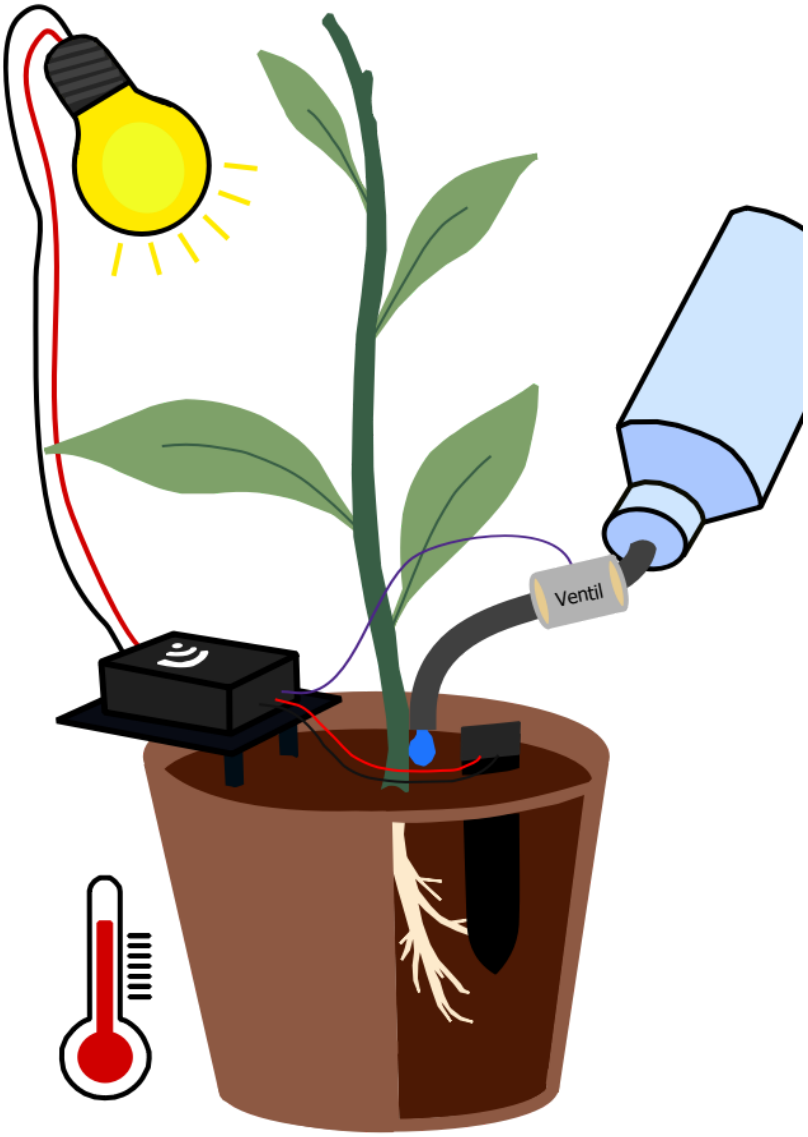
HOCHSCHULE HEILBRONN

Embedded Systems - SS 2020

Smart Garden

Professor: Dr.-Ing. Jürgen Doneit
Dozent: Ulrich Strauß



Mitglieder:
Sebastian Herzog, 197028
Andrés Uribe Stengel, 197033
Timo Volkmann, 199267
Maximilian Seyfried, 197032

