbanksystem. Diese Datenbank ist Zuverlässig und Leistungsstark.

### Visuell

Damit man sich das ganze vorstellen kann, ist das System unten aufgeführt Visuell dargestellt.



Abbildung 10 System Aufbau

## Sensoren

Für den Pilot Versuch können drei verschiedene Sensoren benutzt werden.

* [Adeunis RF](https://www.adeunis.com/en/produit/ftd-network-tester/)
* [Elsys ERS CO2](https://www.elsys.se/shop/product/ers-co2/?v=1ee0bf89c5d1)
* [Tabs Tür Sensor](https://iot-shop.de/produkt/tuer-und-fenstersensor)

Für jeden dieser Sensoren muss je ein Dekodierung Algorithmus geschrieben werden. Für den Adeunis RF und Elsys ERS CO2 wurde dies bereits im Vorfeld gemacht.

Damit man weiss welcher Sensor von welchem Skript dekodiert werden soll, muss eine Verlinkung, welche von den Benutzern der Webseite erstellt werden kann, gemacht werden. Jeder Sensor hat eine eindeutige EUI, welche für die Verlinkung perfekt ist.

## Wert Beobachter

Jedem Sensor soll man mehrere Wert Beobachter anhängen können, da ein Sensor auch mehrere Werte messen kann.

Es soll zwei Arten von Wert Beobachtern geben.

1. Richtwert Beobachter

Beim Richtwert wird eine Meldung ausgegeben, wenn der Sensor diesen Richtwert überschreitet. Als Beispiel kann man einen Temperatur Sensor nehmen. Setzt man den Richtwert auf 22 Grad und der Sensor meldet eine Temperatur von 24, so würde eine Meldung ausgehen.

1. Zählerstand Beobachter

Beim Zählerstand soll gemessen werden wie oft etwas gemacht wurde. Dies ist perfekt für die Messung wie oft eine Toilette benutzt wurde. So könnte man nach 10 Benutzungen eine Meldung ausgeben.

## Authentifizierung

Die Authentifizierung wird per Access und Refresh Tokens funktionieren. Nachdem ein Benutzer mit seinen Login Daten sich einloggte, erhält er je eine der beiden genannten Tokens.

**Access Token**

Dieser Schlüssel hat eine kurze Lebensdauer (15 – 30 min) und beinhaltet Informationen über den Benutzer, diese Informationen können die Rolle und ID sein. Der Token hat eine Signatur angehängt, welche versichert, dass dieser nicht verändert wurde. Bei jedem Aufruf auf die API wird dieser Token mitgeschickt und auf Alter und Signatur geprüft. Der Vorteil ist, dass die API nicht bei jedem Aufruf eine Datenbank Abfrage durchführen muss, sondern direkt die Rolle aus dem Token herauslesen kann.

**Refresh Token**

Der Refresh Token ist ein kryptografisch sicherer zufällig generierter String. Dieser kann dazu verwendet werden, einen neuen Access Token anzufordern. Bei jeder Anfrage für einen neuen Access Token sollte die Aktivität des Benutzers auf Verdächtige Aktionen überprüft werden. Ein Refresh Token ist gleich viel Wert wie ein Login und muss daher sicher gespeichert werden.

Die Authentifizierung hat somit folgenden Ablauf. Das Diagramm ist von links nach rechts zu lesen.



Abbildung 11 Authentifizierung Ablauf

## Frontend

### Admin Dashboard

In der Admin Dashboard Ansicht sollen Administratoren sehen, welche Räume welche Meldungen haben. Eine Meldung kann automatisch von einem Wert Beobachter ausgelöst werden oder auch Rückmeldungen des Reinigungspersonals sein. Alle Meldungen eines Raums können als Gelöst markieren, was als Effekt hat, dass diese in eine andere Liste verschoben werden und nicht mehr auf dem Dashboard erscheinen, oder gelöscht werden, was dazu führt, dass die Meldung aus der Datenbank gelöscht wird.



Abbildung 12 Admin Dashboard

### Personal Verwaltung

Auf dieser Seite können Administratoren das Personal verwalten. Benutzer können entweder die Rolle Personal oder Admin haben.



Abbildung 13 Personal Seite

### Räume Verwaltung

In der Räume Verwaltung kann das Gebäude aufgebaut werden. Räume werden in Stockwerken erstellt welche in einem Gebäude erstellt wurden. So gibt es für die Zukunft die Möglichkeit die Applikation auf mehrere Gebäude zu verteilen.



Abbildung 14 Räume Verwaltung

### Raum Ansicht

In der Raum Ansicht sieht man alle Sensoren und Meldungen eines Raums.

Abbildung 15 Raum Ansicht

### Sensoren Verwaltung

In der Sensoren Verwaltung soll die Art der Sensoren eingesehen werden können und die Sensoren selbst.

Für jede Art von Sensor soll definiert werden können, welches und wie viel Material für eine Meldung vorgeschlagen werden soll.

Beim Erstellen eines Physischen Sensors muss unbedingt eine EUI und Art des Sensors mitgegeben werden. Die Art des Sensors bestimmt welcher Dekodierung Algorithmus verwendet werden soll. Zudem soll dem Sensor mehrere Wert Beobachter angehängt werden können.



Abbildung 16 Sensoren Verwaltung

## Sensor Ansicht

In der Sensor Ansicht können die Wertbeobachter verändert werden oder die Sensordaten eingesehen werden.



