



GreenPI Eco

TECHNOLOGY

Gewächshaus auf Basis eines Raspberry Pi

VORDIPLOMARBEIT HFU USTER 2018-2019

MICHAEL PFISTER, SEVERIN PROBST, STEPHAN HAUSER

Vorwort

Unser Team existiert nun seit eineinhalb Jahren. In dieser Zeit haben wir gemeinsam diverse Arbeiten erstellt. Dadurch sind wir ein eingespieltes Team geworden. Gemeinsam suchten wir nach einem geeignetem Projekt für unsere VDA. Michael wollte gerne etwas mit einem Raspberry Pi realisieren. Severin wollte nach dem PHP-Kurs tiefer in die Welt der Programmiersprachen eintauchen. Auf der Suche nach einem Projekt recherchierte Stephan nach Raspberry PI Projekten. Dabei stach ihm ein Projekt einer automatisierten Bewässerung eines Garten ins Auge. Die Idee gefiel ihm, aber das Projekt war für seinen Garten nicht umsetzbar. Er wusste aber, dass ein Freund von ihm ein Gewächshaus hatte, das er nicht mehr brauchte und das sich optimal als Projekt eignen würde. So entstand die Grundidee.

Gemeinsam haben wir die Idee erweitert. Der Plan war es, eine Software zu schreiben, mit der man Sensoren aus dem Internet ansteuern kann. In unserem Projekt sind es die Raspberry PI Sensoren, die später alle noch detailliert vorgestellt werden. Bei der Programmierung achteten wir darauf, dass unser Projekt erweiterbar ist und dass die Software nicht nur der Aufzucht von Pflanzen dient, sondern auch andere Zwecke erfüllen kann.

Durch die Online-Fähigkeit unseres Gewächshauses kann man in den Ferien ein Auge auf seine Pflanzen werfen. Eine Kamera ermöglicht, es ein Gefühl dafür zu entwickeln, wann gegossen werden muss. Die Anwendungen einer solchen Software können jedoch sehr vielseitig sein, so könnte später auch eine Videoüberwachung mit Bewegungssensoren oder eine einfache Home-Automatisation daraus werden. Man könnte sein Aquarium oder Terrarium damit überwachen und steuern oder die Katzen füttern und nachsehen ob der Herd noch an ist.

Ein grosses Dankeschön an:

Wir haben uns eine besondere Aufgabe ausgesucht, die wir nicht ohne die Geduld unserer besseren Hälften hätten schaffen können. Insbesondere geht der Dank an die, die uns halfen die Arbeit nochmals durchzulesen und uns mit der Gestaltung von Logos und Design unterstützten.

Auch ein Dank an die Leute von Bastelgarage.ch die uns bei der Sensorauswahl unterstützt haben und uns Ihre Bilder für die Arbeit zur Verfügung gestellt haben.

1 Inhaltsverzeichnis

Vorwort	1
Ein grosses Dankeschön an:	1
1 Inhaltsverzeichnis	2
2 Abkürzungen / Abkürzungsverzeichnis	7
3 Management Summery	8
4 Einleitung.....	9
4.1 Ausgangslage	9
4.2 Aufgabestellung.....	9
4.2.1 Muss Ziele:.....	10
4.2.2 Kann Ziele:	10
4.3 Einführungen in die Technologie.....	11
4.4 Zielsetzung und Herausforderungen	11
5 Übersicht der Projektplanung	12
6 Vorstudie	13
6.1 Einleitung.....	13
6.2 Projektorganisation	13
6.2.1 Projektstrukturplan	13
6.2.2 Arbeitspakete mit Vorgängen	13
6.2.3 Pflichtenheft	13
6.2.4 Zeitplan.....	13
6.2.5 Netzplan	13
6.3 Hardware Evaluation	13
6.3.1 Raspberry Pi.....	13
6.3.2 Sensoren.....	14
6.3.3 Kamera.....	14
6.3.4 Relais.....	15
6.3.5 Budget	15

6.4	Software Evaluation	15
6.4.1	Django oder Flask	15
6.4.2	Evaluation Programmiersprache	16
6.4.3	MySQL oder SQLite	16
6.4.4	Sensoren auslesen	17
7	Konzept	17
7.1	Use Case	18
7.2	Entwurf der Datenbank	19
7.2.1	Realitätsanalyse	19
7.2.2	Datenrelation	20
7.2.3	Datenbank Setup	20
7.2.4	Abweichungen im DB-Konzept	20
7.3	Testkonzept	21
7.3.1	Einleitung	21
7.3.2	Testbeteiligte	21
7.4	Softwarestruktur	22
7.4.1	UML	22
7.4.2	Aufbau des Webinterface	22
7.4.3	Felder der Formulare definieren	23
8	Screen Layout	23
8.1	Startseite	23
8.2	Eventlogseite	24
8.3	Sensorseite	24
8.4	Relaisseite	24
8.5	Profilseite	25
8.6	Landingpage	25
8.7	Mobile First	26
9	Clientseitige Programmierung	27

9.1	Übersicht der Clientseite	27
9.2	Softwareanforderung	28
9.2.1	Frontend	28
9.2.2	Backend	28
9.3	HTML, CSS	28
9.4	JavaScript, Frameworks	28
9.4.1	JQuery	28
9.4.2	Toastr	28
9.4.3	Chartist.js	28
10	Serverseitige Programmierung	29
10.1	Einführung Serverseite	29
10.2	Django	29
10.2.1	Django Einführung	29
10.2.2	models.py	30
10.2.1	URL	30
10.2.2	views.py	31
10.2.3	Templates	31
10.2.4	Module (Django-APPs)	31
	31
10.3	MySQL Einführung	32
10.4	Programmstruktur Serverseite	33
10.4.1	Einführung in die Programmstruktur	33
10.4.2	Reader (SchedulerAPI)	34
10.4.3	DB Access	34
10.4.4	Info und Info-Provider Pattern	34
10.4.5	MySQL	35
10.4.6	Sensorskripte	35
11	Hardware	35

11.1	Einleitung	35
11.2	Aufbau Gewächshaus	36
11.3	Raspberry Pi (RPI)	36
11.3.1	Schnittstellen	37
11.3.2	Raspberry Pi Software-Installation	38
11.4	DHT22	38
11.4.1	Anschlussschema DHT22	39
11.5	Bodenfeuchtigkeit	39
11.5.1	Analog/Digital Konverter	39
11.5.2	Bodenfeuchtigkeitssensor	40
11.6	Anschlussschema Bodenfeuchtigkeitssensor	40
11.7	Relais	41
11.7.1	Anschlussschema Relais	42
11.8	Kamera	42
11.8.1	Kamera als Livestream einrichten	43
12	Kontrolle	44
12.1	Allgemeinte Feststellungen bei der Schlusskontrolle	44
12.2	Test der Startseite	44
12.3	Test der Projektseite	45
12.4	Test der Sensorseite	45
12.5	Test der Relais-Seite	46
12.6	Test der Seiten geplante Aufgaben und Event-Log	46
13	Abgabe	46
13.1	Installationsanleitung	46
13.1.1	Installation der Plugins	47
14	Projekt Review	47
14.1	Entwicklungsumgebung	47
14.2	Raspberry Pi Installation	47

14.2.1	Scheduler Engine	47
14.3	Steigende Komplexität	47
15	Literaturverzeichnis	48
16	Abbildung und Tabellenverzeichnis.....	49
17	Anhang und Beilagen auf USB-Stick	50
17.1	Anhang.....	50
17.2	USB-Stick.....	50

2 Abkürzungen / Abkürzungsverzeichnis

Abkürzung	Beschreibung
DB	Datenbank
RPI	Raspberry Pi
ERM	Entity-Relationship-Modell
RM	Relationen-Modell
SQLI	SQL-Injection
XSS	Cross-Site-Scripting
REPO	Repository, im Zusammenhang mit GIT
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
PSP	Projektstrukturplan

3 Management Summery

Smart Garden (wie der «edn» von <https://www.edntech.com/>) werden immer beliebter. Das sind kleine automatisierte Gärten, die es einem erlauben Salate, Kräuter oder Gemüse bequem zuhause anzupflanzen. Die RPI Plattform bietet eine breite Auswahl an Sensoren um ein solches Projekt selber umzusetzen.

In der Konzeptphase haben wir beschlossen die Applikation so zu entwerfen, dass nicht nur Pflanzen aufgezogen werden können, sondern auch zur Auswertung von Sensordaten verwendet werden kann. Die Applikation ist über ein Webinterface bedienbar. In unserem Gewächshaus kann die Bewässerung, Belüftung und die Beleuchtung bedient werden. Zudem kann über eine Webcam das System überwacht werden.

Das Ziel unserer Überwachung ist es bei längeren Abwesenheiten seine Pflanzen gut versorgt zu wissen. Für Gärtner, die gerne die Kontrolle über ihr Pflanzenklima haben, bieten diverse Sensoren Live-Daten z.B. über die Temperatur und die Luftfeuchtigkeit. Als Hardwaregrundlage diente uns der RPI. Die populäre Plattform biete eine gute Grundlage für Leute die unser Projekt nachbauen und unsere Software nutzen möchten.

Die Applikation ist in Python geschrieben und als Webserver wird ein Django-Server verwendet, welcher in der gleichen Programmiersprache geschrieben wurde. Das automatisierte Einlesen der Sensordaten in die Datenbank haben wir, ohne weiter Tools einzusetzen, programmiert. Dadurch haben wir die volle Kontrolle über diese Programmlogik. Die Daten werden in einer MariaDB abgelegt. Das Frontend ist mit HTML und CSS erstellt worden. Gewisse Abfragen werden asynchron über einen «AJAX-Call» mit JavaScript ausgeführt.

4 Einleitung

4.1 Ausgangslage

Es besteht bereits ein funktionierendes Gewächshaus. Die Herausforderung besteht darin, das Gewächshaus zu automatisieren. Dafür wird ein RPI eingesetzt und über diesen verschiedene Sensoren angesteuert. Das Ganze soll über eine Web-Oberfläche steuerbar sein.

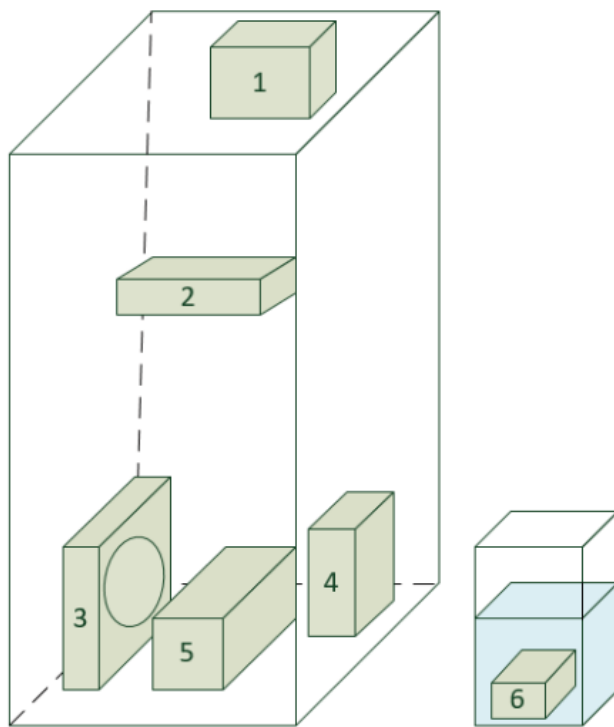


Abb. 1 Gewächshaus

Diese Grafik zeigt die vorhandenen Elemente des Gewächshauses

1. Abluft
2. Lampe
3. Umluft
4. Luftentfeuchter
5. Topf für Pflanzen
6. Tauchpumpe in der Wassertonne

Das Gewächshaus verfügt über eine automatische Bewässerungsanlage, die ihr Wasser über eine Regentonne per Tauchpumpe bezieht. Über eine Lampe bekommen die Pflanzen das notwendige Licht. Ein Umluftventilator sorgt für die notwendige Luftzirkulation. Der Luftentfeuchter kontrolliert das Klima. Für die Abluft ist ein Lüfter im Einsatz.

4.2 Aufgabestellung

Die Aufgabenstellung haben wir in Muss- und Kann-Ziele gegliedert und auch bei der Umsetzung entsprechend priorisiert.

4.2.1 Muss Ziele:



4.2.2 Kann Ziele:

E-Mail-Benachrichtigung

- Bei kritischen Sensorwerten
- Alarm vom Schutzsensor gegen Überschwemmungen

Füllstand Wassertank

Sensoren interagieren

- Wenn der Bodenfeuchtigkeitssensorwert unter den Schwellwert fällt, dann wird die Bewässerung aktiviert
- Wird es zu heiss, soll automatisch die Lüftung aktiviert werden

Bewässerungsanlage aktivieren

- Bodenfeuchtigkeitssensor kann nach Bedarf Bewässerungsanlage aktivieren

Schutzsensor gegen Überschwemmung

- Am Boden des Gewächshauses soll ein Sensor installiert werden, welcher auf Wasserkontakt reagiert

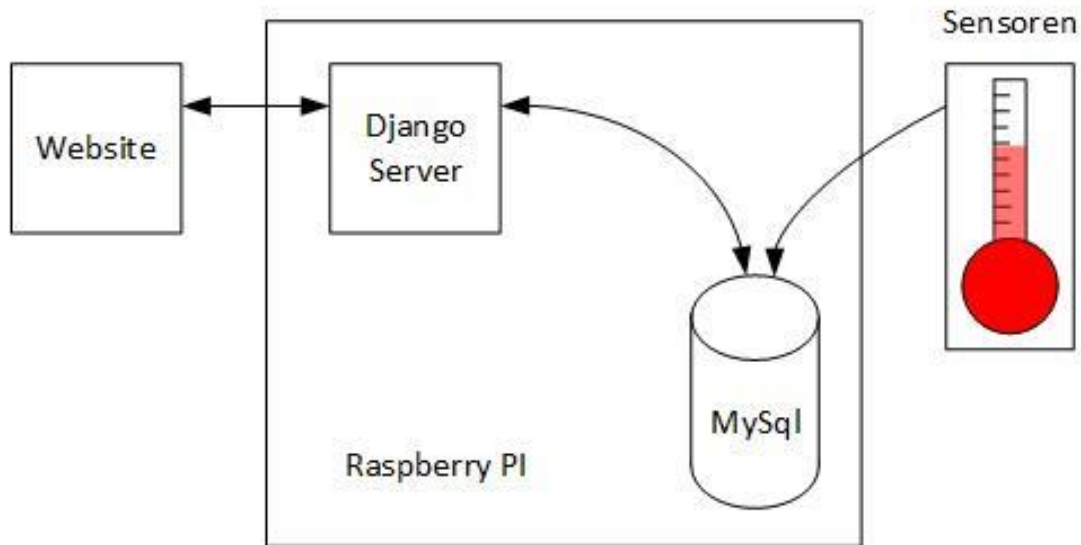
Notabschaltung

- Wenn der Schutzsensor gegen Überschwemmung anschlägt, wird das System heruntergefahren
- Bei unkontrolliertem Temperaturanstieg soll die Notabschaltung ebenfalls erfolgen

Neue Sensoren anlegen

- Es sollen weitere Sensortypen erfasst werden, welche dann bei den Projekten eingesetzt werden können

4.3 Einführungen in die Technologie



Der Benutzer kann sich auf unserer Website einloggen. Danach besteht die Möglichkeit über eine Kamera das ganze System optisch zu überwachen. Er sieht die Daten der Sensoren die graphisch aufbereitet werden. Er hat die Möglichkeit z.B. manuell zu bewässern. Die Website wird mit dem Django Framework umgesetzt, das auf der Programmiersprache Python basiert. Wie bei den meisten Websites verwenden wir auf der Clientseite HTML, CSS, Bootstrap und JavaScript. Serverseitig verwenden wir Python statt PHP. Wir haben uns für Python entschieden, weil die Sensoren einfacher mit Python zu programmieren sind.