Inhaltsverzeichnis

Inhaltsverzeichnis **2**

Abbildungsverzeichnis **4**

1. Idee & Umsetzung **1**

2. Projektplanung **1**

2.1. Organisation des Projekts 1
2.2. Verwendete Teile 3
2.3. Konzept 5
2.4. Steckplan und Schaltplan der Hardware 6

3. Umsetzung **9**

3.1. Die Entwicklungsumgebung 9
3.2. WLAN Verbindung herstellen 9
3.3. MQTT Nachrichten senden und empfangen 11
3.4. Eindeutige Identifikation der Geräte 12
3.5. Persistieren von Einstellungen 12
3.6. Sensor-Loop 12
3.7. Temperatur und Luftfeuchtigkeit messen 12
3.8. Helligkeit messen & Beleuchtungssteuerung 13
3.9. Bodenfeuchtigkeit messen & Bewässerung der Pflanze 14
3.9.1. Kapazitiver Bodenfeuchtesensor (Integrated Sensors) 16
3.9.1.1. Physikalische Grundlagen 16
3.9.1.2. Technische Umsetzung / Sensoraufbau 17
3.10. Luftqualität messen 18
3.11. Web-Applikation mit React 19
3.11.1. Navigation 19
3.11.2. Smart Garden Page 19
3.11.3. Overview Page 20
3.12. Meteor Plattform 22
3.12.1. Was ist Meteor? 22
3.12.2. Wieso Meteor? 22
3.12.3. Wie funktioniert Meteor? 23
3.13. Mongo Datenbank 23
3.14. Data-Collector 23
3.15. Docker 24
3.16. MQTT-Broker & Docker 24

4. Komplikationen **25**
4.1. Hartnäckige Exceptions 25
4.2. WiFi Verbindung verursachte Reboot 25
4.2. ESP-IDF vs. Arduino 25
4.3. Lokal integrierte MongoDB in Meteor 25
4.4. Breadboard ungeeignet für 5V Versorgung 26
4.5. Fehlermeldung bei Konfiguration von Gerät im Frontend 26

5. Ausblick **26**
5.1. Batteriebetrieb & Stromsparmodus 26
5.2. Frontend 27
5.3 Mehrbenutzersystem 27
5.4. Caching und verzögertes Senden von Messdaten 27
5.4. Aufbau 27
5.5 Beleuchtung 28
5.6. Notifications 29

Sonstiges **29**

Quellenangaben **30**
Bilder 30
Links, Querverweise im Text 31

Abbildungsverzeichnis

Abbildung 1: Gantt Chart 1
Abbildung 2: GitLab Issue Board 2
Abbildung 3: AZ-Delivery ESP32 Mikrocontroller 3
Abbildung 4: DHT11 Sensor 3
Abbildung 5: BH1750 Sensor 3
Abbildung 6: Capacitive Soil Moisture Sensor v1.2 4
Abbildung 7: Ventil 4
Abbildung 8: RGB-LED 4
Abbildung 9: Komponentendiagramm 5
Abbildung 10: Steckplan 6
Abbildung 11: Schaltplan 7
Abbildung 12: Gelötete Schaltung oben 8
Abbildung 13: Gelötete Schaltung unten 8
Abbildung 14: Captive-Portal auf einem mobilen Endgerät 10
Abbildung 15: Modi für die unterschiedlichen Wachstumsphasen 13
Abbildung 16: Aktivitätsdiagramm Helligkeitssensor Logik 13
Abbildung 17: Smart Irrigation Aktivitätsdiagramm 15
Abbildung 18: Schaltplan des Bodenfeuchtigkeitssensors 17
Abbildung 19: Navigationbar 19
Abbildung 20: Tabelle mit konfigurierten Geräten 19
Abbildung 21: Form für das Konfigurieren eines Gerätes 19
Abbildung 22: Ansicht für aktiviertes Gerät und Sensordaten 20
Abbildung 23: Dropdown Menü für die Auswahl des aktiven Gerätes 20
Abbildung 24: Overviewpage 21
Abbildung 25: Abweichung der Sensordaten aus dem Sollbereich 21
Abbildung 26: Meteor Architektur 23
Abbildung 27: Gehäuse 28
Abbildung 28: Aufbau Timo 29
Abbildung 29: Aufbau Sebastian 29
Abbildung 30: Aufbau Max 29
Abbildung 31: Aufbau Andy 29

1. Idee & Umsetzung

Das Ziel des Smart Garden Projektes ist die Überwachung und die teilweise automatische Verpflegung einzelner Pflanzen.
Mithilfe von verschiedenen Sensoren und einem Mikrocontroller soll die Temperatur, die Luftfeuchtigkeit, die Bodenfeuchtigkeit sowie die Menge des einfallenden Lichts gemessen werden. Diese Daten sollen dann über WiFi an einen Server gesendet werden.
Mittels einer Web-Applikation sollen die Daten in aufbereiteter Form als Diagramm über einen Zeitraum hinweg betrachtet werden können.
Wenn die Bodenfeuchtigkeit unter einen definierten Grenzwert fällt, soll die Pflanze automatisch gegossen werden. Zusätzlich soll eine Lichtquelle angesteuert werden, falls die Pflanze tagsüber zu wenig Licht erhält.