Abbildung 17 Sensor Ansicht

### Reinigungspersonal Dashboard

Der Sinn dieses Dashboards ist, dass das Reinigungspersonal sich eine Route vorstellen kann. Die Räume mit Meldungen sind nach Gebäude und Stockwerk aufgelistet. Nachdem ein Raum geputzt wurde, kann der Raum als gelöst markiert werden. Auch hier verschwinden dann die Meldungen in eine separate Liste. Wenn gelöst geklickt wird, soll auch gleich gefragt werden, ob noch eine Rückmeldung gegeben werden möchte. Eine solches Feedback kann Schadensmeldungen beinhalten oder ein simples ok.

Damit das Reinigungspersonal auch weiss, wie viel Material mitgenommen werden soll, wird zuoberst auf der Seite noch eine vorgeschlagene Materialliste angezeigt. Diese soll Anhand der offenen Meldungen berechnet werden.



Abbildung 18 Dashboard Reinigungspersonal

## Konzeptionelles Modell



Abbildung 19 Konzeptionelles Model

## Logisches Modell



Abbildung 20 Logisches Modell

## API Struktur

Die API Struktur definiert auf welchen URLs, mit welchen http Methoden, welche Aktion ausgeführt wird.

Die http Methoden sind GET, POST, PUT und DELETE.

* GET wird für das Abrufen von Daten verwendet.
* POST ist für die Erstellung von Daten
* PUT ist für die Aktualisierung von Daten
* DELETE ist für die Endgültige Löschung von Daten.

|  |  |  |
| --- | --- | --- |
| Pfad | Methode | Aktion |
| /login | POST | Entnimmt Login Daten aus dem Request Body und antwortet mit dem Access Token und Refresh Token. |
| /login | PUT | Entnimmt Refresh Token aus dem Request Body und antwortet mit aktualisiertem Access Token. |
| /login | DELETE | Macht den Refresh Token des Benutzers ungültig. |
| /personal | GET | Auflistung des Personals |
| /personal | POST | Erstellt eine neue Person und Login aus den Daten des Request Body. |
| /personal/<id> | PUT | Aktualisiert Person mit den Daten des Request Body. |
| /personal/<id> | DELETE | Löscht Person aus der Datenbank. Alle Meldungen dieser Person werden somit auch gelöscht. |
| /gebaeude | GET | Auflistung der Gebäude |
| /gebaude | POST | Erstellung eines Gebäude |
| /gebaude/<id> | PUT | Aktualisierung eines Gebäude |
| /gebaude/<id> | DELETE | Löschung eines Gebäude |
| /stockwerke | GET | Auflistung der Stockwerke |
| /stockwerke | POST | Erstellung eines Stockwerks |
| /stockwerke /<id> | PUT | Aktualisierung eines Stockwerks |
| /stockwerke /<id> | DELETE | Löschung eines Stockwerks |
| /raume | GET | Auflistung der Räume |
| /raume | POST | Erstellung eines Raums |
| /raume /<id> | PUT | Aktualisierung eines Raums |
| /raume /<id> | DELETE | Löschung eines Raums |
| /raume/<id>/sensoren | GET | Auflistung aller Sensoren eines Raumes |
| /raume/<id>/meldungen | GET | Auflistung aller Meldungen eines Raumes |
| /sensoren | POST | Erstellung eines Sensors für den Raum mit ID |
| /sensoren/<eui> | PUT | Aktualisierung eines Sensors |
| /sensoren/<eui> | DELETE | Löschung eines Sensors |
| /sensoren/<eui> | GET | Auflistung aller Werte eines Sensors |
| /sensoren/<eui>/beobachter | GET | Auflistung aller Beobachter eines Sensors |
| /beobachter | POST | Erstellung eines Beobachter für einen Sensor |
| /beobachter/<id> | PUT | Aktualisierung eines Beobachters |
| /beobachter/<id> | DELETE | Löscht einen Beobachter |
| /sensoren/<eui>/meldungen | GET | Auflistung aller Meldungen eines Sensors |
| /beobachter/<id>/meldungen | GET | Auflistung aller Meldungen eines spezifischen Beobachters |
| /beobachter/<id>/meldungen | POST | Manuelle Auslösung einer Meldung |
| /materialien | GET | Auflistung aller Materialien |
| /materialien | POST | Erstellung eines Material |
| /materialien/<id> | PUT | Aktualisierung eines Material |
| /materialien/<id> | DELETE | Löschung von Material |
| /beobachter/<id>/maeterialien | POST | Material zu Beobachter hinzufügen |
| /beobachter/materialien/<id> | DELETE | Entfernen von Material von Beobachtern |
| /views/dashboard | GET | Antwortet mit allen nötigen Daten für das Dashboard je nach Rolle |
| /views/personal | GET | Antwortet mit allen nötigen Daten für die Personal Verwaltung |
| /views/personal/<id> | GET | Antwortet mit allen nötigen Daten für die Person Ansicht |
| /views/raeume | GET | Antwortet mit allen Daten für die Räume Verwaltung |
| /views/raum/<id> | GET | Antwortet mit allen Daten für die Raum Ansicht |
| /views/sensoren | GET | Antwortet mit allen Daten für die Sensoren Verwaltung |
| /views/sensoren/<eui> | GET | Antwortet mit allen Daten für eine Sensor Ansicht. |

## Sicherheit

### Feld Validierung

Allgemein müssen alle Felder auf Inhalt und Länge validiert werden. Alle Eingaben eines Benutzers sollten von Tags und dergleichen bereinigt werden.

### SQL Injection

SQL Abfragen enthalten keine Rohdaten der Benutzer. Die Werte für die Abfrage werden separat der Datenbank geschickt und von PostgreSQL überprüft. Somit ist es nicht möglich, Eingabefelder zu missbrauchen.