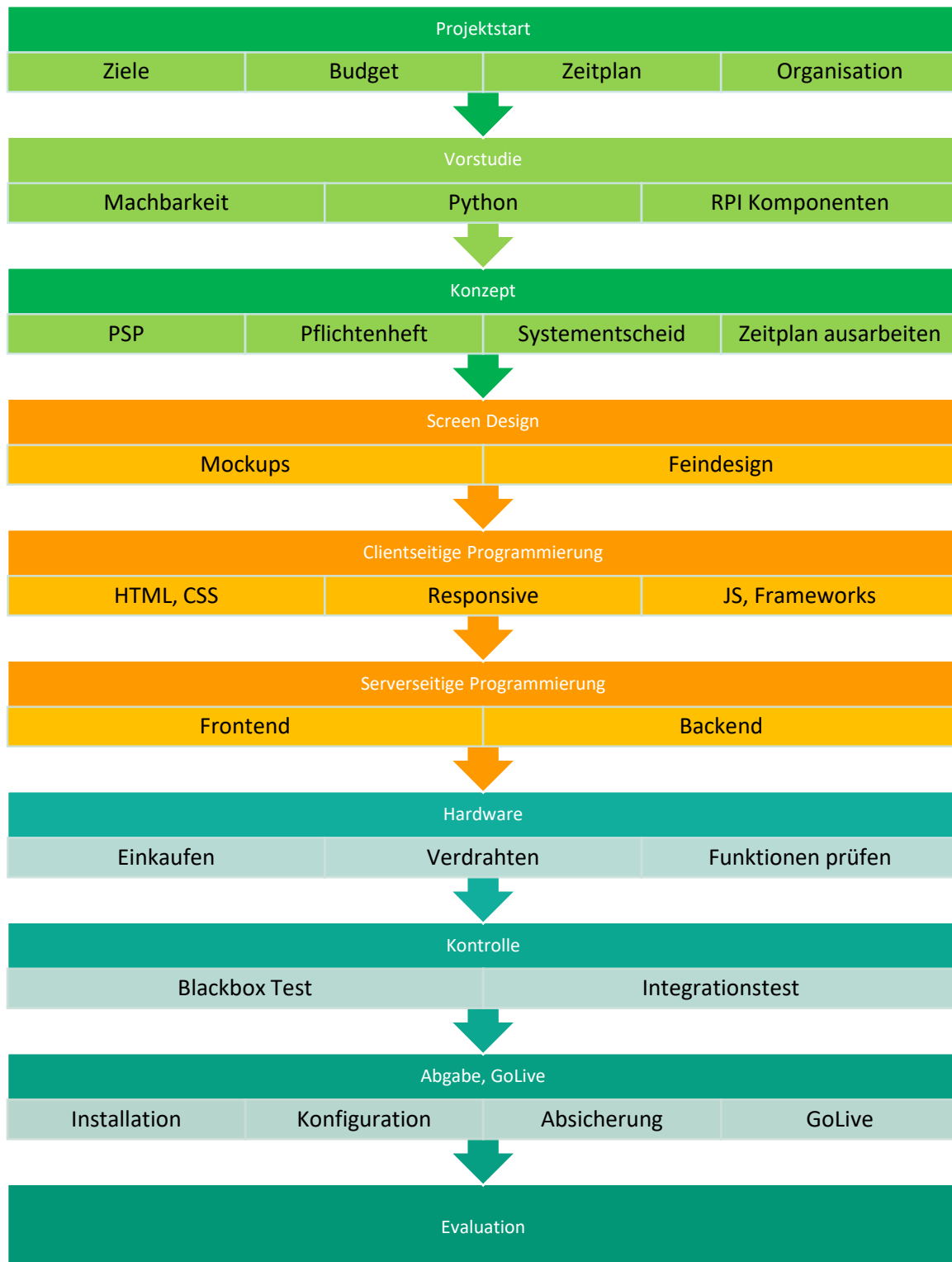
Das System basiert auf einem RPI auf dem unser Django-Server und eine MySQL DB läuft. Die DB fungiert als Speicher für die Daten. Die Sensoren können ihre Werte darin abspeichern und über Django kann man diese dann auf der Website darstellen.

4.4 Zielsetzung und Herausforderungen

Die Zielsetzung der Arbeit hat sich durch unsere spannenden Diskussionen immer weiter verfeinert. Am Anfang stand, wie im Vorwort erwähnt, die Automatisierung eines Gewächshauses im Mittelpunkt. Wir stellten aber schnell fest, dass wenn wir den Schwerpunkt etwas verschieben, eine Software schreiben können, die für mehr als nur für ein Gewächshaus tauglich ist. Das würde bedeuten, dass es möglich sein musste von der clientseitigen Bedienung neue Sensoren hinzuzufügen und zu entfernen, was den Aufbau der Arbeit um einiges komplizierter macht. Es würde z.B. eine viel komplexere DB Struktur notwendig sein.

Auch eine ungelöste Frage ist, wie man die Sensoren in regelmässigen Abständen abfragen sollte. Im Kapitel 6.4.4 "Sensoren auslesen" werden die möglichen Lösungen genauer tematisiert.

5 Übersicht der Projektplanung



6 Vorstudie

6.1 Einleitung

In der Vorstudie musste die Machbarkeit der Idee bewiesen werden. Wegen des engen Zeitplans mussten wir beweisen, dass unser Vorhaben realisierbar war. Schnell wurde uns die Komplexität unseres Vorhabens bewusst und somit auch das Bedürfnis nach Werkzeugen um unser Projekt zu organisieren.

6.2 Projektorganisation

6.2.1 Projektstrukturplan

Der Projektstrukturplan ist in Phasen aufgeteilt. Für dieses Projekt wurde die Form einer Mindmap bevorzugt, da bereits der erste auf Papier entstandene Plan ebenfalls diese Form hatte. Der Projektstrukturplan befindet sich im Anhang.

6.2.2 Arbeitspakete mit Vorgängen

Um die Aufgaben im Team richtig verteilen zu können, wurde eine ausführliche Liste mit Arbeitspaketen und Vorgängen erstellt. Sie zeigt den Umfang der Arbeit. Diese Liste ist ebenfalls im Anhang zu finden.

6.2.3 Pflichtenheft

Das Pflichtenheft befindet sich im Anhang.

6.2.4 Zeitplan

Aus den Arbeitspaketen haben wir mittels MS Project einen Zeitplan erstellt. Dieser befindet sich auf dem USB-Stick. Entweder als PDF oder als MS Project Datei.

6.2.5 Netzplan

Der Netzplan wird im MS Project aus dem Terminplan erstellt. Dieser zeigt auch gleich den kritischen Pfad auf. Beides befindet sich auf dem USB-Stick.

6.3 Hardware Evaluation

Der Teil Hardware Evaluation beschäftigt sich damit, warum wir uns für diese Hardware entschieden haben. Weitere Details zum Thema Hardware findet man unter dem gleichnamigen Kapitel.

6.3.1 Raspberry Pi

Es gibt zwei bekannte Systeme, die für die Arbeit in Frage kommen. Zum einen ist das der RPI zum andern der Arduino. Der Arduino ist aber nur ein Board, das zusätzlich einen Computer braucht. Auf

dem Computer wird der Code geschrieben, der später auf den Arduino geladen wird. Ein Arduino kann also z.B. Code für eine Rollladen-Steuerung ausführen, mehr aber nicht. Der RPI hingegen ist ein eigener kleiner Computer, auf dem Programme entwickelt und parallel ablaufen können. Zudem kann unser Django Server und die MySQL DB direkt darauf laufen.(1)

Die Entscheidung ist auf den RPI 3B gefallen, da es diesen in einem Starterpaket im Angebot gab. Es wäre bereits das neuere Modell 3B+ auf dem Markt, dieses unterscheidet sich aber nur in nicht projektkritischen Details vom 3B Modell.

6.3.2 Sensoren

Um die «Muss Ziele» zu erfüllen braucht es mindestens die folgenden Sensoren:

- Bodenfeuchtigkeitssensor, der in der Lage ist die Feuchtigkeit in der Erde zu messen.
- Temperatursensor
- Luftfeuchtigkeitssensor

Bei den Bodenfeuchtigkeitssensoren gibt es zwei Kategorien. Die einen arbeiten mit dem Widerstand der sich je nach Feuchtigkeit ändert. Die andere arbeiten mit der Veränderung der Kapazität an einem Kondensator, die sich ebenfalls bei Feuchtigkeitsunterschieden ändert. Es sind beides analoge Sensoren und erfordern einen Analog/Digital Konverter (ADC), da der RPI nicht standardmässig mit solchen Anschlüssen geliefert wird. Die kapazitiven Sensoren kosten ungefähr das Doppelte im Vergleich zu denen, die mit Widerständen arbeiten. Sie sind jedoch genauer und gegen Korrosion unempfindlich und darum unsere erste Wahl. (2)

Bei der Temperatur und der Luftfeuchtigkeit gibt es Sensoren, die beides messen und ausgeben können. Die DHT Reihe hat einen günstigeren DHT11 und einen teureren DHT22 Sensor. Beide können die Luftfeuchtigkeit und die Temperatur messen. Der DHT22 ist ungefähr dreimal teurer als der DHT11, hat dafür die wesentlich besseren Kennwerte. (2)

<i>DHT Vergleich</i>	<i>DHT11</i>	<i>DHT22</i>
<i>Luftfeuchtigkeit</i>	20-80% +/-5%	0-100% +/-2-5%
<i>Temperatur</i>	0-50°C +/-2°C	-40-125°C +/-0.5°C
<i>Preis</i>	3.50	11.50

Tab. 1 DHT Vergleich
(2)

Die Wahl ist dank der besseren Werte auf den DHT22 gefallen.

6.3.3 Kamera

Bei der Kamera gibt es die RPI Camera V2 und die RPI Kamera IR-Cut. Die RPI Camera V2 bietet im Vergleich zu der IR-Cut Kamera eine höhere Auflösung und ist kompakter und leichter. Die IR-Cut Kamera bietet den Vorteil der Nachtsichtfunktion. Da wir auch nachts nach dem Rechten sehen möchten, haben wir uns für die IR-Cut Kamera entschieden.

6.3.4 Relais

Um die «Muss Ziele» zu erreichen braucht es mindestens fünf Relais. Relais sind Schalter die grosse Lasten mit einem 5V Impuls schalten können. Die Relais werden gebraucht um folgendes zu steuern: (2)

- Pflanzenlampe (An/Aus)
- Bewässerungsanlage (An/Aus)
- Abluft (An/Aus)
- Umluftventilator (An/Aus)
- Luftentfeuchter (An/Aus)

Da die Möglichkeit besteht, dass auch noch «Kann Ziele» umgesetzt werden, wird ein 8-Kanal Modul für die Umsetzung erworben.

6.3.5 Budget

<i>Budget</i>	<i>Anzahl</i>	<i>Stückpreis</i>	<i>Total</i>
<i>Raspberry Pi 3B</i>	3	54.90	164.70
<i>Kamera</i>	2	29.50	59.00
<i>DHT22</i>	3	13.50	40.50
<i>Bodenfeuchtigkeitssensor</i>	2	11.50	23.00
<i>16Bit 4-Kanal ADC</i>	1	9.90	9.90
<i>Diverse Kleinteile</i>	1	20.00	20.00
<i>Total</i>			317.10

Tab. 2 Budget
(2)

6.4 Software Evaluation

Die Evaluation von Software und Programmiersprache stellte sich teilweise als knifflige Aufgabe heraus, da wir uns in den meisten Fällen tiefer einlesen mussten.

6.4.1 Django oder Flask

Hier haben wir eine pragmatische Entscheidung getroffen. Da die Webseite vom Flask eher veraltet wirkte, war das Django-Framework schnell unser Favorit. Es bot auf den ersten Blick alle notwendigen Funktionen.

6.4.2 Evaluation Programmiersprache

Unser Programmierer hatte hauptsächlich Erfahrung im Erstellen von Webapplikationen mit C# .Net. Daneben noch ein Basiswissen von Java und C++. Python war für uns alle zwar ein Begriff, aber wir kannten diese Sprache kaum.

Java

Diese Programmiersprache fiel schnell weg, da diese in einer virtuellen Umgebung läuft und dementsprechend weiter von der Hardware entfernt ist.

C# .Net

Das ursprüngliche .Net Framework ist auf dem Linux nicht ohne weiteres lauffähig. Ein Docker-Container hätte es gleichwohl ermöglicht. Eine weitere Option wäre der Umstieg auf .Net Core gewesen. Beides schien uns gewagt.

C++

Der Vorteil gegenüber Java ist, dass die Sprache hardwarenahe ist und auch auf dem RPI im Einsatz ist bzw. Teile der Sensoren-Bibliotheken sind in C oder C++ geschrieben. Jedoch war das für uns zu ambitioniert. Es wird in der Regel davon abgeraten als Anfänge mit dieser Programmiersprache zu starten

Python

Die Sprache hatte etwas anziehendes. Jeder von uns wollte Python lernen. Wir hatten Bedenken, da die Sprache grosse Abweichungen zu den anderen Programmiersprachen hatte.

Der Entscheid, Python zu verwenden, wurde durch den breiten Community-Support gestützt. Die Skripte in den Starterbaukits waren in Python geschrieben. Somit haben wir uns für Python entschieden.

6.4.3 MySQL oder SQLite

Der Django-Server verwendet als Standard die SQLite DB, welches ein File-System basiertes Datenbanksystem ist. Dieses Kriterium, so wie die schwache Typisierung von SQLite, hat uns dazu bewogen nach weiteren DB-Systemen zu suchen. Im PHP haben wir MySQL kennen gelernt. Wir empfanden auch das Erstellen und Verwalten massiv einfacher. Deshalb haben wir MySQL ausgewählt

6.4.4 Sensoren auslesen

Die Sensoren sollten in regelmässigen Abständen ausgelesen werden können. Wir haben dazu drei Varianten geprüft.