Die Daten laufen in einer von uns gestellten Server-Infrastruktur zusammen, sodass sich potenzielle Kunden später nicht mit der Technik auseinandersetzen müssen. Es ist lediglich eine Anmeldung und die Registrierung der Geräte notwendig, um komfortable auf das Dashboard zugreifen zu können, auf dem mit einem Blick die gemessenen Werte zu sehen sind. Die Geräte können dann genauso komfortabel über die Weboberfläche konfiguriert werden.

2. Projektplanung

2.1. Organisation des Projekts

In folgender Abbildung, ist die Planung des Projektes als Gantt Chart zu sehen.

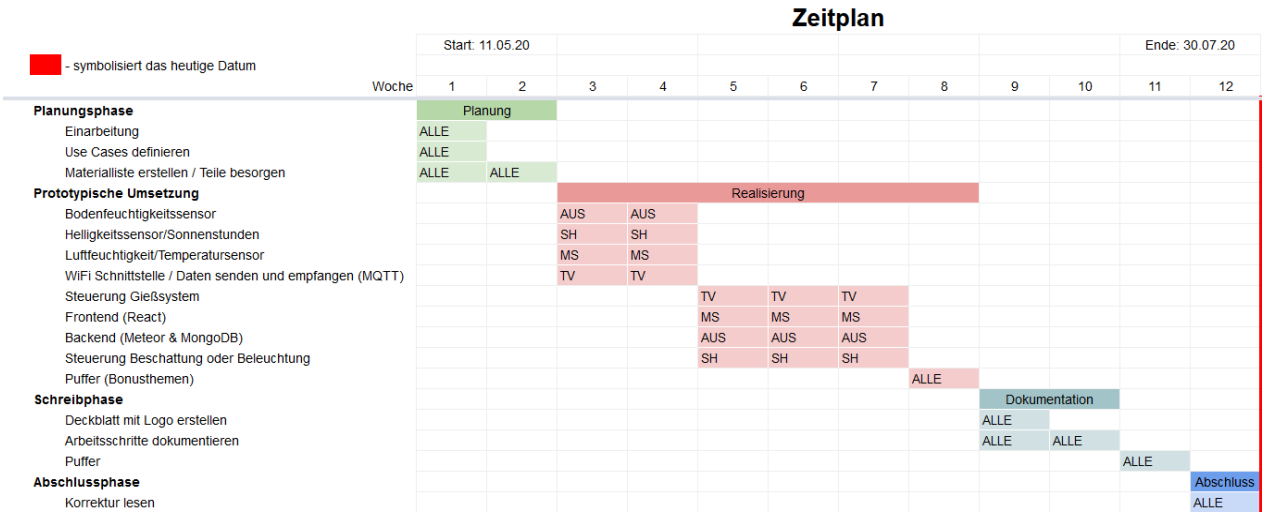


Abbildung 1: Gantt Chart

AUS: Andrés Uribe Stengel
SH: Sebastian Herzog
MS: Maximilian Seyfried
TV: Timo Volkmann

Organisiert wurde das Projekt mithilfe von GitLab und den darin enthaltenen Milestones, Burndown-Charts und dem Issue Board.