### Cross Site Scripting

Damit Cross Site Scripting nicht möglich ist, wird der Meta Tag Content-Security-Policy so konfiguriert, dass keine inline Scripts und nur JavaScript Files von derselben Domäne erlaubt sind. Der einzige Nachteil hier ist, dass man keinen CDN für JQuery oder dergleichen benutzen kann. So verhindert man aber auch das Schadcode von einem gehackten CDN auf die eigene Seite geladen wird.

## Testen

### Testmethoden

Beim Testen wird zwischen Unit Tests und Integration Tests unterschieden. Alle unten aufgeführten Tests sind Integration Tests. Gut zu wissen ist, dass Integration Tests prüfen mehrere verknüpfte Funktionen auf einmal, während Unit Tests nur eine einzige Methode testet. Die Unit Tests sind eher für das Backend und sollen für alle Public Methoden geschrieben werden.

### Testkonzept

|  |  |
| --- | --- |
| Testnummer | Testgebiet |
| 1 | Authentifizierung |
| 2 | Sensor Wert Dekodierung |
| 3 | Sensor Wert Beobachter |
| 4 | Gebäude Verwaltung |
| 5 | Eingabe Validierung |
| 6 | Design und Browserkompatibilität |

### Testfallspezifikationen

#### Authentifizierung

|  |  |  |  |
| --- | --- | --- | --- |
| Testfallnummer | Beschreibung | Voraussetzungen | Eingabe |
| 1.1 | Meldet sich ein Benutzer an, so erhält er eine Access Token und Refresh Token. | Das Backend und Frontend muss gestartet sein. Login Seite muss geöffnet sein. | Korrekter Benutzername und Passwort eingeben. |
| 1.2 | Sendet der Benutzer ein Refresh Token, so erhält er einen frischen Access Token. | Das Backend und Frontend muss gestartet sein. Erstes Login muss bereits getätigt sein. Warten auf Ablauf des ersten Access Token. | Neuer Aufruf auf die API. Es wird danach automatisch erkannt, dass der Refresh Token gesendet werden muss. |
| 1.3 | Bei invaliden Anmelde Daten ergibt sich einen Fehler. | Das Backend und Frontend muss gestartet sein. Login Seite muss geöffnet sein. | Falscher Benutzername und falsches Passwort eingeben. |
| 1.4 | Bei falschem Refresh Token ergibt sich einen Fehler. | Das Backend und Frontend muss gestartet sein. Benutzer muss eingeloggt sein. Refresh Token muss fälschlicherweise ausgetauscht werden | Aufruf auf die API ausführen. |

#### Sensor Wert Dekodierung

|  |  |  |  |
| --- | --- | --- | --- |
| Testfallnummer | Beschreibung | Voraussetzungen | Eingabe |
| 2.1 | Bei Erhalt eines Sensor Wertes wird die richtige Dekodierung ausgewählt und der Wert in JSON umgewandelt. | Loriot Gateway aktiv und verbunden mit Applikation über Websockets. Datenbank geöffnet (pgAdmin). | Druck auf Button des Adeunis RF Sensors. |
| 2.2 | Ein Wert des Adeunis RF Sensors wird korrekt in folgende Werte umgewandelt.   * Temperatur vorhanden * Übertragung durch Klick ausgelöst * Übertragung durch Bewegung ausgelöst * GPS vorhanden * Uplink Counter * Downlink Counter * Batterie Status * - (RSSI, SNR) | Loriot Gateway aktiv und verbunden mit Applikation über Websockets. Datenbank geöffnet (pgAdmin). | Klick auf Adeunis RF. |
| 2.3 | Ein Wert eines Elsys CO2 Sensors wird Korrekt in folgende Werte umgewandelt.   * Temperatur * Feuchtigkeit * Licht * Bewegung * CO2 * Batterie Spannung | Loriot Gateway aktiv und verbunden mit Applikation über Websockets. Datenbank geöffnet (pgAdmin). | Warten auf neue Daten des Sensors. Elsys schickt alle 5 Minuten. |
| 2.4 | Ein Wert eines Tabs Sensors wird Korrekt in folgende Werte umgewandelt.   * Status * Batterie Spannung * Time * Zähler * - (Temp PCB) | Loriot Gateway aktiv und verbunden mit Applikation über Websockets. Datenbank geöffnet (pgAdmin). | Den Status eines Sensors ändern in dem man den Magnet wegnimmt. |

#### Sensor Wert Beobachter

|  |  |  |  |
| --- | --- | --- | --- |
| Testfallnummer | Beschreibung | Voraussetzungen | Eingabe |
| 3.1 | Es können einem Sensor einen Wert Beobachter zugefügt werden. | Web Applikation mit Backend gestartet. Sensor Werte von Loriot abhören. | Wert Beobachter erstellt. Richtwert oder Zählerstand. |
| 3.2 | Ein Richtwert Beobachter erstellt eine Meldung für den Elsys CO2 Temperatur Wert, wenn sich dieser um 1 Grad erwärmt hat. | Web Applikation mit Backend gestartet. Sensor Werte von Loriot abhören. Elsys CO2 Sensor muss hinzugefügt sein. | Beobachter hinzufügen mit einem Richtwert ein Grad wärmer als die zuletzt gemeldete Wärme. Danach Sensor in mit der Hand erwärmen. |
| 3.3 | Ein Zählerstand Beobachter erstellt eine Meldung wenn der Zählerstand eines Adeunis RF zehn erreicht. | Web Applikation mit Backend gestartet. Sensor Werte von Loriot abhören. Adeunis Sensor hinzugefügt mit Zählerstand Beobachter. | Zehn Mal auf den Knopf des Sensors drücken. |
| 3.4 | Es können mehrere Beobachter für den Elsys CO2 erstellt werden. Einen für die Temperatur und einen für das Licht. | Web Applikation mit Backend gestartet. Sensor Werte von Loriot abhören. Elsys CO2 Sensor hinzugefügt und Richtwert Beobachter für Licht und Temperatur erstellt. | Sensor erwärmen und abdunkeln. |