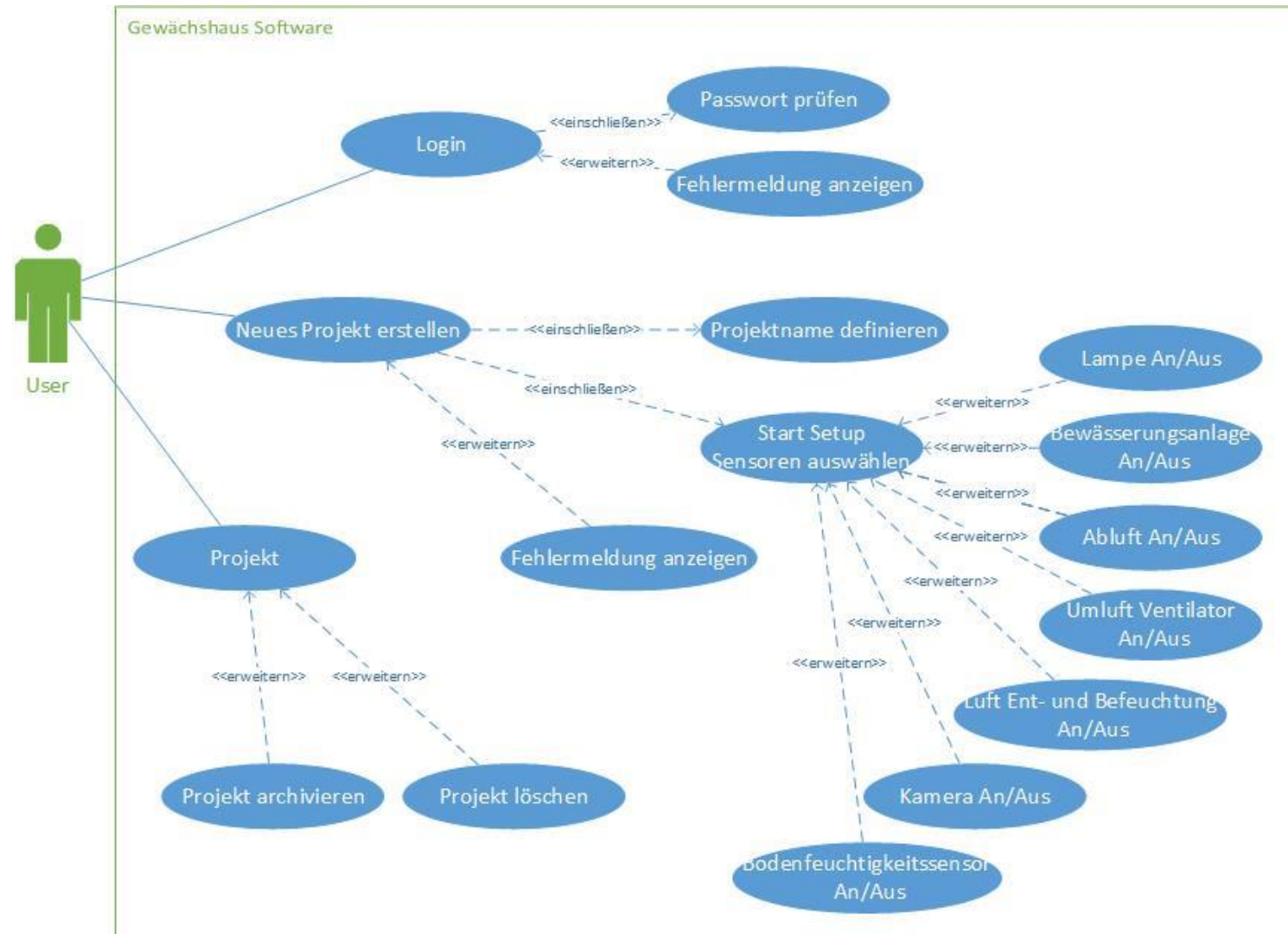
Variante 1 (Chronjob)	Variante 2 (Django Celery)	Variante 3 (Eigenes Skript)
Nicht sehr komplex, als Taktgeber sicher gut geeignet.	Sehr komplex, auch erfahrene Programmierer gaben an, dass sie sich mehrere Tage einarbeiten mussten	Ebenfalls etwas komplexer als ein Chronjob. Aber bei weitem einfacher als Celery
Funktion von Linux	Django-Funktion	Ein eigenstehendes Python-Skript, welches Zugriff auf die Django Module hat.

Unter Berücksichtigung des Zeitdrucks entschieden wir uns für ein eigenes Skript. So hatten wir volle Kontrolle über den Aufbau und wurden nicht abhängig von einer weiteren Komponente.

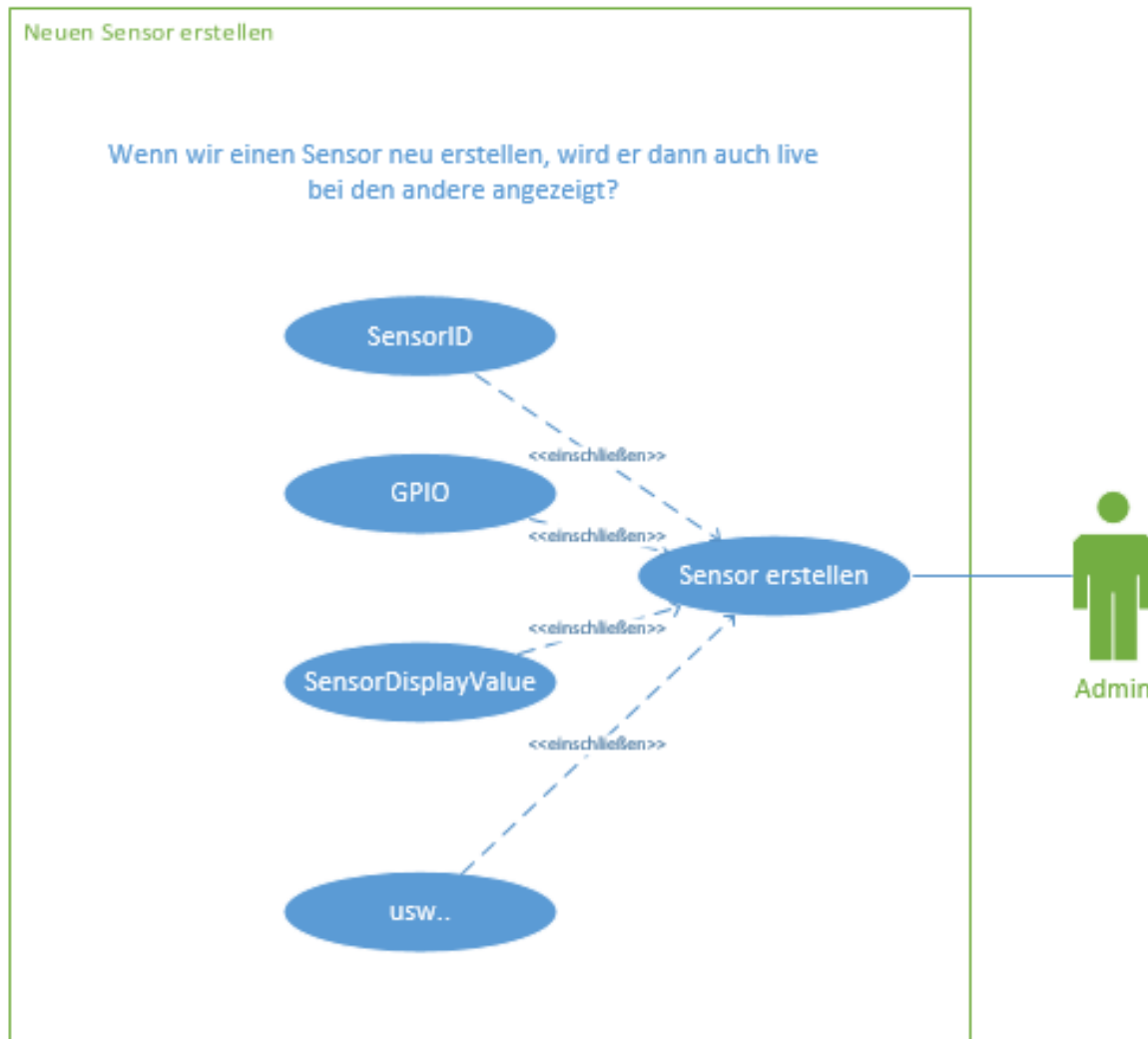
7 Konzept

In der Konzeptphase wurden viele Aufgaben in eine grobe Richtung geleitet. Vieles diente als Diskussionsgrundlage oder der gruppeninternen Kommunikation.

7.1 Use Case



Einschliessen = Ist eine Aktion die so oder so ausgeführt wird.
Erweitern = Ist eine Aktion die nur in einem speziellen Anwendungsfall zum tragen kommt.



7.2 Entwurf der Datenbank

Bei der Entwicklung der DB haben wir mit einer Realitätsanalyse gestartet. Aus dieser wurde ein ERM sowie ein RM erstellt. Diese zwei Dokumente befinden sich auf dem USB-Stick. Im Laufe des Projekts haben sich neue Anforderungen an die DB ergeben. Diese werden in einem Unterkapitel dieses Abschnitts thematisiert. Durch das Entity Framework, welches der Django-Server bietet, konnten wir diese zügig implementieren.

7.2.1 Realitätsanalyse

- Es können Projekte erstellt werden, welche mit dem Setzen der Pflanze beginnen und mit der Ernte enden.
- Projekte haben eine eindeutige ID und werden mit einem Codenamen, Anzeigenamen, Start- und Enddatum erfasst und ob sie aktiviert sind.
- Es kann nur ein Projekt auf einmal aktiv sein, dies wird durch die Applikation sichergestellt.
- Beim Erstellen des Projekts wird die ID des Benutzers im Projekt hinterlegt.
- Benutzer werden mit folgenden Angaben erfasst: Benutzername, Vorname, Nachname, Mail, Passwort, aktiviert und Privileg-Level.
- Einem Projekt können mehrere Sensoren zugewiesen werden.

- Sensoren werden mit folgenden Angaben erfasst: Sensorname, Sensor-Displayname, GPIO und Beschreibung.
- Es können Relais gleich wie die Sensoren erfasst werden.
- Die aufgezeichneten Sensorwerte sind einem Projekt und Sensor zugewiesen und verfügen über Datum und Zeit. Die Werte sind entweder dezimale oder natürliche Zahlen.
- Die Applikation verfügt über einen Event-Log mit folgenden Angaben:
Typ(E;I;W), Eventzeit, Quelle, Beschreibung, URL, Maschinename, UserAgent, UriReferrer, BenutzerID.
- Es können geplante Aufgaben erfasst werden, welche eine eindeutige ID sowie einen Codenamen, Anzeigenamen, Intervall, Dateipfad, letzte Ausführung, Datum und Zeit, nächste Ausführung, letztes Resultat des Tasks und Anzahl Ausführungen des Tasks haben. Zudem können die folgenden Flags angegeben werden:
Der Task wird nach dem letzten Ausführen gelöscht, der Task ist aktiviert und der Task wird gerade ausgeführt.
- Die Sensorwerte sollen periodisch in die DB geschrieben werden.
Diese Intervalle müssen in der DB gespeichert werden, damit sie übers Web steuerbar sind.

7.2.2 Datenrelation

Uns war es wichtig, dass wir keine verwaisten Datensätze haben. In den Django-Models lässt sich das bequem konfigurieren. Man kann bestimmen, was mit den verbundenen Datensätzen geschieht, wenn ein Datensatz gelöscht wird. Beispielsweise werden Sensorwerte aufgezeichnet, welche in einer eigenen DB geloggt werden. Wird nun ein Sensor gelöscht, so werden automatisch auch seine Daten gelöscht.

7.2.3 Datenbank Setup

Da wir unsere DB regelmässig neu erstellen mussten und wir alle die gleichen Datensätze als Grundlage benötigten, haben wir uns dazu entschieden, ein DB Setupskript zu erstellen, in welchem wir die Benutzer, Sensoren, Relais und geplanten Aufgaben automatisch erzeugten. Später ist dieses File angedacht, um das Grundsetup der DB für den Benutzer zu übernehmen. So kann der Benutzer dann bestimmen, mit welchen Daten er starten möchte.

7.2.4 Abweichungen im DB-Konzept

Model	Attribut	Begründung
RelaisInfo	RelaisState (Boolean)	Während der Implementation der Relaissteuerung. hatten wir uns dazu entschlossen, dieses Attribut hinzuzufügen um das Rendering der einzelnen Relais zu vereinfachen. So konnten wir schon in der View den Zustand des Relais hinzufügen.
ScheduledTaskInfo	ObjectType(String) und ObjectID(Int)	Im ObjectType wird definiert, was für eine ID im ObjectID Feld ist. Es kann ein Relais oder Sensor ausgewählt werden. Als Zusatzoption kann auch ein File ausgewählt werden. In dem Fall würde der FilePath und nicht mehr die ObjectID berücksichtigt werden.

SensorInfo	ShowOnHome	Auf der Startseite werden aktuelle Sensorwerte angezeigt. Damit später der Benutzer bestimmen kann, welche Sensoren dort angezeigt werden, haben wir dieses Feld hinzugefügt.
SensorInfo	SensorUnit	Das Feld haben wir hinzugefügt, da es praktisch ist, wenn direkt beim Sensor angegeben werden kann, in welcher Masseinheit er die Werte aufzeichnet.
SensorInfo	SensorCustomData	Über das Feld können diverse Werte mitgegeben werden. Auf der Startseite werden z.B. die aktuellen Sensorwerte hineingeschrieben.
SensorInfo	SensorType	Dieses Feld wird verwendet, um zu entscheiden mit welchem Sensorskript und Funktion der Sensorwert ausgelesen werden kann. Der DHT22 ist ein Spezialfall, dort wird ein physischer Sensor als zwei virtuelle Sensoren geführt.
Allgemein	*CodeName	In den meisten DB Models hat es einen CodeName Attribute z. B. SensorCodeName. Die Namensgebung haben wir angepasst, da häufig der Unterschied zwischen dem SensorName und SensorDisplayName nicht ganz klar war.

7.3 Testkonzept

Im Testkonzept werden die technischen und organisatorischen Rahmenbedingungen, unter denen die Tests abzuwickeln sind, identifiziert und geklärt. Es ist zu klären, welche Testziele verfolgt werden bzw. welche Fragestellungen oder Probleme mit den Tests untersucht werden sollen. Die Ergebnisse dieser Überlegungen werden im Testkonzept dokumentiert.

7.3.1 Einleitung

In unserem Projekt sollen alle Funktionen gemäss Pflichtenheft und unseren Vorstellungen gewährleistet sein. Darum haben wir uns dazu entschieden einen so genannten BlackBox-Test zu machen, da dieser funktionsorientiert ist. Zudem können optional noch externe Helfer mit einbezogen werden.

7.3.2 Testbeteiligte

An den Tests werden das Projektteam sowie auch gegebenenfalls Helfer beteiligt sein. Während das Projektteam das nötige Knowhow mitbringt, bieten die optionalen Helfer eine andere Sicht auf das Projekt.

Das Projektteam ist während des Projekts verantwortlich für die Komponententests. Zudem sollen die Verantwortlichen des Projektteams eigenständig ein Review machen um zu prüfen, ob alle Anforderungen erfüllt werden, das Layout eingehalten wird usw. Zudem ist das Projektteam auch für die Integrationstests verantwortlich. Dazu wird nach den Komponententests das Zusammenspiel der

einzelnen Komponenten geprüft. Anschliessend wird ein Systemtest durchgeführt um die Gesamtheit des Projekts zu prüfen. Ist dies auch erfolgreich, wird noch ein Abnahmetest anhand des Pflichtenhefts durchgeführt.

Die optionalen Helfer sollen das «fertige Produkt» prüfen um Fehler zu erkennen.

7.4 Softwarestruktur

Beim Aufbau der Software haben wir darauf geachtet, zusammengehörende Elemente in Django-APPs zu kapseln. Es war auch wichtig, dass nur ein APP Abhängigkeiten zum RPI hat, da dieses nur dort lauffähig ist. Für die Demo auf dem PC oder zum Testen haben wir ein Dummy-APP entwickelt, welches auch auf dem PC läuft.

7.4.1 UML

Wir haben ein UML Diagramm erstellt als interne Diskussionsgrundlage für den Aufbau der Applikation. Dort konnte auch gut aufgezeigt werden, in welcher Django-APP eine entsprechende Klasse implementiert wird. Während des Projekts, haben wir uns entschlossen auf die Django-Models umzusteigen. Dadurch haben sich diverse Änderungen im Aufbau der Models ergeben, welche wir aus zeitlichen Gründen nicht mehr nachgeführt haben.

7.4.2 Aufbau des Webinterface

Wir bauten das Webinterface Mobile-First auf. Da die Webseite nicht öffentlich zugänglich ist, haben wir auf eine Onpage-Optimierung verzichtet.