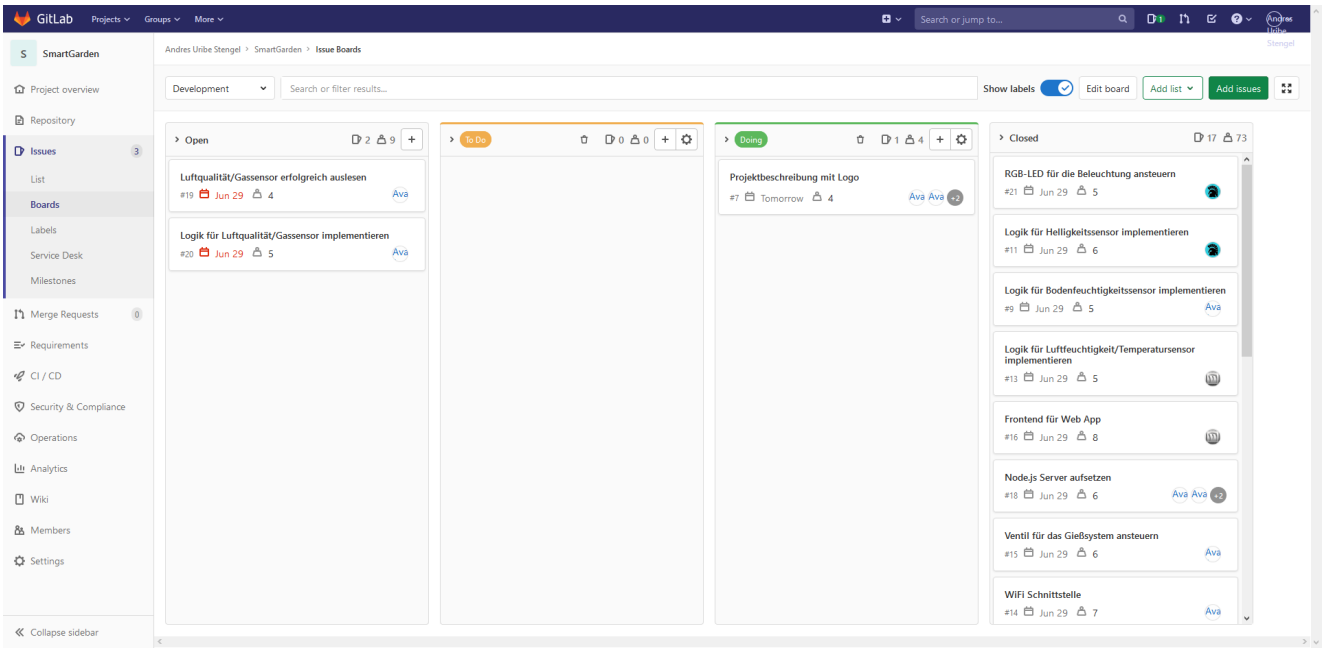


Abbildung 2: GitLab Issue Board

In der obigen Abbildung sind auf der linken Seite zwei noch offene Issues zu sehen. Diese gehören zu einer der möglichen Erweiterungen im Projekt, die Implementierung des Luftqualität/Gas Sensors. In diesem Falle wurde im Projekt beschlossen andere relevantere Erweiterungen anzugehen, wie zum Beispiel das Gehäuse für die Hardware oder das dynamische Lichtspektrum für die verschiedenen Phasen der Pflanze.

2.2. Verwendete Teile

Als Mikrocontroller wird ein ESP32 eingesetzt, da dieser ein eingebautes WLAN Modul besitzt, billig ist und über genug Pins zur Ansteuerung aller Hardwareteile verfügt.
Der ESP32 wird mit einer Spannung von 5 V versorgt werden. Alle Pins bis auf den 5V-Pin haben eine Ausgangsspannung von 3,3 V.

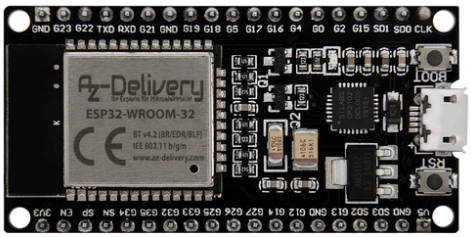


Abbildung 3: AZ-Delivery ESP32 Mikrocontroller

Um die Temperatur und Luftfeuchtigkeit zu messen, wird ein DHT11 Sensor eingesetzt. Dieser benötigt lediglich eine Spannung von 3,3 V. Der Sensor wird digital ausgelesen, kann nach Bedarf aber auch analog gelesen werden.