#### Gebäude Verwaltung

|  |  |  |  |
| --- | --- | --- | --- |
| Testfallnummer | Beschreibung | Voraussetzungen | Eingabe |
| 4.1 | Es kann ein Gebäude erstellt werden. | Web Applikation mit Backend gestartet. | Gebäude Name eingeben und erstellen. |
| 4.2 | Es können Stockwerke einem Gebäude hinzugefügt werden | Web Applikation mit Backend gestartet. Mindestens ein Gebäude erstellt. | Gebäude auswählen und Stockwerk Name, Niveau eingeben. |
| 4.3 | Es können Räume einem Stockwerk hinzugefügt werden. | Web Applikation mit Backend gestartet und mindestens ein Stockwerk bereits erstellt. | Raum Name eingeben und Stockwerk auswählen. |
| 4.4 | Räume werden nach Stockwerk Niveau und Gebäude sortiert. | Web Applikation mit Backend gestartet und Raum Verwaltung geöffnet. | Keine. |

#### Eingabe Validierung

|  |  |  |  |
| --- | --- | --- | --- |
| Testfallnummer | Beschreibung | Voraussetzungen | Eingabe |
| 5.1 | Leere Eingaben werden abgefangen. Leer ist auch ein String der Länge 0 und nicht nur None. | Web Applikation mit Backend gestartet. | Zum Beispiel ein neues Gebäude erstellen mit leerem Name |
| 5.2 | JavaScript Code in Eingaben werden nicht ausführbar in der Datenbank abgespeichert. | Web Applikation mit Backend gestartet. | Neues Gebäude erstellen mit dem Namen <script>alert(«Not Safe»);</script> |
| 5.3 | Gibt man einen Gebäudenamen ein welcher länger als 100 Charakters ist, so wird dieser abgelehnt. | Web Applikation mit Backend gestartet. | 101 Charakter String eingeben als Gebäudenamen. |

#### Design und Browserkompatibilität

|  |  |  |  |
| --- | --- | --- | --- |
| Testfallnummer | Beschreibung | Voraussetzungen | Eingabe |
| 6.1 | Das Design passt sich für den Desktop an. | Applikation auf einem Monitor mit der Grösse 1920x1080 geöffnet. | Keine. |
| 6.2 | Das Design passt sich für das Tablet an. | Applikation auf einem Monitor mit der Grösse 1024x768 geöffnet. | Keine. |
| 6.3 | Die Webseite wird im Edge unterstützt. | Applikation in genannten Browser geöffnet. | Durch die Seite navigieren. Auf fehlende Anzeigen achten. |
| 6.4 | Die Webseite wird im Firefox unterstützt. | Applikation in genannten Browser geöffnet. | Durch die Seite navigieren. Auf fehlende Anzeigen achten. |
| 6.5 | Die Webseite wird im Chrome unterstützt. | Applikation in genannten Browser geöffnet. | Durch die Seite navigieren. Auf fehlende Anzeigen achten. |
| 6.6 | Werden die Farben und Schriften gemäss BBB\_CD\_Manual.pdf eingehalten. | Applikation geöffnet. | Keine. |
| 6.7 | Die Luftqualitäts-Messwerte sollen korrekt dargestellt | Sensor Ansicht eines Elsys CO2 Sensors geöffnet. | Keine. |
| 6.8 | Verfügt die grafische Darstellung der der Messwerte über einen Zeitfilter? | Applikation gestartet. Sensor Ansicht geöffnet. | Zeitfilter betätigen. |

## Vorgehen

Nun das alle Teilelemente des Projektes bekannt sind, kann das gröbere Vorgehen beschrieben werden.

Zuerst wird das Backend vollständig realisiert, dies beinhaltet auch Unit Tests. Folgend werden alle API Aufrufe in JavaScript umgesetzt und letztendlich wird das Frontendumgesetzt, welches mit dem zuvor umgesetzten JavaScript Code sich mit dem Backend verbindet.

# Entscheiden

## Allgemein

Es steht nur noch die Entscheidung für die JavaScript Diagramm Bibliothek offen. Die Optionen sind Chart.js und Chartist.js. Abgesehen von der Vielfältigkeit haben die zwei Bibliotheken keine grösseren Unterschiede. Daher wird eine Nutzwertanalyse erstellt.

## Kriterien

Die zu bewertenden Aspekte sind:

* Einfachheit der Anwendung

Je einfacher die Bibliothek anzuwenden ist, desto schneller werde ich vorankommen.

* Unterstützt Darstellung von verschiedenen Werten in einem

Da ein Sensor nicht nur ein Wert misst, wäre es von Vorteil könnte die Bibliothek mehrere Werte auf einem Diagramm darstellen.

* Vielfältigkeit

Je mehr «Out oft he box» funktioniert, desto weniger muss selbst geschrieben werden.

* Grösse der Bibliothek

Muss eine Webseite weniger laden, so ist sie auch schneller. Da sich aber solche Bibliotheken cachen lassen, spielt dies keine grosse Rolle.

* Browserkompatibilität

Chrome, Firefox und Edge müssen unterstützt sein.

* Responsiveness

Ein Muss für Tablets.