Mobile First VS Desktop-Frist

In der Desktop-Frist Variante wird mit der maximalen Bildschirmbreite in Pixel gearbeitet. Beziehungsweise Sobald der Bildschirm kleiner als der angegebene Wert ist wird der Inhalt der Media-Query berücksichtigt. Eine Webseite, welche auf einen Desktop optimiert ist, so umzubiegen ist schwer. Schlussendlich ist es massiv einfacher zuerst die Mobileversion zu gestalten und von dieser Version über die Ipad Version zur Desktop Version vorzuarbeiten.

```
/*>>> Mobile Frist <<<*/  
  
#RelayControl {}  
  
@media (min-width: 576px) {}  
  
@media (min-width: 768px) {}  
  
@media (min-width: 992px) {}  
  
@media (min-width: 1200px) {}
```

Auf der linken Seite siehst du Media-Queries, welche beim Mobile-Frist Ansatz verwendet werden.

Dabei sind die Breakpoint exakt die gleichen wie bei Bootstrap. So können wir bei jeder Veränderung Einflussnahmen.

7.4.3 Felder der Formulare definieren

In der Planung gingen wir davon aus, dass wir unseren eigenen DB Models verwenden. Dadurch war es notwendig das einzelne Formular genau zu definieren. Als Grundlage haben wir UML verwendet, da die Felder Attribute einer Klasse sind. Als wir auf die Django-Models umstiegen, wurden die Formularfelder bereits in den einzelnen Models definiert.

8 Screen Layout

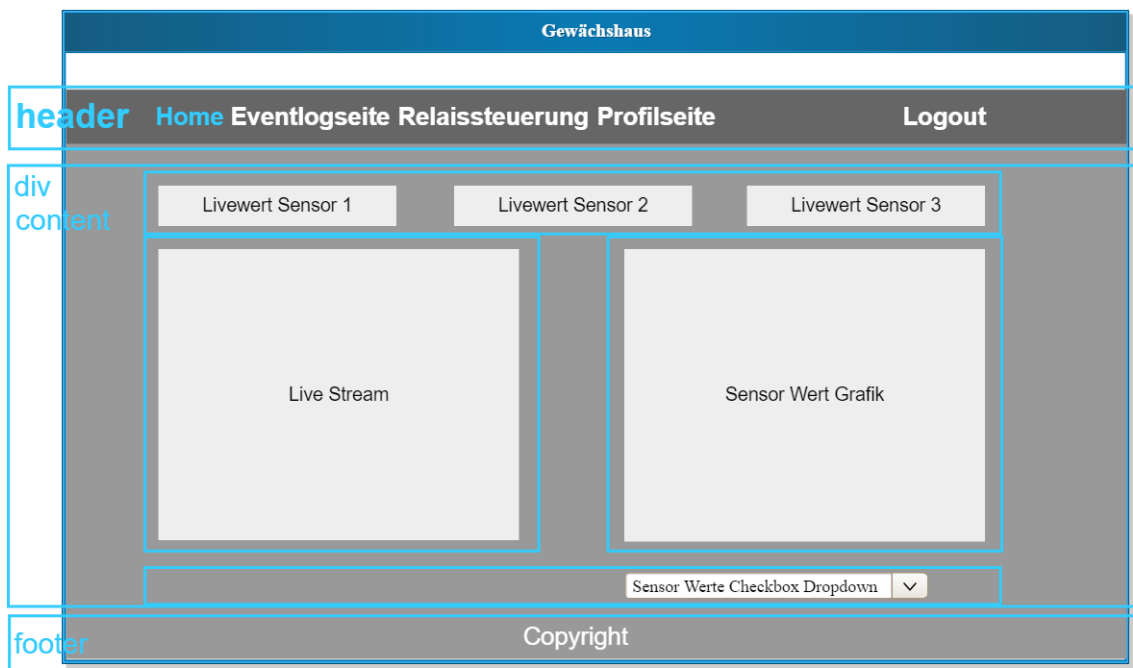
Die Layouts entsprechen unseren Wunschvorstellungen und müssen nicht zwingend genauso umgesetzt werden.

Der Header (Menu) und Footer (Fusszeile) sind auf allen Seiten gleich. Die Seite, auf der man sich gerade befindet wird im Menu farblich markiert.

8.1 Startseite

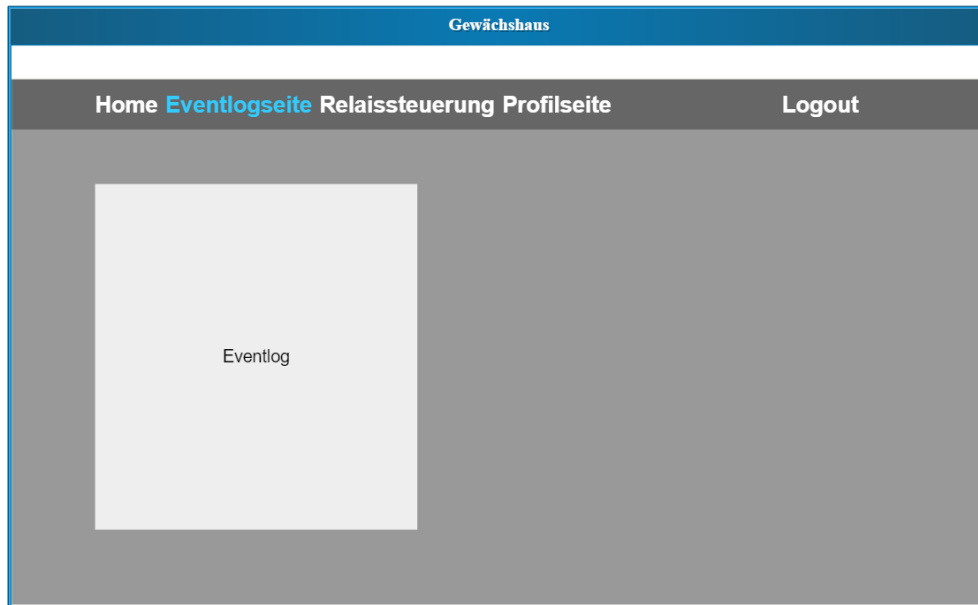
Da einige unseres Teams keine sehr guten HTML Kenntnisse hatten, haben wir auf der Startseite noch einige HTML Tags mit eingezeichnet um HTML besser zu verstehen und die Programmierung zu vereinfachen.

Auf der Startseite sollen drei aktuelle Sensorwerte angezeigt werden. Zudem soll ein Livestream zu sehen sein und eine Grafik. Bei der Grafik sollen die angezeigten Werte ausgewählt werden können.



8.2 Eventlogseite

Auf der Eventlogseite werden Ereignisse geloggt, die stattgefunden haben. Es können Informationen, Warnungen sowie Fehler aufgezeichnet werden.

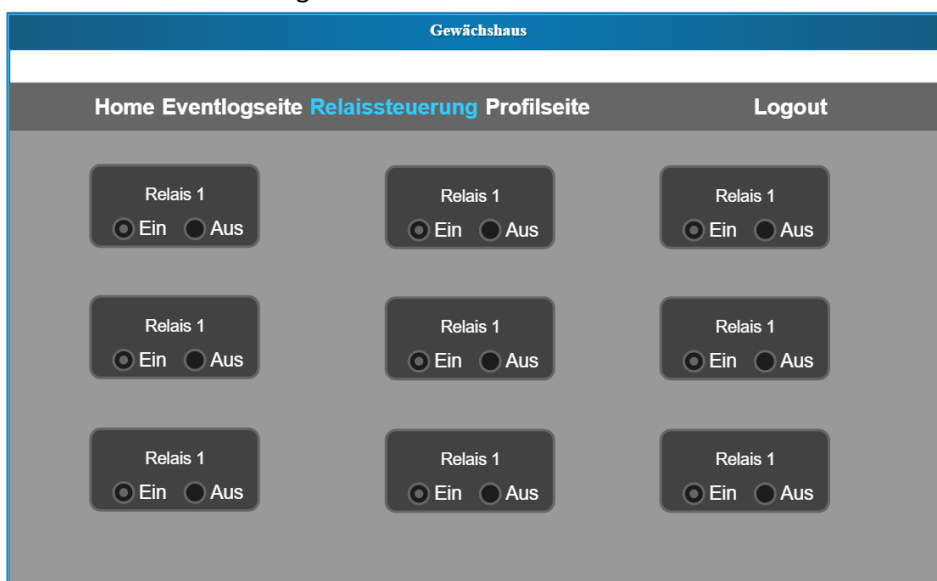


8.3 Sensorseite

Die Sensorseite kam erst im Verlauf des Projekts dazu und entsprechend gibt es kein Layout. Die Sensorseite soll die aktuell dem Projekt zugeordneten Sensoren tabellarisch anzeigen. Zudem soll sie alle verfügbaren Sensoren anzeigen und die Möglichkeit bieten neue Sensoren hinzuzufügen.


8.4 Relaisseite

Auf der Relaisseite sollen die am Projekt angeschlossenen Relais im aktuellen Zustand angezeigt werden. Zudem sollen die Relais steuerbar sein. Am Ende wird eine Schaltfläche angezeigt, mit der man neue Relais hinzufügen kann.



8.5 Profilseite

Auf der Profilseite soll man sein eigenes Profil bearbeiten und neue User registrieren können. Dies erfolgt mit den bekannten Kriterien und zusätzlich noch der Rechtevergabe.



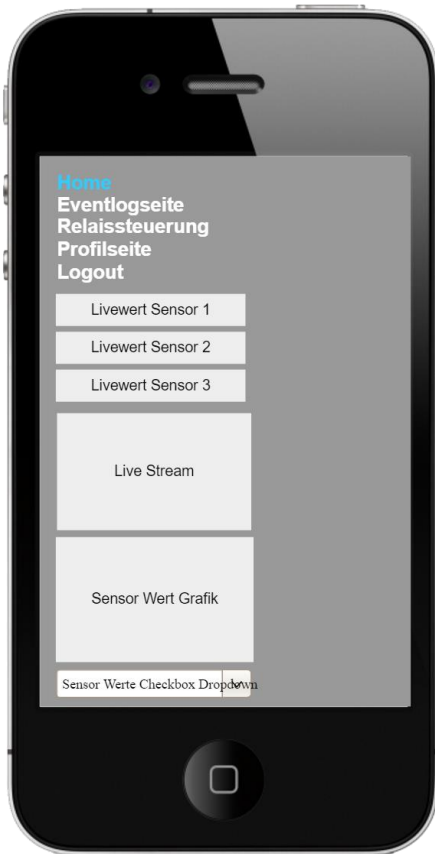
8.6 Landingpage

Da wir eine Kamera mit Livestream implementieren und das Ganze von aussen erreichbar sein soll, haben wir uns dazu entschieden, es mit einem Login zu schützen. Dies soll gleich als erstes, wenn man auf die Webseite zugreifen will, erscheinen.



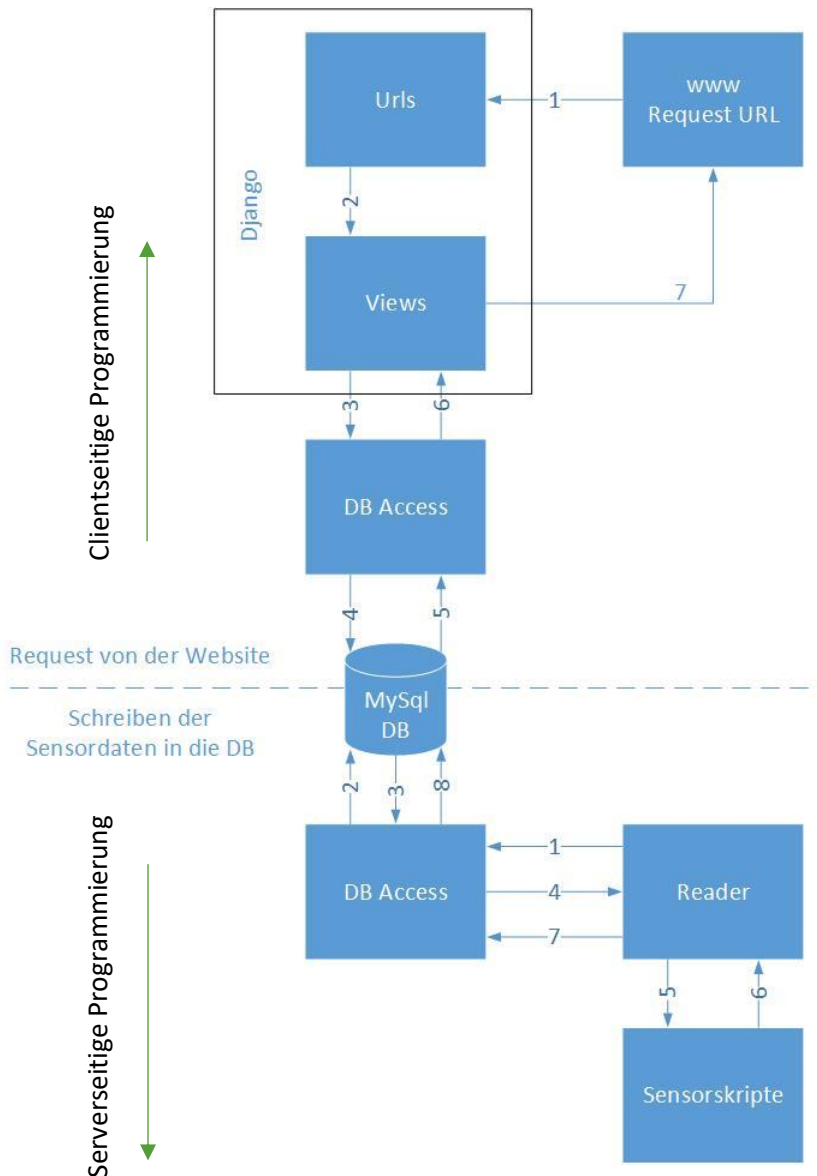
8.7 Mobile First

Wir wollen die Webseite nach Mobile-First aufbauen, entsprechend gibt es auch ein Layout dazu. Es beinhaltet die Startseite auf einem Mobile Gerät.



9 Clientseitige Programmierung

9.1 Übersicht der Clientseite



Client Seite

1. Im Browser wird z.B. der URL von der Startseite eingegeben. Es wird ein Request an den Server gesendet.
2. Über die entsprechenden URL-Pattern wird auf eine entsprechende View verwiesen.
3. In den Views wird der Request verarbeitet und auf die Logik der Website zugegriffen, welche in separate Klassen verkapselt ist.
4. Über das DB-Access APP wird eine Verbindung zur MySQL DB aufgebaut.
5. Die aufgerufene Klasse im DB-Access liest nun die angefragten Daten aus der DB.
6. Die aufgerufene Klasse gibt die angefragten Daten an die Views zurück.
7. Die Rohdaten und das HTML-Template werden nun zusammengefügt und an den Browser zurückgesendet.

9.2 Softwareanforderung

Die Anforderungen haben wir in Client- und Serverseitig aufgeteilt.

9.2.1 Frontend

Das HTML / CSS muss zwingend W3C-Konform sein. Dokument-Semantik und Onsite-SEO können vernachlässigt werden, da das Tool nicht öffentlich zugänglich ist.

9.2.2 Backend

Das Backend soll Modularisiert sein. Der Datenbankzugriff soll immer über das APP DB-Access erfolgen und nie direkt aus einem anderen Modul. Im Optimalfall wird der entsprechende Info-Provider verwendet. Nur in einem APP dürfen Codes existieren, welche nur auf dem RPI lauffähig sind.

9.3 HTML, CSS

Wir haben als Grundlage unserer Webseite das Grid-System des CSS-Framework Bootstrap im Einsatz. Um die Semantik der Seite zu optimieren haben wir uns an den HTML5 Standard gehalten und mit CSS3 gewisse Animationen implementiert.

9.4 JavaScript, Frameworks

Die meisten modernen Webseiten verwenden JavaScript. Auch bei unserem Tool ist teilweise diese Skriptsprache im Einsatz.