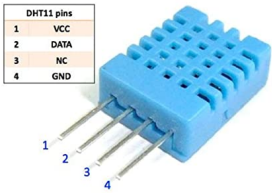


Abbildung 4: DHT11 Sensor

Als Lichtsensor wurde der BH1750 eingesetzt. Dieser misst die Helligkeit in Lux. Die Messung wird in digitaler Form als Wert im Wertebereich von 0 bis 65536 übertragen. Ein Büro hat etwa eine Beleuchtung von 500 lx. Weitere Informationen zur Lux Einheit finden sich hier. Der BH1750 funktioniert bei einer Spannung von 3,3-5 V. Die Kommunikation zum BH1750 erfolgt über das I2C-Protokoll.

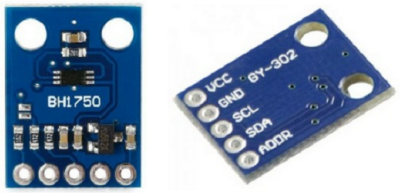


Abbildung 5: BH1750 Sensor

Für die Bodenfeuchtigkeit wird der Capacitive Soil Moisture Sensor v1.2 verwendet. Dieser hat gegenüber anderen Bodenfeuchtigkeitssensoren den Vorteil, dass es keine offen liegenden Metallflächen gibt, wodurch dieser auch nicht korrodiert. Dadurch bleibt der Sensor länger am leben und sondert auch wegen der Elektrolyse keine giftigen Stoffe ab. Für die Stromversorgung werden 3,3-5 V benötigt. Die Messwerte werden analog ausgelesen.

Abbildung 6: Capacitive Soil Moisture Sensor v1.2

Für die Bewässerung wird ein einfaches, elektromagnetische Ventil verwendet. Dieses benötigt eine Spannung von 12 V und verbraucht etwa 250 mA. Im stromlosen Zustand ist das Ventil immer geschlossen.

Abbildung 7: Ventil

Die Beleuchtung wird mithilfe einer RGB-LED simuliert. Der lange Pin ist bei dieser LED Ground. Eine weitere LED dient als Status-Indikator.

Abbildung 8: RGB-LED

2.3. Konzept

Um das Konzept und den Zusammenhang der einzelnen Komponenten zu verdeutlichen, wurde das folgende Diagramm erstellt.