## Nutzwertanalyse



Abbildung 21 Nutzwertanalyse

Aus der Nutzwertanalyse ergibt sich das Chart.js der Sieger ist. Auch wenn ich anfangs Probleme mit der Grösse der Diagramme hatte, bietet Chart.js mehr als Chartist.js und wird somit mit weniger Aufwand hilfreicher sein und für eine solche kurze Zeitspanne in welcher meine IPA läuft, wird sich dies auszahlen.

# Realisieren

## Backend

### Backend Aufsetzen

#### Quellcode

Am einfachsten installiert man sich das GIT Repository. Der Quellcode ist unter realisieren/backend gespeichert.

#### Datenbank

Falls man PostgreSQL nicht installiert hat, sollte man dies zusätzlich noch tun. Damit man die Datenbank anschauen kann, sollte man PgAdmin gleich mit installieren.

Danach gibt es das «db\_setup.py» Skript im Repository unter realisieren/backend/ mit welchem man alle Tabellen gemäss dem logischen Modell erstellen lassen kann. Dieses File kann man normal mit Python ausführen.

#### Python Packages

Damit man alle benötigten Python Bibliotheken installiert hat, kann man folgende Befehlen im realisieren/backend Ordner ausführen.

* python setup.py sdist
* pip install dist/bbbapi-1.0.tar.gz

#### Server starten

Letztendlich kann man den Server mit ‘uvicorn main:app’. Auch hier muss man wieder im realisieren/bbbapi Ordner sein. Nun ist der ASGI Server gestartet, somit läuft das Backend.

Damit die Authentifizierung reibungslos funktioniert, müssen noch die Public und Private Keys im RSA Format erstellt werden. Diese werden gebraucht um die Signatur eines Access Tokens zu erstellen und validieren. Zusätzlich müssen die Dateinamen des Public und Private Keys identisch sein und eine Länge welche durch sechzehn teilbar ist haben. Dies ist so, weil im Header eines Access Tokens die Public Key Kennung vorhanden sein muss, damit das Backend weiss mit welchem Schlüssel die Signatur validiert werden kann. Im config.ini File sind die Schlüssel unter assets/keys/public und assets/keys/private verlinkt. Mehr über die Konfiguration folgt gleich hierunter.

### Konfiguration

Das Backend lässt sich natürlich auch minimal konfigurieren, die Werte und ihre Auswirkung sind in folgender Tabelle beschrieben.

Das Format von config.ini sieht folgenerweise aus:

[TITEL]  
name=wert   
[KEYS]   
private-keys=assets/keys/private

#### Keys

|  |  |
| --- | --- |
| Name | Beschreibung |
| private-keys | Pfad zu den privaten RSA Schlüsseln |
| public-keys | Pfad zu den passenden RSA öffentlichen Schlüsseln |
| Identifier-secret | 32 Charakter langer String welcher den Dateinamen von privaten Schlüsseln verschlüsselt damit er ohne Erkennungsproblem in den Header eines Access Token geschrieben werden kann. |

#### Token

|  |  |
| --- | --- |
| Name | Beschreibung |
| expire | Zeit in Sekunden bis ein Access Token nach Erstellung abläuft. |
| issuer | Domain des Servers welcher den Access Token erstellt hat. |
| audience | Domain des Servers für welchen einen Access Token anwendbar ist. |
| refresh-token-bytes | Grösse des zu erstellenden Refresh Token in bytes. |

#### Scrypt

Scrypt ist der Algorithmus mit welchem die Passwörter gehashed werden.

|  |  |
| --- | --- |
| Name | Beschreibung |
| mem\_cost | Wie viel Memory die Generierung des Hashes benutzen sollte. Hierbei sollte ein Wert von 16 genügen. |
| rounds | Wie oft gehashed werden soll. Hierbei sollte ein Wert von 8 genügen. |

#### DB\_CREDENTIALS

|  |  |
| --- | --- |
| Name | Beschreibung |
| database | Name der Datenbank |
| user | Benutzername des Datenbank Benutzers |
| password | Passwort des Datenbank Benutzers |

#### DB

|  |  |
| --- | --- |
| Name | Beschreibung |
| close-timeout | Zeit in Sekunden wie lange gewartet werden soll bis die Verbindung zur Datenbank terminiert werden soll nach Schliessung des Servers. |

#### ASGI

|  |  |
| --- | --- |
| Name | Beschreibung |
| max-body-size | Grösse in Bytes welche ein Aufruf auf die Datenbank nicht überschreiten darf. |

### Tabs Sensor Dekodierung

#### Status

Für den Tabs Sensor war es nur wichtig zu wissen, ob die Türe offen oder geschlossen ist.

Ein Sensor Wert kann folgenderweise Aussehen ‘00fc35fa25060000’. Dieser steht in Hex. Wenn wir dem Datenblatt folgen, wissen wir, dass das erste Byte den Status wiederspiegelt. Ein Byte hat acht Bits. Ein Charakter in Hex ist ein Nibble und ein Nibble ist 4 Bits. Somit entsprechen die ersten zwei 0 unserem Status.

Ausgeschrieben in Bits entspricht dies 0000 0000, unserem Byte. Wäre die letzte null eine 1, so wäre die Türe offen. Dies sähe so aus 0000 0001 und in einem Sensor Wert Beispiel ‘01fc35fa25060000’.

### Loriot Listener

Der Loriot Listener ist das Skript, welches die Daten der Sensoren von Loriot holt, dekodiert und die angehängten Beobachter des Sensors informiert. Das Skript ist so geschrieben, dass es sich bei Errors nach 5 Sekunden automatisch neu startet. Dies ist eine einfache Vorbeugung, falls die Verbindung zu Loriot unterbrochen wird. Das Skript besteht aus 3 Hauptfunktionen. Main, Listen und Handle.