9.4.1 JQuery

Dieses weitverbreitete Framework haben wir gewählt, da es sehr stabil ist und etliche Funktionen bietet. Unter anderem verwenden wir den AJAX-Call, um asynchrone Anfragen an den Server zu senden. Bei den Relais wollen wir nicht jedes Mal einen Postback verursachen, wenn wir ein Relais umschalten. So haben wir dort diese Methode verwendet.

9.4.2 Toastr

Toastr ist ein JQuery Plugin, welches wir dazu verwenden haben, um schönere Popup-Nachrichten anzuzeigen als über eine «alert» Funktion. Es unterscheidet zwischen Information, Warnung und Error. Diese drei Meldungsarten können nach Belieben formatiert und eingefärbt werden.

9.4.3 Chartist.js

Mit dieser JavaScript Library lassen sich Diagramme zeichnen.

10 Serverseitige Programmierung

10.1 Einführung Serverseite

In diesem Kapitel gibt es eine Einführung in Django und MySQL, welche die Basis des Projekts darstellen. Im weiteren Verlauf des Kapitels wird die Struktur des Backend im Detail erläutert.

10.2 Django

10.2.1 Django Einführung

Django ist ein Webframework, das auf einem Model-View-Presenter-Schema basiert. Ursprünglich wurde es für die News-Seite «Lawrence Journal-World» im Jahr 2005 entwickelt und nach dem Jazz-Gitarristen Django Reinhardt benannt. (2)

Ein Nutzer kann in seinem Browser eine Anfrage an den Webserver starten. Das Django Framework, das auf dem Server installiert wurde, verarbeitet diese Anfrage und sendet die angefragte Website an den Client-PC zurück.

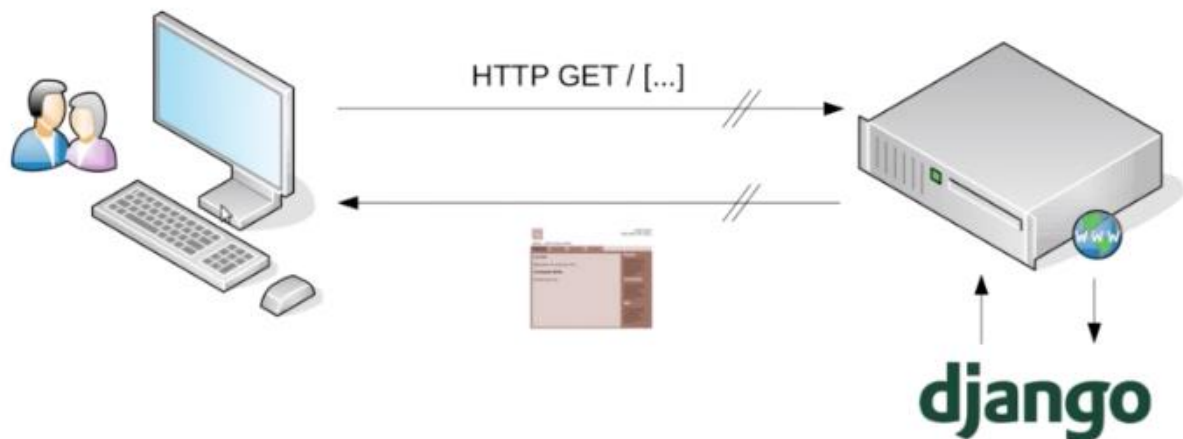


Abb. 2 Django Server
(3)

Django funktioniert im Model-View-Presenter-Schema kurz MVP. D.h. Django hat fünf Kernaufgaben:

1. Der Benutzer gibt die URL der Seite ein, welche er aufrufen möchte.
2. Der Django-Server prüft, ob ein URL-Pattern gleich ist und leitet den Request an die entsprechende View weiter.
3. In der View wird entschieden, welche Elemente hartcodiert sind und welche dynamisch aus der DB geladen werden.
4. In der «Render» Funktion der View wird dann das entsprechende HTML-Template geladen und mit den Daten zusammengeführt.
5. Abschliessen wird die HTML-Page als Response an den Benutzer zurückgesendet.

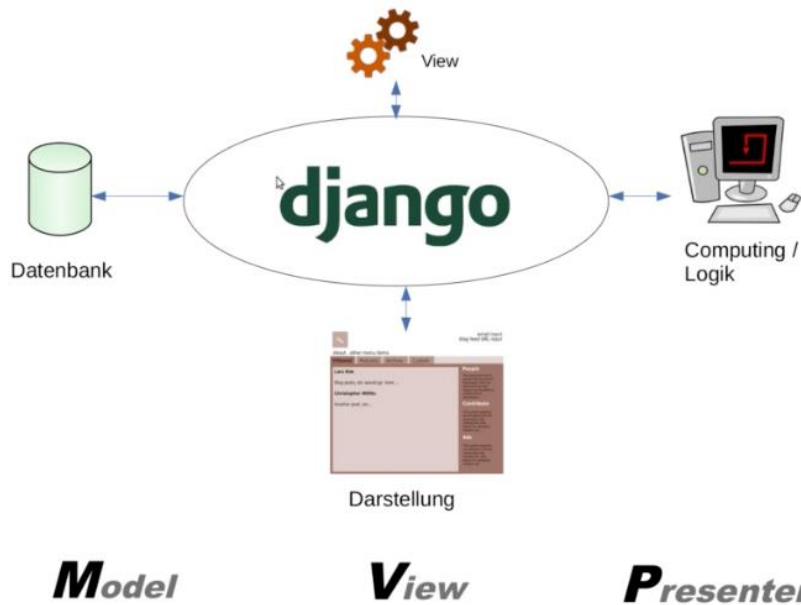


Abb. 3 Django MVP
(3)

Auf der offiziellen Django Website (www.djangoproject.com) steht, dass Django mehr ein MTV Framework ist. MTV steht in diesem Fall für Model-Template-View. Im Model werden die Datenbanken modelliert, unter Template wird das Aussehen definiert und unter View wird definiert welche Daten angezeigt werden sollen.

10.2.2 models.py

In Django wird die Datenbankstruktur in der Datei model.py modelliert. Diese angelegten Strukturen können dann migriert werden. Das bedeutet, dass sie in die DB übernommen werden und die entsprechenden Tabellen erstellt werden. Macht man Änderungen in der Datei model.py, wird lediglich ein erneutes Migrieren notwendig und alle Anpassungen werden automatisch in die MySQL DB übernommen. Bei diesem Vorgang gehen bestehende Datensätze nicht verloren. Durch die Implementation von Model lassen sich Abfragen an die DB objektorientiert erstellen. Das heisst, man muss nicht ein Delete-, Insert- oder Update-Statement schreiben. Bei komplexeren Abfragen ist es hingegen immer noch möglich, ein eigenes SQL-Statement in Kombination mit dem Model zu erstellen. Dies erleichtert nicht nur die Arbeit des Programmierens, es verhindert auch sogenannte SQL-Injections¹.

In unserem Fall haben wir die Model aus der einen Datei ausgelagert, da es mit der Zeit unübersichtlich wurde und nur einer an den Models arbeiten konnte. Unsere Models befinden sich im Verzeichnis «_models» und werden ins models.py importiert.

10.2.1 URL

«Durch diese hohle Gasse muss er kommen», sprich jeder Request des Benutzers wird über diese Datei geleitet. Dort wird anhand der URL entschieden, welche View aufgerufen wird.

¹ SQL Injections sind Hackerangriffe auf Datenbanken, indem direkt SQL Codes in ein Eingabefeld geschrieben werden.

10.2.2 views.py

In views.py befindet sich bei kleineren Projekten die einzelnen Funktionen, welche den Request des Benutzers handeln. In unserem Fall haben wir einen Ordner «views». Es ist notwendig im «__init__.py» jede View zu importieren, damit das url.py File dies erkennt.

In der Methode im entsprechenden View-File wird eine Request-Variable als Parameter mitgegeben. Als erstes wird immer geprüft, ob es sich effektiv um ein HTTP-Request handelt. In der Regel haben wir die Request-Art POST und GET unterschieden.

GET-Request

In der Regel wird in dem Fall die Seite zum ersten Mal aufgerufen. Bei den Sensoren haben wir ebenfalls ein GET-Request verwendet um die SensorID über den Query-String mitzugeben zum Löschen des entsprechenden Sensors.

POST-Request

Dieser Request geschieht immer dann, wenn ein Formular versendet wird. In dem Fall werden die Formulardaten validiert und in die DB abgespeichert.

Render-Funktion

Diese Funktion erstellt den Rückgabewert der View-Methode aus einem HTML-Template und den geladenen Daten.

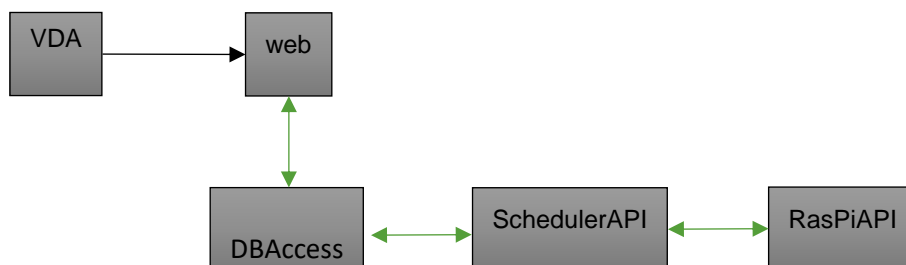
10.2.3 Templates

In den Template Ordner werden die HTML-Dateien abgelegt. Die restlichen Files wie Layout-Bilder, CSS, oder JS werden im Ordner «static» abgelegt und bei Bedarf in der HTML-Datei verlinkt.

10.2.4 Module (Django-APPs)

Nach der Grundinstallation von Django benötigt das Programm mindestens ein App. In der Grundinstallation befinden sich die globalen Einstellungen. Die Software kann nun in verschiedene Module (Apps) aufgeteilt werden. Dies bietet den Vorteil, dass ein Programmierer seine Apps in verschiedenen Produkten verwenden kann.

Für das Projekt wurde die Software in folgende Module aufgeteilt:



VDA

In diesem Modul befinden sich die Settings des Django-Frameworks. Seine Aufgabe ist es, die Anfragen (Requests) des Benutzers über die Weboberfläche abzufangen und an die entsprechende Django-View (Logic) weiterzuleiten.

web

Das web-Modul umfasst die notwendigen Ressourcen für das Darstellen der Webseite. Der Zugriff auf die DB erfolgt immer über das Modul DBAccess und nie direkt aus diesem Modul.

DBAccess

In diesem Modul befinden sich die Django-Model, welche den Aufbau der DB bestimmen. Des weiteren hat es pro Model einen Infoprovder, welcher den direkten Zugriff auf das entsprechende Model kapselt. So ist sichergestellt, dass der Aufruf nach aussen immer gleich ist. Ausnahme sind Delete-, Insert- und Update-Funktionen.

SchedulerAPI

Über dieses Modul wird das Auslesen der Sensoren gesteuert.

RasPiAPI

Dieses Modul darf Code beinhalten, welcher nur auf dem RPI lauffähig ist. Für die Testumgebung wird ein Dummy-Modul implementiert, welches Zufallswerte anstelle der Sensorwerte zurückliefert.



10.3 MySQL Einführung

Abb. 4 MySql
(10)

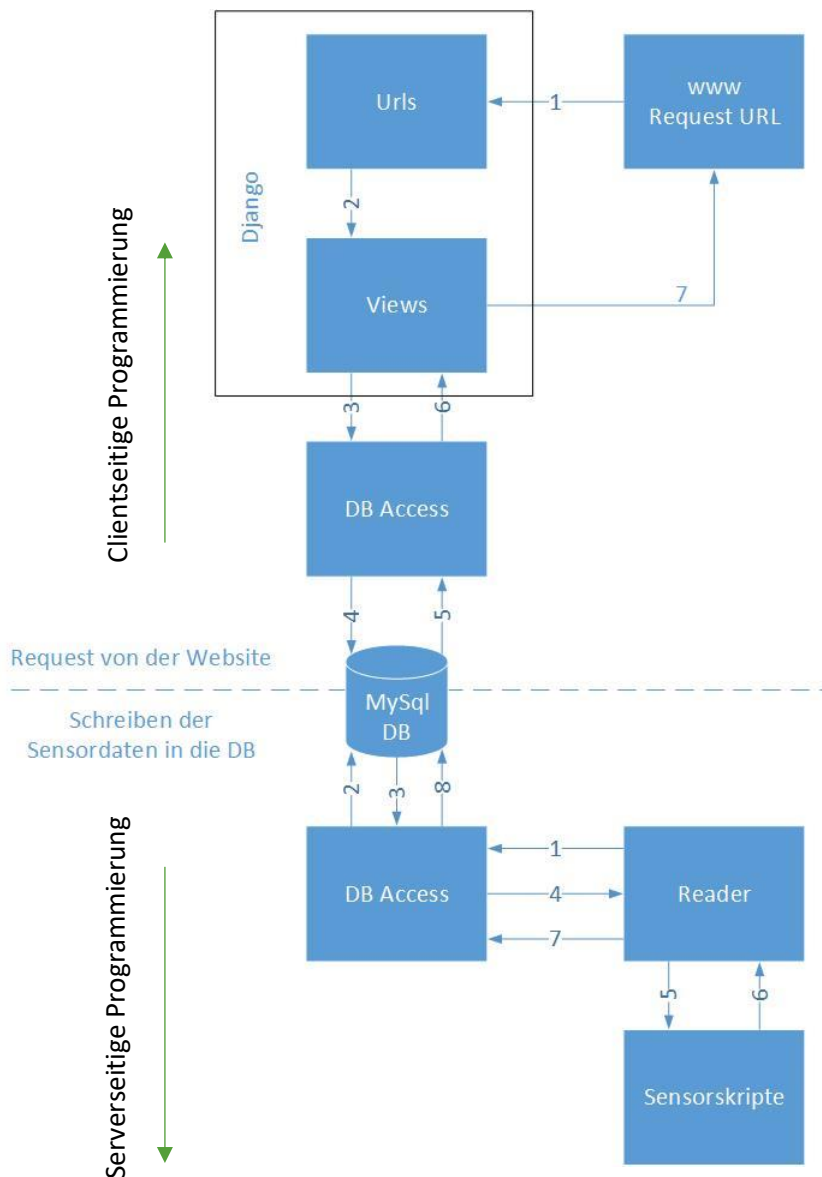
MySQL ist eine der beliebtesten Datenbanken für Datenspeicherungen von Webseiten. Die in 2010 von Oracle übernommene Opensource-Software wird dank dem Ableger Maria DB auch in Zukunft kostenlos zur Verfügung stehen. (4)

Standardmässig verwendet Django die SQLite DB. Durch die Flexibilität des Frameworks können aber verschiedene Datenbanken benutzt werden. Durch die Komplexität der Tabellen und aufgrund des Vorwissens der MySQL-Struktur haben wir uns für die Umstellung auf MySQL entschieden.

10.4 Programmstruktur Serverseite

10.4.1 Einführung in die Programmstruktur

Im Folgenden wird detailliert auf die Serverseite eingegangen. Mit Serverseite ist hauptsächlich der untere Teil der Schematischen Darstellung gemeint, jedoch gibt es auch in oberen Teil Anwendungen von der Serverseite. Ausgehen vom Reader werden alle Pfeile der Reihe nach beschrieben. Auch zu erwähnen gilt es das die DB Access in der Darstellung zweimal auftaucht. Hierbei handelt es sich um dieselbe DB Access, die dient rein zur vereinfachten Darstellung.