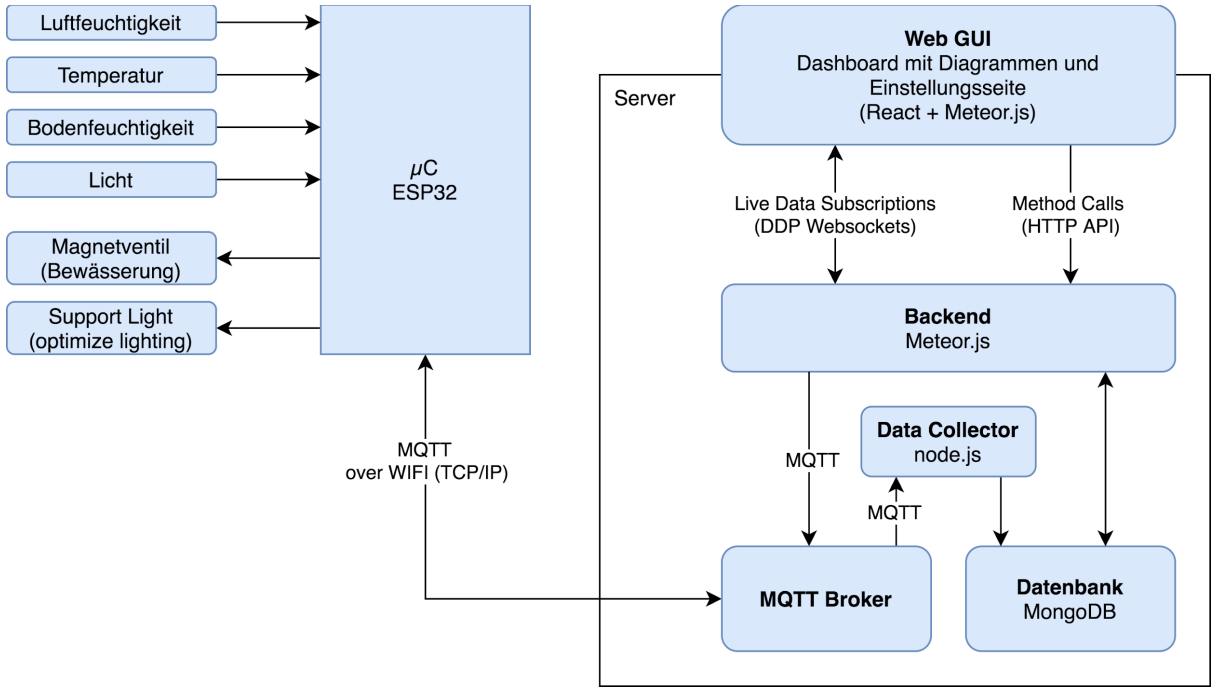


Abbildung 9: Komponentendiagramm

Links in der Abbildung dargestellt sind die Ein- und Ausgänge des Mikrocontrollers zu sehen. Die Eingänge bilden verschiedenen Sensoren, die die Daten der Pflanze ermitteln sollen. Damit eine Automatisierung über den ESP32 realisiert werden kann, sind als Ausgänge das Magnetventil und die zusätzliche Lichtquelle in Form einer RGB LED definiert.
Die Kommunikation zwischen dem Mikrocontroller und dem Serverbereich erfolgt per WiFi und dem Message Queuing Telemetry Transport (MQTT) Protokoll. Die Nachrichten werden zunächst an einen MQTT Broker gesendet, der für die Verteilung und Verwaltung zuständig ist.
Somit werden die Daten dann über MQTT und mithilfe von einem Data Collector von node.js in eine MongoDB-Datenbank gespeichert.
Das Backend ermöglicht die Zugriffe auf die Datenbank und wurde mit Meteor.js realisiert. Für das Frontend wurde eine Web-Applikation mit React umgesetzt, hier erhalten die Benutzenden einen Überblick der aktuell gemessenen Pflanzendaten und ein Diagramm, das die ausgewerteten Daten über einen längeren Zeitraum abbildet. Außerdem sollen über das Frontend Einstellungen vorgenommen werden können, um die Automatisierung auf die eingestellte Pflanze anzupassen. Die vorgenommen Konfigurationen werden ebenfalls über das Backend und anschließend über MQTT zurück an den Mikrocontroller geschickt.

2.4. Steckplan und Schaltplan der Hardware

Neben den bereits erwähnten Komponenten wurden folgende Teile verwendet:

Bauteil	**Stückzahl**
LM2940 Spannungswandler	x1
BD237 Transistor	x1
15Ω Widerstand	x4
68Ω Widerstand	x2
1kΩ Widerstand	x1
4,7kΩ Widerstand	x1
10uF Kondensatoren	x2

Da das Ventil 12 V für die Stromversorgung benötigt, wird der gesamte Aufbau mit 12 V versorgt. Der ESP32 verträgt allerdings nur 5 V. Deshalb wird der Spannungswandler benötigt. Dieser wandelt die 12 V in 5 V um. Die Kondensatoren am Spannungswandler gleichen mögliche Spannungsschwankungen aus.
Geschaltet wird das Ventil über einen Transistor. Erhält der Transistor eine Spannung über den ESP32, wird die GND Verbindung vom Ventil hergestellt und es fließt Strom, wodurch das Ventil geöffnet wird.
Der 1 kΩ Widerstand vor dem Transistor ergibt sich durch die 250 mA die das Ventil verbraucht. Die 15 Ω und 68 Ω Widerstände sind für die RGB-LEDs, damit diese nicht zu viel Spannung erhalten. Der 4,7 kΩ Widerstand zwischen Strompin und Datenpin am DHT11 Sensor dient als Pull-up und sorgt dafür, dass vernünftige Werte gelesen werden.
Im normalen Betrieb benötigt die Schaltung eine Stromstärke von ~140 mA. Wenn das Ventil geöffnet ist, steigt dieser Wert auf ~340 mA.