* **Main** verbindet sich mit der Datenbank und dient als Wrapper für Listen. Passiert in Listen ein Fehler, so wird Listen nach 5 Sekunden wieder erneut aufgerufen.
* **Listen** verbindet sich per Websocket mit Loriot und empfängt die Rohen Sensor Daten und gibt diese den passenden Dekodierer. Damit das Skript weiss, welcher Dekodierer verwendet werden soll, muss der Sensor in der Sensoren Tabelle erscheinen. In dieser Tabelle gibt es die Spalte Art, welche aussagt, was für ein Sensor es ist. Der Sensor wird durch die Device EUI identifiziert, diese erhält man mit den Rohdaten. Nach der Dekodierung wird der Sensor und Wert der Handle Methode übergeben.
* **Handle** lädt zuerst die Daten in die Sensor Wert Tabelle und alarmiert danach die Beobachter, welche in der Beobachter Tabelle zum Sensor mit passender Device EUI referenziert sind.

Vereinfacht sieht dies folgenderweise aus:



Abbildung 23 Loriot Listener

### Modelle

Ein Modell besteht aus Feldern und repräsentiert eine Reihe aus einer Datenbank Tabelle.

Als Beispiel ein Screenshot des Gebäude Modell.

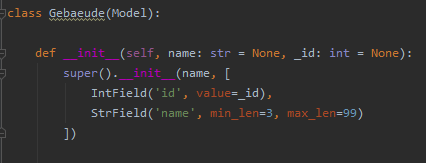


Abbildung 24 Gebäude Modell

Es besteht aus einer ID und einem Namen, der Name wird beim einlesen von Benutzerdaten auf die Mindestlänge von 3 und Maximum Länge von 99 geprüft. Modelle können auch verschachtelt werden, so wird ein Foreign Key zu einem Modell.

So wurde beim Stockwerk, anstatt von einem IntField für den Fremdschlüssel für idGebaeude ein Gebäude Modell eingefügt.

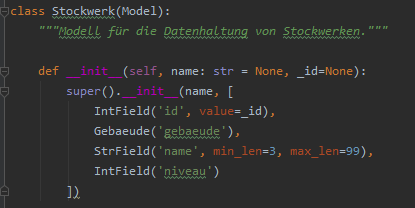


Abbildung 25 Stockwerk Modell

Daten werden von der Datenbank in der Form eines Dicts zurückgegeben. Als Keys für diese Dicts kann man einen Key Pfad setzen. Solche sehen wie folgt aus für ein Stockwerk: ‘id’, ‘name’, ‘niveau’, ‘gebaeude.id’ und ‘gebaeude.name’. Einen solchen Pfad kann man ablaufen und den Wert des Feldes setzen lassen.

Hätte man nun eine Instanz eines Stockwerk Modells, so würde ‘gebaeude.id’ die ID des Gebaeude Feldes setzen.

Setzt man also die Schlüssel des Antwort Dicts der Datenbank mit Key Pfaden, so kann man die Werte automatisch in das Modell einlesen lassen.

Beispiels Abfrage mit Gelb angestrichenen Werte als Key Pfad.

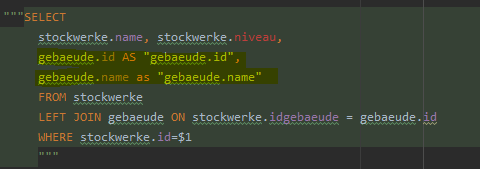


Abbildung 26 Beispiel Abfrage mit Key Pfaden

Das Resultat dieser Abfrage würde folgenderweise aussehen und automatisch anhand von einer Methode, welche die Key Pfade abläuft, eingelesen werden

{

‘name’: ‘Stockwerk Name’,

‘niveau’: 0,

‘gebaeude.id’: 0,

‘gebauede.name’: ‘Gebäude Name’

}

**Implementierte Modelle**

* Beobachter
* Gebäude
* Material
* Meldung
* Personal
* Raum
* Sensor
* Stockwerk

### Modell Controller

Ein Model Controller ist zuständig für die Erstellung, Aktualisierung und Löschung von Modellen. Vor jeder Aktion werden die Modells auf Inhalt überprüft.

Die Modell Controller verfügen über die einfachen Methoden Create, Update und Delete.

**Implementierte Modell Controller**

* Beobachter
* Gebäude
* Material
* Meldung
* Personal
* Raum
* Sensor
* Stockwerk

### Auflistungen

Für die Auflistung von Modellen wird die ListController Klasse benutzt. Diese Teilt die Daten in der Tabelle auf, so dass nicht alles auf einmal geholt werden muss.

So sieht die Klasse für die Gebäude Auflistung aus.

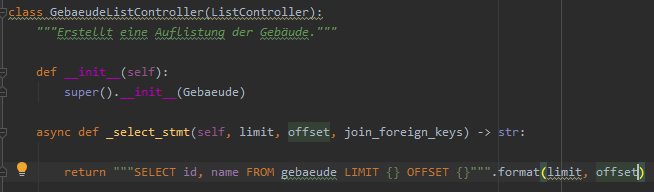


Abbildung 27 List Controller Beispiel

Die List Controller Klasse ist so implementiert, dass man nur noch die Methode \_select\_stmt überschreiben muss. Hierbei ist jedoch wichtig, dass man das Limit und Offset nicht in der Abfrage vergisst, ansonsten würde man jedes Mal die gleichen Daten überkommen.

**Implementierte Auflistungen**

* Gebäude
* Personal
* Sensoren
* Stockwerk
* Räume

### Beobachter

Die Beobachter sind dafür zuständig die Sensor Werte auf gewisse Bedingungen zu prüfen. Damit die Beobachter nicht ständig eine Meldung ausgeben, haben sie ein Timeout von einer Stunde (konfigurierbar), bis sie eine neue Meldung ausgeben können.