Server Seite

1. Der Reader ermöglicht es periodisch Sensorabfragen durchzuführen.
2. Über das DB Access APP wird eine Verbindung zur DB aufgebaut.
3. In der MySQL-DB wird nachgeschaut welche Sensoren im Projekt zur Verfügung stehen. Diese Information wird an das Access Skript zurückgesendet.
4. Das DB Access Skript bereitet die Daten auf und gibt Sie an den Reader zurück.
5. Der Reader steuert die entsprechenden Sensorskripte an.
6. Die Werte werden ausgelesen und
7. über den DB-Access schliesslich in MySQL-DB abgespeichert.
8. Dort stehen die Daten für den Client bereit.

10.4.2 Reader (SchedulerAPI)

Die Reader Skripte bilden das Herzstück des Programms. Sie definieren, in welchen Zeitabständen Daten von den Sensoren abgefragt werden können.

SchedulerEngin.py

In diesem Skript befindet sich der Herzschlag des Readers. Nach dem Start wird zu jeder vollen Minute das Skript ausgeführt. Es liest die Tasks ein, welche ausgeführt werden sollen. Wenn der Objekttyp vom Typ Sensor ist, wird die Funktion «WriteSensorValueToDB» im «MainSensorReader» ausgeführt.

MainSensorReader.py

In dem Skript befindet sich nur die Funktion «WriteSensorValueToDB». Diese Funktion erwartet die SensorID als Parameter. Anhand des Sensortyp wird entschieden, welches Sensorskript verwendet wird. Auf ein Switch-case haben wir hier verzichtet, da er nicht native in Python vorhanden ist, Python diesen Mechanismus aber imitieren kann.

10.4.3 DB Access

Zu den DB Access Skripten zählen jene Skripte, die im Zusammenhang mit der MySQL DB stehen. Beispielsweise sind das Helper (Helfer) Skripte, die nur eine Verbindung zur DB auf- und wieder abbauen. Andere haben komplexere Funktionen, auf die in diesem Kapitel eingegangen wird.

MySQLHelper.py (Veraltet)

Diese Klasse öffnet die Datenbankverbindung zum MySQL-Server. Mit dem Umstellen der Logik auf die Django-Models wurde die Verbindung neu über das Django-Framework geregelt.

10.4.4 Info und Info-Provider Pattern

Für jede Datenbanktabelle haben wir eine Info-Klasse erstellt. Diese Klasse ist ein Django-Model und befindet sich im gleichen Python Modul wie der dazugehörige Info-Provider. Das Modul ist im APP DB-Access abgelegt. Der Info-Provider bietet Funktionen, um die Daten aus der DB abzufragen und in sein dazugehöriges Info-Objekt abzuspeichern. Beispielsweise bietet der ProjektInfoProvider eine Methode an, welche das aktive Projekt zurückgibt. Generell hat jeder Provider eine Methode, welche ein einzelnes Infoobjekt über die ID ausliest, sowie eine Weiter-Funktion, welche eine Liste der Info-Objekte zurückgibt.

Der SensorInfoProvider hat z.B. die Methode GetSensors. Ihr können eine SQL Where-Kondition oder eine «Order by» Klausel mitgegeben werden.

Der InfoProvider soll den Zugriff auf die DB vereinfachen und nach aussen abkapseln. So kann ein Programmierer einfacher Daten abfragen, ohne dass er sich mit der Architektur des Django-Models auskennt.

10.4.5 MySQL

Hier werden alle Sensordaten gespeichert und können wieder abgefragt werden. Die DB ist also die zentrale Datenverwaltung des Projekts.

10.4.6 Sensorskripte

Die Sensorskripte stellen die direkte Kommunikation zur Hardware sicher. Dafür wurden im speziellen drei Skripte geschrieben.

- DHT22.py, dieser Sensor kann zum Auslesen der Luftfeuchtigkeit und der Temperatur verwendet werden.
- Relais.py, direktes Ansteuern des 8 Kanal Relaismodul.
- SoilMoisture.py, ein Skript, dass den Wert der Bodenfeuchtigkeit ausgibt.

Zum SoilMoisture und DHT22 Skript wurden jeweils identische Skripte geschrieben, die aber einfach zufällige Werte zurückgeben. D.h. man kann an der Software arbeiten, ohne direkt am RPI und somit an den Sensoren angeschlossen zu sein und dennoch Werte für einen Test erhalten. Wir haben dafür die Python Funktion 'Random' verwendet.

11 Hardware

11.1 Einleitung

Die Arbeit basiert auf einem RPI. Diese kleinen Computer sind perfekt für ein Projekt wie unseres. Sie bieten eine Vielzahl an Schnittstellen und können mit einer grossen Anzahl an erprobten Sensoren kombiniert werden.

11.2 Aufbau Gewächshaus

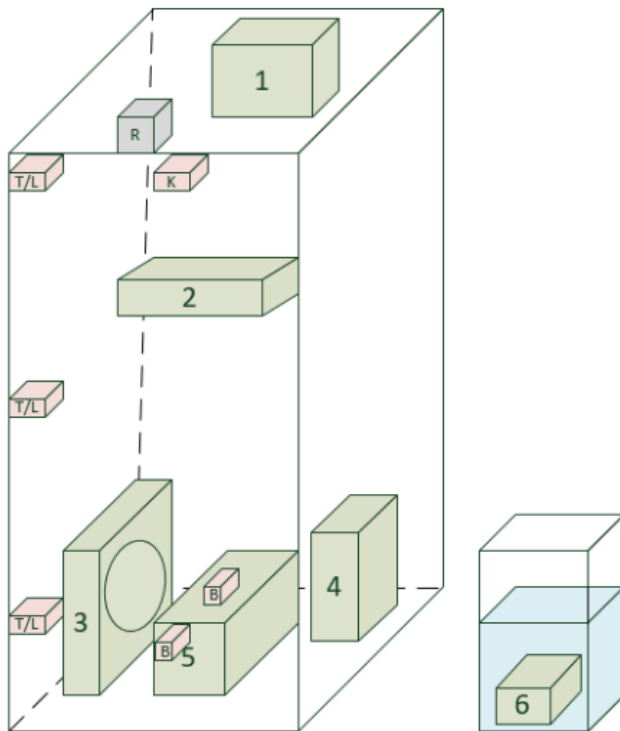


Abb. 5 Aufbau Gewächshaus

Rot sind die verbauten Sensoren

K = Kamera

T/L = Temperatur- und Luftfeuchtigkeitssensor

B = Bodenfeuchtigkeitssensor

Grau (R) ist die Box mit dem RPI. Sie beinhaltet auch die Relais zum ein und ausschalten der Geräte.

Die Elemente 1-6 werden in der Einleitung erläutert.

11.3 Raspberry Pi (RPI)

Der RPI 3B ist für viele Anwendungen die erste Wahl. Durch seine vielseitigen Schnittstellen ist er flexibel einsetzbar. Wo auch immer ein kleiner Computer gebraucht wird, der mit wenig Strom betrieben werden kann, ist er nicht weit. Der RPI ist das Herz unseres Projektes. Das Django Framework wird darauf laufen sowie unsere DB. Dank seiner WLAN Tauglichkeit ist es einfach mittels Portweiterleitung auf ihn zuzugreifen und so die Webseite über das Internet abrufbar zu machen.



Abb. 6 Raspberry Pi (7)

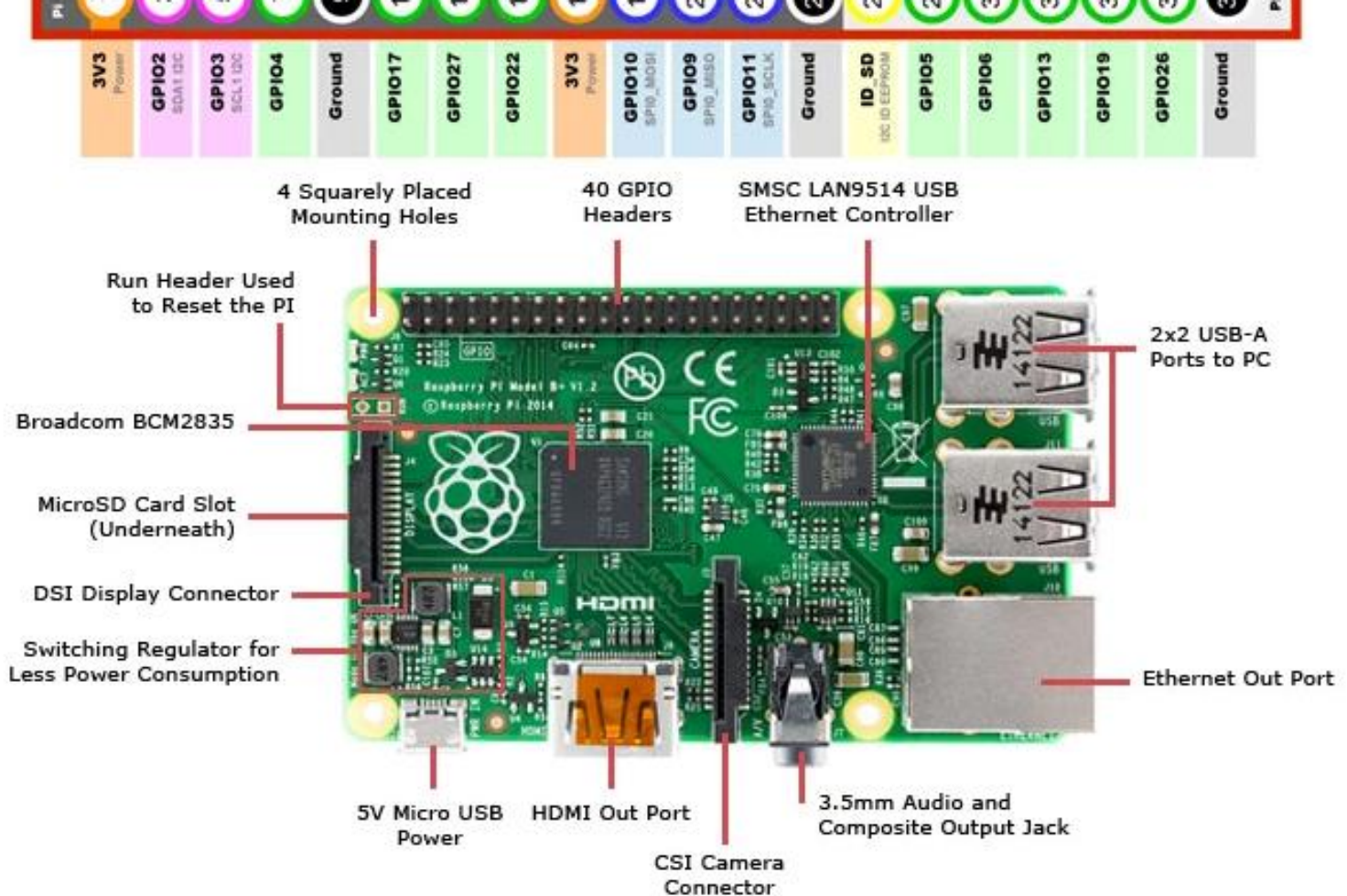


Abb. 7 GPIO
(9)

11.3.1 Schnittstellen

Das Bild im oberen Abschnitt zeigt alle Schnittstellen des RPI. Die für die Arbeit relevanten Schnittstellen werden hier erläutert. Die meisten Sensoren brauchen eine Spannungsversorgung, die z.B. über die 5V Pluspol und GND Minuspol realisiert wird. Abfragen des Sensors werden mittels dem GPIO gemacht.

- **GPIO** (General purpose input/output) sind die Pins, über die verschiedene Sensoren oder LEDs angesteuert werden können. Sie sind frei programmierbar.
- **3V3** und **5V** sind die Pins, die für die Spannungsversorgung benötigt werden. Analog zu ihrem Namen liefern die einen 3,3 Volt und die anderen 5 Volt. Ihr Einsatz hängt vom verwendeten Sensor ab.
- **GND** (Ground) ist der Minuspol.
- **SDA** (Pin3) und **SCL** (Pin 5) gehören zur Schnittstelle I²C. Sie werden benötigt, um einen Analog/Digital Konverter anzuschliessen.
- Ein **WLAN-Modul** ist beim RPI 3B Standard und ermöglicht ein einfaches Einbinden in das Netzwerk.

- **DNS Server auf RPI** Dazu haben wir in unserem Projekt den DNS Dienstleister no-IP benutzt. Natürlich können auch andere Dienstleister genutzt werden. Auf der Seite «noip.com» kann man sich registrieren und eine Webseite auswählen. Danach kann der DNS Dienst auf dem RPI eingerichtet werden. Dazu muss man das Paket von noip.com installieren.
 1. Mit “sudo wget <http://www.noip.com/client/linux/noip-duc-linux.tar.gz>” herunterladen.
 2. Entpacken mit „sudo tar xf noip-duc-linux.tar.gz“
 3. In den Ordner «noip-2.*» wechseln und mit «sudo make install» installieren.
 4. Nun wird man nach dem Benutzernamen und Passwort von noip.com gefragt. Gibt man dieses ein, kann man gleich zwischen den verfügbaren DNS Adressen auswählen. Jetzt kann der Service mit dem Befehl: sudo noip2 gestartet werden. Am Besten fügt man diesen Befehl in den Autostart ein.

11.3.2 Raspberry Pi Software-Installation

Der RPI wurde mit dem Raspbian Betriebssystem Stretch aufgesetzt. Unter diesem Betriebssystem kann nur Maria DB installiert werden. Da diese DB fast der MySQL DB entspricht, war das kein Hindernis. Weiter wurde Python 3.6 und Django 2.1.5 installiert.

11.4 DHT22

Der DHT22 ist ein Sensor zum Messen von Temperatur und Luftfeuchtigkeit. Im Projekt werden drei davon eingesetzt um diese Werte an verschiedenen Positionen im Gewächshaus messen und vergleichen zu können.

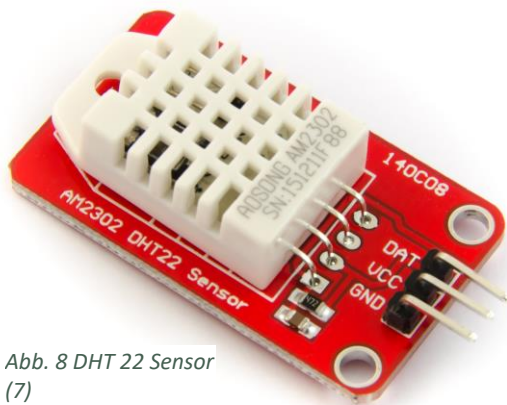


Abb. 8 DHT 22 Sensor
(7)

Anschlüsse:

- VCC = 5V oder 3.3V
- Dat = GPIO
- GND = GND

Technische Details:

- Messbereich Luftfeuchtigkeit:
0-100% bei 2-5% Abweichung
- Messbereich Temperatur:
-40-125°C +/- 0.5°C
- Antwortzeit ca. 2 Sekunden

11.4.1 Anschlussschema DHT22

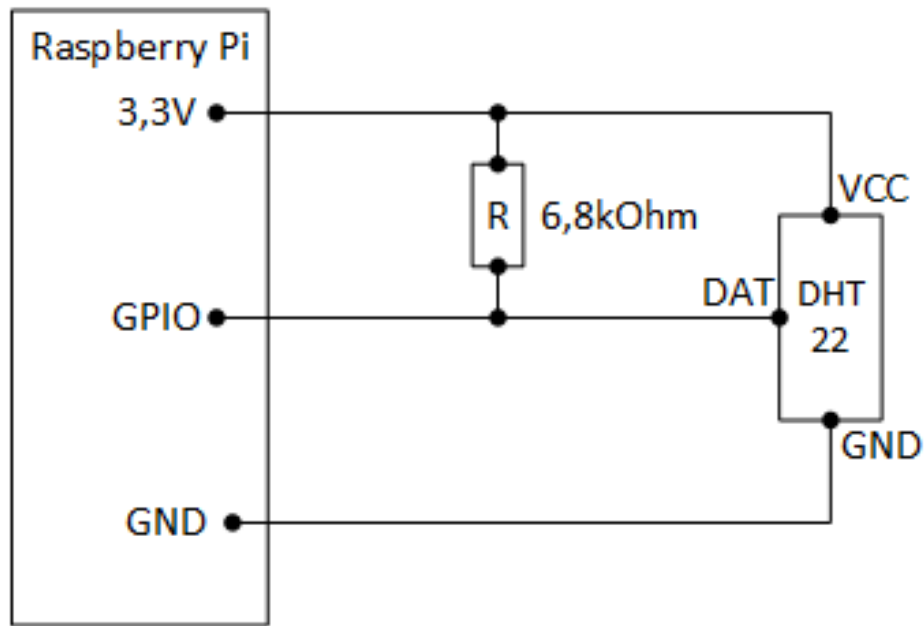


Abb. 9 DHT22 Schema

(5)

11.5 Bodenfeuchtigkeit

Um die Bodenfeuchtigkeit zu bestimmen wird ein kapazitiver Sensor eingesetzt. Da dieser analog funktioniert wird zusätzlich ein Analog/Digital Konverter eingesetzt.