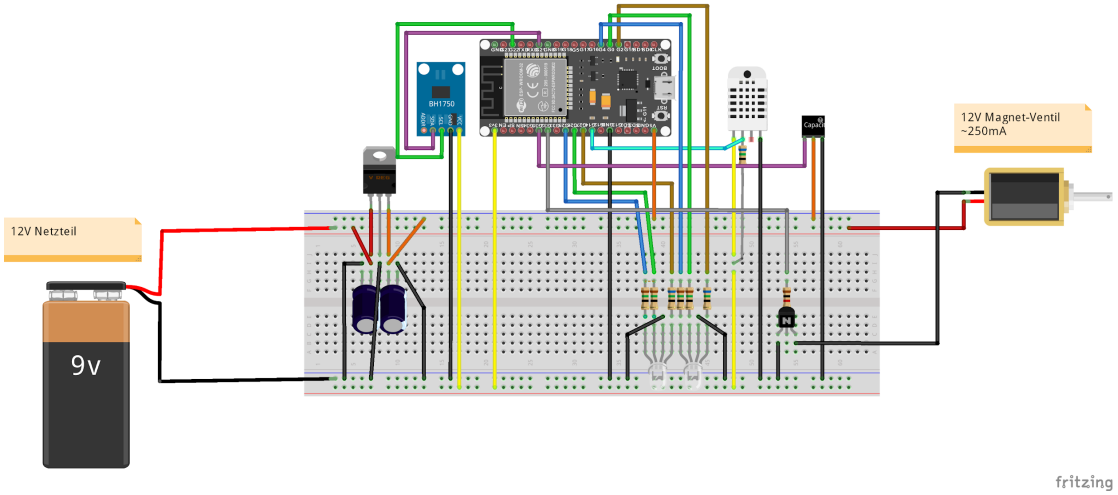


Abbildung 10: Steckplan

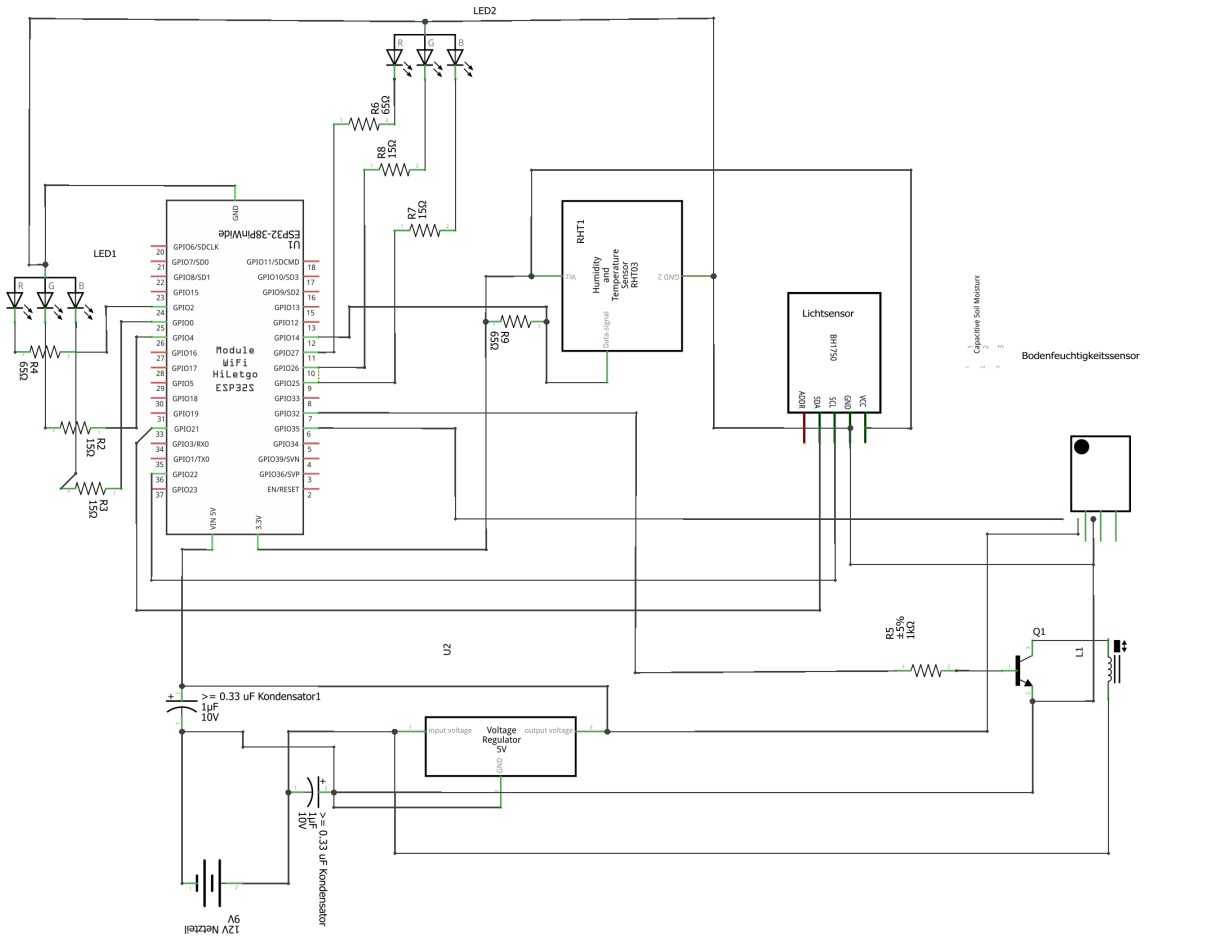


Abbildung 11: Schaltplan

Die fertig verlötete Schaltung unterscheidet sich leicht vom original Schaltplan. An der Stromquelle gibt es nun einen weiteren Kondensator, um sowohl Überspannungen als auch Unterspannungen auszugleichen. Außerdem gibt es einen weiteren Kondensator vor dem Mikrocontroller, um