#### Richtwert Darüber

Dieser misst, ob ein gewisser Wert einen definierten Wert überschritten hat. Beispiel:

Der Beobachter misst den Wert Temperatur eines Adeunis Sensors und gibt eine Meldung, wenn dieser den Wert 30 überschritten hat. Wenn der Sensor nun einen Wert von 31 meldet, wird eine Meldung ausgegeben.

#### Richtwert Darunter

Dieser Beobachter ist einfach das Gegenteil des letzten Beobachters, anstatt das der Wert darüber liegen muss, muss dieser nun darunter liegen. Beispiel:

Der Beobachter misst den Wert Temperatur eines Adeunis Sensors und gibt eine Meldung, wenn dieser den Wert von 18 unterschritten hat. Wenn der Sensor nun einen Wert von 17 meldet, wird eine Meldung ausgegeben.

#### Zählerstand

Der Zählerstand misst wie oft der Sensor sich meldet und inkrementiert seinen Stand, überschreitet nun der Sensor einen definierten Wert mit seinem Stand, so wird eine Meldung ausgegeben und der Stand wird auf null zurückgesetzt. Beispiel:

Der Beobachter zählt den Stand des Adeunis Sensors und soll nach der zehnten Nachricht eine Meldung ausgeben. Nun hat sich der Sensor zehn Mal gemeldet, und egal welche Werte er meldete, wird eine Meldung ausgegeben.

#### Beobachter mit Sensor verknüpfen

Zuerst erstellt man ein Beobachter Modell, setzt die Sensor Dev EUI auf den des gewünschten Sensors und erstellt danach das Modell mit dem Beobachter Controller. Dies könnte folgenderweise aussehen.

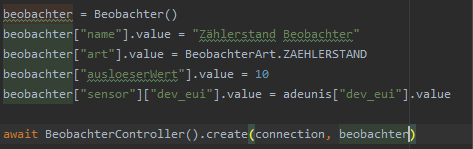


Abbildung 28 Beobachter erstellen

Mit diesem Zählerstand Beobachter wird nach zehn Nachrichten eines des Adeunis Sensors eine Meldung ausgeben.

#### Beobachter hinzufügen

Möchte man einen neuen Beobachter hinzufügen, so ist es wichtig, dass man zuerst einmal die Enumeration BeobachterArt in common\_types.py erweitert. Danach erweitert man die Klasse BeobachterInterface und implementiert die Methode watch. In dieser Methode kann man die Werte des Sensors überprüfen auf die erwünschten Bedingungen und wenn nötig eine Meldung ausschreiben. Danach muss man nur noch im loriot\_listener.py Skript folgender variable mit einer Instanz des Beobachters erweitern.

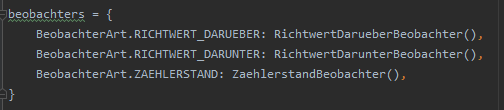


Abbildung 29 Loriot Listener Beobachters

Nun jedes Mal, wenn ein Sensor einen Wert schickt, werden die Beobachter aus der Beobachter Tabelle mit der richtigen Sensor Dev EUI ausgelesen und anhand von der oben gezeigten Variable und der Spalte ‘art’ in der Tabelle zu den korrekten Instanzen geleitet.

### Resources

Eine Resource nimmt jeweils die Anfrage eines Benutzers entgegen, prüft den Benutzer auf Rolle und bearbeitet die gewünschten Aktionen anhand von Controllern.

Für Modell Controllern und List Controllern gibt es bereits vom benutzen Framework her Resources welche die Benützung dieser Vereinfacht.

Als Beispiel wird hier die Resource dargestellt, welche dafür zuständig ist, ein Material einem Beobachter zu zuweisen. Dieses Material wird dann jedes Mal vorgeschlagen, wenn dieser Beobachter eine Meldung ausgibt.

Die Resource hat eine on\_post und on\_delete Methoden, hier wird die on\_post gezeigt.

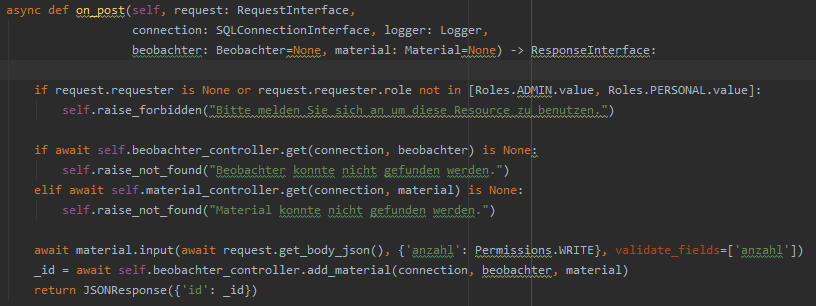


Abbildung 30 on\_post Material zu Beobachter Resource

Die Resource prüft zuerst ob der Benutzer welcher die Anfrage geschickt hat überhaupt über die notwendigen Rechte besitzt, diese entsprechen hier der Rollen ADMIN und PERSONAL. Danach wird geschaut, dass der Beobachter und das Material auch wirklich existieren und folgend wird noch der JSON im Body der Anfrage in das Material Modell eingelesen. Dabei wird auch gleich geprüft, dass das Feld Anzahl nicht leer ist. Zu aller Letzt wird dann der Eintrag in die MaterialZuBeobachter Tabelle anhand des Beobachter Controllers geschrieben. Als Antwort schickt die Resource die ID des neuen Eintrags, damit diese auch falls gewünscht wieder gelöscht werden kann.