11.5.1 Analog/Digital Konverter

Der 16bit analog/digital Konverter ADS1115 wurde bei der Firma Bastelgarage.ch erstanden. Er verfügt über vier analoge Eingänge und kommuniziert über SCL und SDA mit der I²C Schnittstelle.

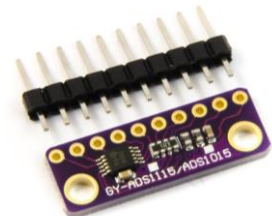


Abb. 11 ADC (7)

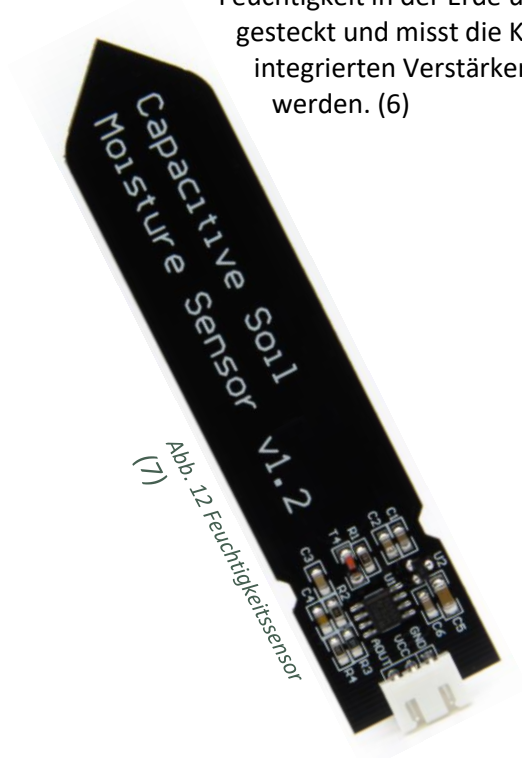


Abb. 10 ADC Board (7)

Pinbelegung:
 V = 3.3 Volt
 G = GND
 SCL = Serial Clock
 SDA = Serial Data
 ADDR = Address
 Alert = Alert
 A0 = Analog In 0
 A1 = Analog In 1
 A2 = Analog In 2
 A3 = Analog In 3

11.5.2 Bodenfeuchtigkeitssensor

Der kapazitive Feuchtigkeitssensor wurde bei der Firma Bastelgarage.ch erstanden. Dieser misst die Feuchtigkeit in der Erde um den Giesszeitpunkt zu bestimmen. Der Sensor wird ins Erdreich gesteckt und misst die Kapazität der Erde, die sich mit der Feuchtigkeit ändert. Durch einen integrierten Verstärker kann er direkt an den analog/digital Wandler angeschlossen werden. (6)



Pinbelegung:

VCC = 3.3 Volt

GND = GND

AUDT = Analog Output

11.6 Anschlussschema Bodenfeuchtigkeitssensor

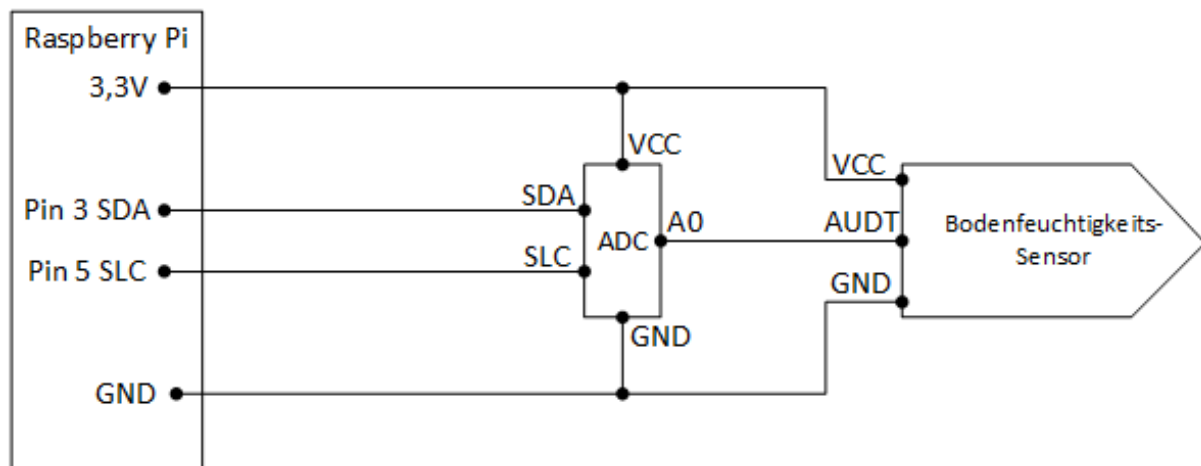


Abb. 13 Feuchtigkeitssensor

(6)

11.7 Relais

Relais sind elektromechanische Schalter. Man kann damit Geräte ein- und ausschalten, die am 230V Netz betrieben werden. Statt eines Fingers, der ein Schalter betätigt, verwendet der RPI seine 5V um ein Gerät ein- oder auszuschalten. Mit den Relais können wir also z.B. unsere Lampe, die Tauchpumpe oder die Ventilatoren ein- und ausschalten.



Pinbelegung:

GND

In1

In2

In3

In4

In5

In6

In7

In8

VCC

Abb. 14 Relaisboard
(7)

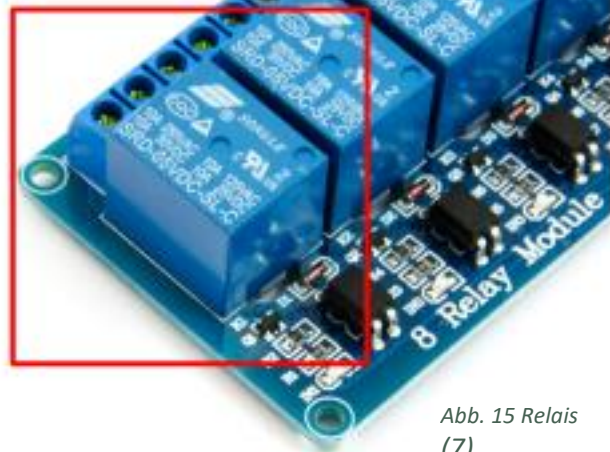
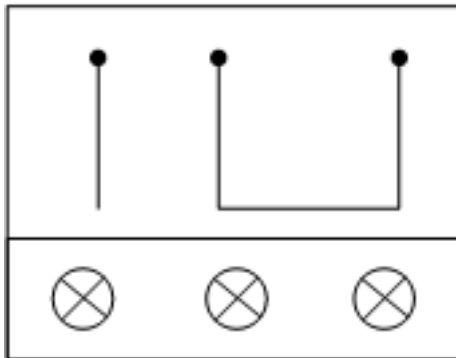


Abb. 15 Relais
(7)

Hier kommt der Leiter (L) des zu schaltenden Geräts hin, wenn beim Schalten des Relais ausgeschaltet werden soll.

Hier kommt **immer** der Leiter (L) aus der Steckdose hin.

Hier kommt der Leiter (L) des zu schaltenden Geräts hin, wenn beim Schalten des Relais eingeschaltet werden soll.

11.7.1 Anschlussschema Relais

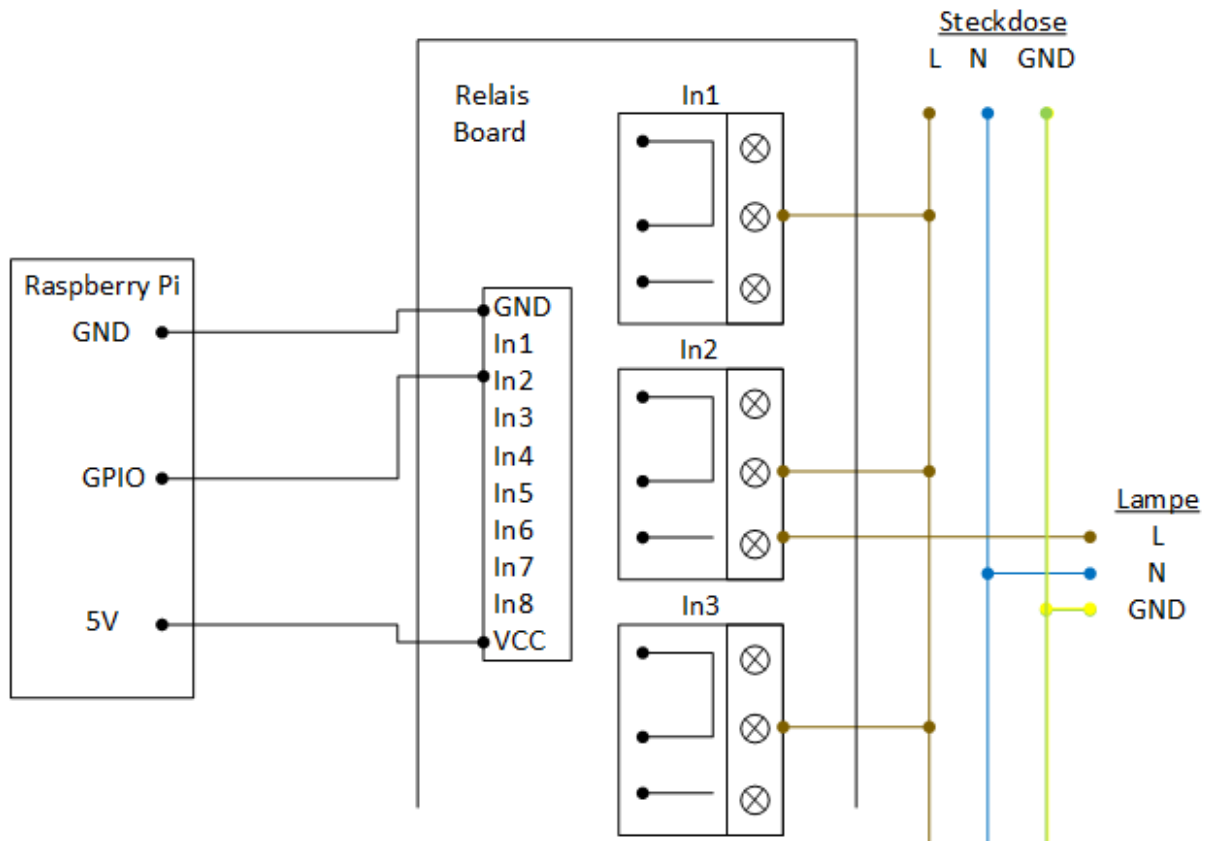


Abb. 16 Relais Schema

(8)

11.8 Kamera

Die Kamera wird am RPI direkt am CSI Camera Connector mit dem mitgelieferten Kabel angeschlossen. Um die Kamera anschliessend nutzen zu können, müssen noch einige Einstellungen vorgenommen werden. Zunächst sollte der RPI auf dem neusten Stand mit Updates sein. Ist dies gemacht, kann man mit dem Befehl «sudo raspi-config» im Terminal auf die Konfiguration des RPI zugreifen. Hier wird unter Interface Options die Kamera aktiviert.

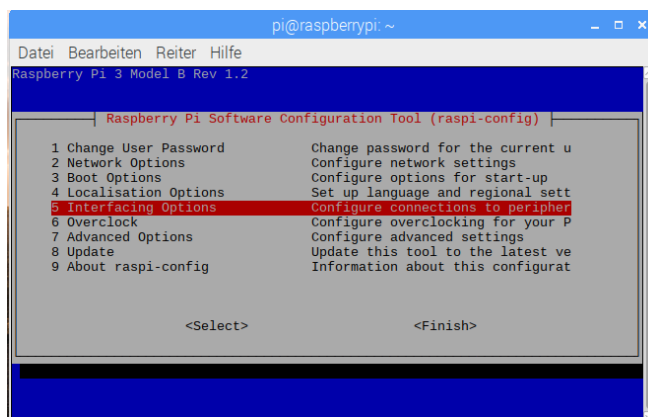


Abb. 17 Raspberry Pi Configuration

Nun kann man mittels Terminal Fotos oder Videos machen:

Fotos:

```
raspistill -o image.jpg
```

Videos:

```
raspivid -o video.h264 -t 10000
```

Wichtig bei den Videos ist, dass das t in Millisekunden angegeben ist.

11.8.1 Kamera als Livestream einrichten

Man kann hier nicht nur die Kameras, die über den CSI Connector Camera angeschlossen sind verwenden, sondern auch USB-Kameras. Die Konfiguration ist in beiden Fällen gleich.

Zunächst benötigt man das Tool Motion. Dies kann mit folgendem Befehl installiert werden:

```
«sudo apt-get install motion -y».
```

Um die Kamera nun anzuzeigen kann folgender Befehl verwendet werden: «ls /dev/video*».

Nutzt man die Originalkamera wie in unserem Fall, so benötigt es allerdings noch den folgenden Befehl, damit sie korrekt angezeigt wird: «sudo modprobe bcm2835-v4l2». Diesen Befehl sollte man am besten dem Autostart hinzufügen, ansonsten würde der Livestream beim nächsten Neustart oder bei einem unerwarteten Neustart nicht mehr funktionieren.

Mit dem Befehl «v4l2-ctl -V» kann man nun die Werte der Kamera auslesen. Diese werden später in der Konfigurationsdatei benötigt. Die Datei wird auch gleich konfiguriert. Dazu mit dem Befehl «sudo nano /etc/motion/motion.conf» die Konfigurationsdatei öffnen.

Hier werden folgende Anpassungen gemacht:

```
# Start in daemon (background) mode and release terminal (default: off)
daemon on
...
# Restrict stream connections to localhost only (default: on)
stream_localhost off
...
# Target base directory for pictures and films
# Recommended to use absolute path. (Default: current working directory)
target_dir /home/pi/Monitor
```

In den folgenden Zeilen kann man noch die Kameraeinstellungen ändern:

```
v4l2_palette 15 # Nummer aus der Tabelle davor entnehmen, 15 entspricht YUYV
...
# Image width (pixels). Valid range: Camera dependent, default: 352
width 640

# Image height (pixels). Valid range: Camera dependent, default: 288
height 480 # Maximum number of frames to be captured per second.

# Valid range: 2-100. Default: 100 (almost no limit).
framerate 10
```

Anschliessend wird die Datei gespeichert und verlassen.

Als nächstes ändert man die Datei «motion» mit dem Befehl «sudo nano /etc/default/motion».

Hier wird beim Befehl «start_motion_daemon=no» das „no“ durch ein „yes“ geändert.

Der Livestream bildet sich mit den vorhergehenden Einstellungen aus 10 Bildern pro Sekunde. Damit dies möglich ist, müssen die Bilder zwischengespeichert werden. Der Pfad dafür ist oben bereits angegeben.

Um das Verzeichnis zu erstellen, führt man den Befehl «mkdir /home/pi/Monitor» aus. Anschliessend wird noch die Gruppe auf «motion» geändert. Dazu den Befehl «sudo chgrp motion/home/pi/Monitor» ausführen. Nun kann man der Gruppe noch die Lese-, Schreib- und Ausführungsrechte geben.

Somit sind nun die Einstellungen fertig und man kann den Service starten. Dazu den folgenden Befehl eingeben: `sudo service motion start`.

Den Livestream kann man nun in einem Internet Browser ansehen und gibt dazu folgende Adresse mit dem Port 8081 (diesen Port kann man in der Datei «motion.conf» anpassen) ein:

http://raspberrypi:8081/ oder

IP Adresse von RPI: 192.168.0.60:8081

12 Kontrolle

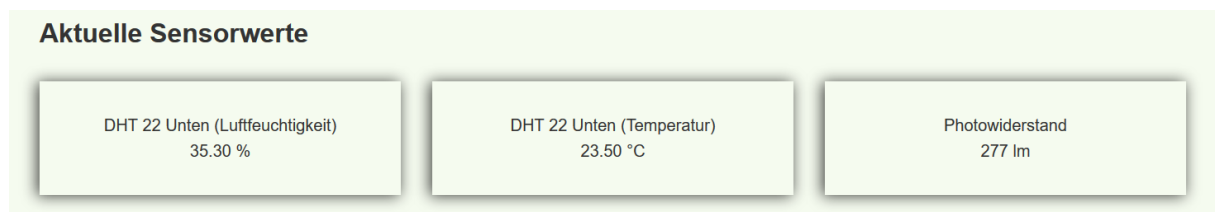
Bedingt durch die technischen Herausforderungen, so wie das aufwendige Projektmanagement, haben wir die Abschlusskontrolle am letzten Projekttag durchgeführt. Während der Implementation haben wir immer wieder die einzelnen Module getestet. Die Landingpage wurde aus zeitlichen Gründen weggelassen. Die Funktionsnavigation mit Kontakt- und Logout-Link wäre eigentlich angedacht, aber ebenfalls nicht implementiert.

Die Profilseite ist implementiert, jedoch hat die Zeit nicht mehr gereicht, um die Logik sauber abzuschliessen.

12.1 Allgemeinte Feststellungen bei der Schlusskontrolle

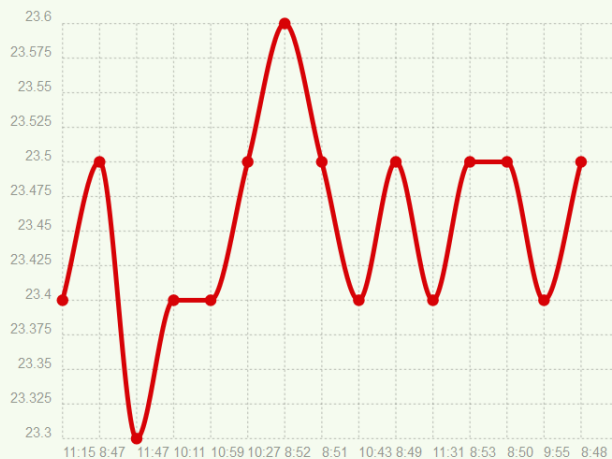
Das Programm ist lauffähig, es fehlen jedoch Komfortfunktionen, wie z.B. ein Zurück-Button bei den Editierfunktionen. Teilweise wird noch nicht angezeigt, dass ein Wert gespeichert wurde. Die Tabellen mit den Werten sind nur begrenzt responsive.

12.2 Test der Startseite



Die aktuellen Werte werden auf der Startseite angezeigt. Jedoch wird beim Photowiderstand bis jetzt noch ein Zufallswert angezeigt, da es ein Fehler in der Verdrahtung des Sensors gibt und dieser darum keine Werte liefert.

Sensorwerte vom 3.2.2019



Die Sensorwerte werden soweit schön im Diagramm dargestellt. Etwas unschön ist noch, dass die Achsen nicht beschriftet sind und die Uhrzeit nicht chronologisch ist. Diese Werte werden bereits von der Webmethode in dieser Reihenfolge geliefert, was auf ein Fehler in der Datenbankabfrage hinweist.

12.3 Test der Projektseite

Ein Projekt kann erstellt, bearbeitet oder gelöscht werden. Wenn man ein Projekt löschen möchte, wird eine Bestätigung der Löschung verlangt. Es kann nur ein Projekt aktiviert werden. Dies wird programmiertechnisch geprüft, bzw. wird ein zweites Projekt aktiviert, wird das aktive Projekt automatisch deaktiviert. Diese Funktionalität wurde durch Tests verifiziert.

12.4 Test der Sensorseite

Liste der Sensoren

Neuer Sensor erfassen

Sensor ID	Sensorname	GPIO		
1	DHT22T_t	14		
2	DHT22M_t	14		
3	DHT22B_h	4		
4	DHT22B_t	4		
5	photowiderstand	4		

Es könne neue Sensoren erfasst werden. Beim Löschen und Bearbeiten wird jedoch eine Benachrichtigung angezeigt, ob die Daten effektiv gespeichert wurden.

Sensor dem aktiven Projekt hinzufügen

Sensor

Speichern

Bei der Zuweisung eines Sensors hat es noch folgenden Fehler:

Wenn der gewünschte Sensor im Dropdown-Menü ausgewählt wurde und anschliessen auf „Speichern“ geklickt wird, erscheint die Fehlermeldung, dass kein Sensor ausgewählt wurde. Erst wenn man nach der Auswahl irgendwo neben das Dropdown-Menü klickt und anschliessend speichert, funktioniert es. Diesen Fehler konnten wir bis anhin leider nicht reproduzieren.

12.5 Test der Relais-Seite



The screenshot shows a web form titled "Relais Zuweisen". It contains a label "Relais" above a dropdown menu. The dropdown menu currently displays a dashed line "-----". Below the dropdown is a green button labeled "Speichern".



The screenshot shows the same web form, but the dropdown menu has been replaced by the text "ProjectRelaisInfo object (2)". The "Speichern" button remains below it.

Auf der Relais-Seite hat es den gleichen Fehler wie bei der Sensorseite. Zusätzlich wurde beim Speichern das Dropdownmenu durch «ProjectRelaisInfo object (3)» ersetzt. Problem war, dass die Forminstanz beim Speichern mit der Modelinstanz des Formulars überschrieben wurde. Dies konnte behoben werden.

Doppelzuweisungen sind nicht möglich, es wird aber keine Fehlermeldung ausgegeben. Es geschieht lediglich nichts.

12.6 Test der Seiten geplante Aufgaben und Event-Log

Diese zwei Seiten sind so aufgebaut, dass es lediglich eine Hilfe für den Programmierer darstellt. Beim Erfassen der geplanten Aufgaben wird keine Benachrichtigung, ob die Daten gespeichert wurden, ausgegeben. Ebenfalls wird noch ein Textfeld angezeigt, in welchem man den Zeitintervall für die geplante Aufgabe als JSON eingeben muss.

13 Abgabe

13.1 Installationsanleitung

Auf dem USB-Stick befindet sich ein Ordner mit dem Name „Setup“. In diesem ist die erste Installationsanleitung abgelegt. Diese war noch für das Betriebssystem Jessie angedacht und diente lediglich intern dazu, dass alle den gleichen Stand auf ihrem RPI hatten.

Da wir erst mitte Januar von dem Betriebssystem Jessie auf Stretch umgestiegen sind, konnten wir dort keine detaillierte Anleitung erstellen.

Es war aber massiv einfacher als das ursprüngliche Setup. Da MariaDB bereits installiert und die Python Version 3.5 ebenfalls kompatibel mit dem Django Framework war, mussten wir nur noch gewisse Plugins installieren.

13.1.1 Installation der Plugins

Im Visual Studio ist es möglich ein requirements.txt zu erstellen, welches alle benötigten Erweiterung enthält. Mit dem Befehl «pip3 install -r requirements.txt» lassen sich dann alle benötigten Plugins installieren.

14 Projekt Review

Durch die Komplexität des Projektes konnten wir zum ersten Mal das gelernte Wissen im Projektmanagement anwenden. Es war sehr spannend im Team eine Lösung zu erarbeiten. Auch wenn es nicht immer ganz rund lief, hatten wir doch Spass als Team diese Aufgabe zu bewältigen.

Im Allgemeinen hatten wir zu viele Zeit für das Projektmanagement aufgewendet. Diese Zeit fehlte uns dann beim Einarbeiten in die vielen verschiedenen Technologien.

14.1 Entwicklungsumgebung

Zur besseren Zusammenarbeit wollten wir alle die gleiche Entwicklungsumgebung benutzen. Die Schule stellt dazu das Visual Studio Pro zur Verfügung. Durch die unterschiedlichen Betriebssysteme kam es bei der Installation zu einem Mehraufwand.

14.2 Raspberry Pi Installation

Die Software wurde erst auf einem PC programmiert, um die Komfortfunktionen des Visual Studio nutzen zu können. Die Installation auf dem RPI hat sich aber als zeitaufwendiger herausgestellt als gedacht. Unter dem Betriebssystem Jessie von Raspbian mussten wir erst die korrekte Python-, Django-Framework-, MySQL-Version evaluieren, was uns schon im Vorprojekt viel Zeit gekostet hatte. Als wir dann entschieden die Django-Model zu verwenden, waren wir ebenfalls gezwungen die MySQL Version anzuheben, was zu längeren Recherchen führte. Nach dem Entschluss auf das Betriebssystem Strecht mit MariaDB zu wechseln, gelang die Installation zügig und ohne weiter Fehlfunktionen.

14.2.1 Scheduler Engine

Das Skript zu schreiben war weniger komplex als das Aufschalten auf dem RPI. Dort wurde unsere Haupt-APP nicht mehr gefunden. Beim Recherchieren stellten wir ebenfalls fest, dass fast für jede Django-Version (1.4, 1.6, 1.9) das Vorgehen sich stark unterscheidet. Nachdem wir herausgefunden hatten, wie wir die Engine als Standalone-Skript einrichten konnten, merkten wir, dass auf dem RPI die Zeitzone berücksichtigt werden, aber da wir die Pythonfunktion für das Auslesen des aktuellen Datums verwendeten, wurden entsprechende Warnungen ausgegeben. Diese Korrektur war ebenfalls zeitaufwendig.

14.3 Steigende Komplexität

Am Anfang hatten wir viele Ideen was unsere Software alles können sollte. Uns war jedoch nicht vollständig klar, wie komplex diese Ideen in der Umsetzung sein würden. Für die Programmieranfänger in der Gruppe wurde es schnell zu komplex. Wir teilten uns daher so auf, dass einer sich auf Python fokussiert und ein anderer auf HTML und CSS. Durch den Zeitdruck, welcher durch diverse technische Herausforderungen entstand, haben wir in der Schlussphase das Team neu aufgeteilt. Die Programmieranfänger fokussierten sich auf das Dokumentieren der Arbeit. Der Informatiker konzentrierte sich voll auf das Implementieren der Software.

15 Literaturverzeichnis

1. **Techtag.** [www.techtag.de](https://www.techtag.de/it-und-hightech/arduino-vs-raspberry-pi-wo-liegt-der-unterschied/). [Online] 2019. [Zitat vom: 12. 01 2019.] <https://www.techtag.de/it-und-hightech/arduino-vs-raspberry-pi-wo-liegt-der-unterschied/>.
2. **Wikipedia.** [www.wikipedia.org](https://de.wikipedia.org/wiki/Django_(Framework)). [Online] 2018. [Zitat vom: 04. 01 2019.] [https://de.wikipedia.org/wiki/Django_\(Framework\)](https://de.wikipedia.org/wiki/Django_(Framework)).
3. **Viktor Garske.** www.udemy.com. [Online] [Zitat vom: 07. 01 2019.]
4. **Wikipedia.** [www.wikipedia.org](https://de.wikipedia.org/wiki/MySQL). [Online] 2019. [Zitat vom: 07. 01 2019.] <https://de.wikipedia.org/wiki/MySQL>.
5. **ITtv.** [youtube.com](https://www.youtube.com/watch?v=BSGO9Np-AHc). [Online] 2018. [Zitat vom: 12. 12 2018.] <https://www.youtube.com/watch?v=BSGO9Np-AHc>.
6. **Anthony Cartwrite.** [www.youtube.com](https://www.youtube.com/watch?v=oQ2lz2ZckRQ). [Online] 2018. [Zitat vom: 12. 12 2018.] <https://www.youtube.com/watch?v=oQ2lz2ZckRQ>.
7. **Keller, Philippe.** [Bastelgarage.ch](https://www.bastelgarage.ch/). [Online] 2018. [Zitat vom: 12. 12 2018.] <https://www.bastelgarage.ch/>.
8. **Antisteo.** [www.youtube.com](https://www.youtube.com/watch?v=90XCgZQC7vo). [Online] 2018. [Zitat vom: 12. 12 2018.] <https://www.youtube.com/watch?v=90XCgZQC7vo>.
9. **Jameco.** [https://www.jameco.com](https://www.jameco.com/Jameco/workshop/circuitnotes/raspberry-pi-circuit-note.html). [Online] 2018. [Zitat vom: 12. 12 2018.] <https://www.jameco.com/Jameco/workshop/circuitnotes/raspberry-pi-circuit-note.html>.
10. **MySQL.** <https://www.mySQL.com/de/>. [Online] 2019. [Zitat vom: 07. 01 2019.] <https://www.mySQL.com/de/>.
11. **DNS einrichten.** [https://tutorials-raspberrypi.de/](https://tutorials-raspberrypi.de/webserver-installation-teil-6-dns-server-via-no-ip/). [Online] 2018. [Zitat vom 17.11.2018.] <https://tutorials-raspberrypi.de/webserver-installation-teil-6-dns-server-via-no-ip/>
12. **Livestream.** [https://tutorials-raspberrypi.de/](https://tutorials-raspberrypi.de/raspberry-pi-ueberwachungskamera-livestream-einrichten/). [Online] 2018. [Zitat vom 18.11.2018] <https://tutorials-raspberrypi.de/raspberry-pi-ueberwachungskamera-livestream-einrichten/>

16 Abbildung und Tabellenverzeichnis

Abb. 1 Django Server (3).....	29
Abb. 2 Django MVP (3)	30
Abb. 3 MySQL (10).....	Fehler! Textmarke nicht definiert.
Abb. 4 Aufbau Gewächshaus.....	36
Abb. 5 Raspberry Pi (7)	36
Abb. 6 GPIO (10).....	37
Abb. 7 DHT 22 Sensor (7).....	38
Abb. 8 DHT22 Schema	39
Abb. 9 ADC (7)	39
Abb. 10 ADC Board (7).....	39
Abb. 11 Feuchtigkeitssensor (7)	40
Abb. 12 Feuchtigkeitssensor	40
Abb. 13 Relaisboard (7)	41
Abb. 14 Relais (7).....	41
Abb. 15 Relais Schema	42
Tab. 1 DHT Vergleich	14
Tab. 2 Budget.....	15

17 Anhang und Beilagen auf USB-Stick

17.1 Anhang

- Eigenständigkeitserklärung
- Pflichtenheft

17.2 USB-Stick

- Sensorskripte (Ordner)
- Setup (Ordner)
- VDA_HTML_Template (Ordner)
- Django-Server (Ordner)
- Vorschlag
- Projektstrukturplan
- Arbeitspakete mit Vorgangsliste
- Zeitplan
- Netzplan
- ERM und RM für DB-Planung
- UML
- Testprotokoll