Anhand von solchen Resourcen kann man alle Anfragen auf das Backend eines Benutzers abarbeiten.

### Benutzer Eingaben Validierung

#### Escaping von HTML

Für die Bereinigung von HTML in Strings wird eine Bibliothek benutzt, nämlich [Bleach](https://github.com/mozilla/bleach) von Mozilla. Mit einer einfachen Methode wandelt Sie mögliche böse Tags in sichere HTML Encodings um.

Beispiel: Ein ‘<’ wird zu ‘&lt;’. Dieses Zeichen wird dann von Browsern nicht als ein Tag angeschaut.

Als Beispiel den Unit Test welche die Bereinigungsfunktion testet.



Abbildung 22 HTML Bereinigungsbeispiel

#### Länge, Not Null Validierungen

Bei einem String muss natürlich auch noch geprüft werden, dass dieser nicht null ist. Darum wird vor dem Erstellen eines Modells die Felder noch auf ‘not None’ und mit ‘len(string)’ auf Länge geprüft.

# Glossar

|  |  |
| --- | --- |
| Begriff | Definition |
| ASGI | Asynchronous Server Gateway Interface, Nachfolger des Web Server Gateway Interface ist ein Web Server welches die Bibliothek asyncio von Python unterstützt. |
| Docstring | Ein Docstring ist ein einzeiliger oder mehrzeiliger Kommentar in Python welcher die Funktion einer Funktion beschreibt. |
| Sphinx | Sphinx ist ein Dokumentation Tool für Python, welches automatisch Docstrings zusammenstellen kann. |
| Mockups | Skizzen des Endresultats |
| Loriot | Loriot ist Webapplikation von welcher die Sensor Daten geholt werden können. |
| Websockets | http Protokoll |
| Uvicorn | ASGI Server |
| Nginx | Statischer Webserver |
| Apache | Webserver |

# Literaturverzeichnis

|  |  |  |
| --- | --- | --- |
| Abgerufener Link | Wann | Begründung |
| <https://www.chartjs.org/docs/latest/charts/line.html> | 23.03.2020 | Hilfestellung für Chart.js Beispiel. |
| <https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types> | 23.03.2020 | Mime Types wurden für verschiedene Diagramme verwendet. |
| <https://www.slant.co/options/10578/alternatives/~chart-js-alternatives> | 23.03.2020 | Hilfe Auswahl von JavaScript Diagramm Bibliotheken. |
| <https://gionkunz.github.io/chartist-js/getting-started.html> | 23.03.2020 | Hilfestellung für Chartist.js |
| <https://channels.readthedocs.io/en/latest/asgi.html> | 23.03.2020 | Hilfestellung für eigene Erklärung |
| <https://de.wikipedia.org/wiki/Web_Server_Gateway_Interface> | 23.03.2020 | Hilfestellung für eigene Erklärung |
| <https://iot-shop.de/wp-content/uploads/2020/03/RM_Door-_-Window-Sensor_20200205_v2.pdf> | 24.03.2020 | Hilfestellung für Dekodierung des Sensor Wertes |
| <https://www.hacksplaining.com/> | 24.03.2020 | Hilfestellung für wichtigste Sicherheitsaspekte der Webapplikation |
| <https://www.postgresql.org/docs/9.2/plpython-data.html> | 24.03.2020 | Datentypen von PostgreSQL |
| <https://stackoverflow.com/questions/9002901/how-to-import-module-to-sphinx> | 25.03.2020 | Hilfestellung für Problem mit Sphinx |
| <https://stackoverflow.com/questions/5464627/how-to-have-same-text-in-two-links-with-restructured-text> | 25.03.2020 | Problembehebung |
| <https://github.com/mozilla/bleach> | 27.03.2020 | Lösung für die Bereinigung von HTML in Strings. |
| <https://stackoverflow.com/questions/1957273/how-do-i-generate-a-random-string-of-length-x-a-z-only-in-python> | 30.03.2020 | Hilfestellung für die Erstellung eines Random Strings einer gewissen Länge. |

# Abbildungsverzeichnis

[Abbildung 1 Chart.js Resultat 15](#_Toc35948450)

[Abbildung 2 Chart.js Code 15](#_Toc35948451)

[Abbildung 3 Chartist.js Resultat 15](#_Toc35948452)

[Abbildung 4 Chartis.js Code 15](#_Toc35948453)

[Abbildung 5 Sensor Wert Ablauf 16](#_Toc35948454)

[Abbildung 6 Chromium Entwickler Modus 17](#_Toc35948455)

[Abbildung 7 Chromium Responsive Modus 17](#_Toc35948456)

[Abbildung 8 Google Style Docstrings 17](#_Toc35948457)

[Abbildung 9 Use Case 19](#_Toc35948458)

[Abbildung 10 System Aufbau 20](#_Toc35948459)

[Abbildung 11 Authentifizierung Ablauf 22](#_Toc35948460)

[Abbildung 12 Admin Dashboard 22](#_Toc35948461)

[Abbildung 13 Personal Seite 23](#_Toc35948462)

[Abbildung 14 Räume Verwaltung 24](#_Toc35948463)

[Abbildung 15 Raum Ansicht 25](#_Toc35948464)

[Abbildung 16 Sensoren Verwaltung 26](#_Toc35948465)

[Abbildung 17 Sensor Ansicht 27](#_Toc35948466)

[Abbildung 18 Dashboard Reinigungspersonal 28](#_Toc35948467)

[Abbildung 19 Konzeptionelles Model 29](#_Toc35948468)

[Abbildung 20 Logisches Modell 30](#_Toc35948469)