sicherzustellen, dass dieser eine stabile 5 V Versorgung besitzt. Am Spannungswandler befindet sich nun auch ein Passivkühler, da dieser bei offenem Ventil sehr viel Hitze produziert. Für alle externen Teile wie Sensoren und Stromversorgung gibt es Steckplätze. Die Stromversorgung am Gehäuse funktioniert über einen Klinkenstecker

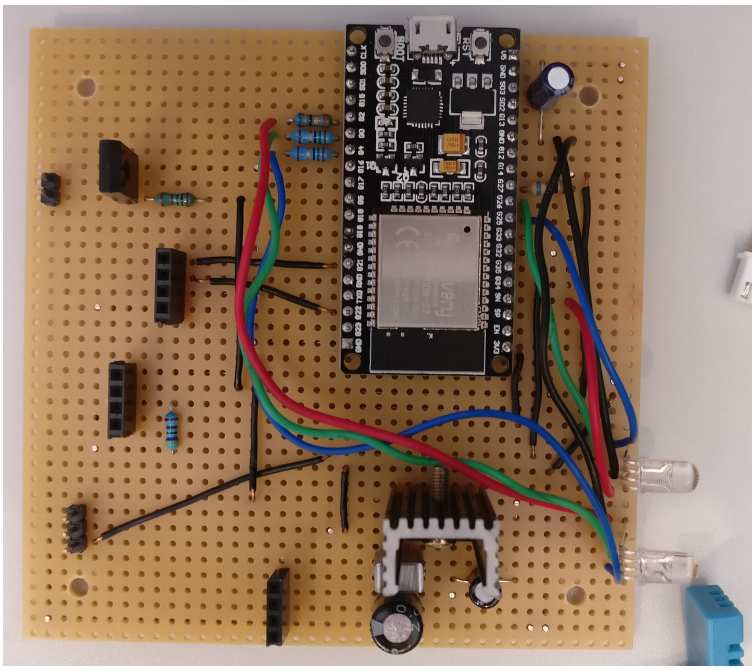


Abbildung 12: Gelötete Schaltung oben

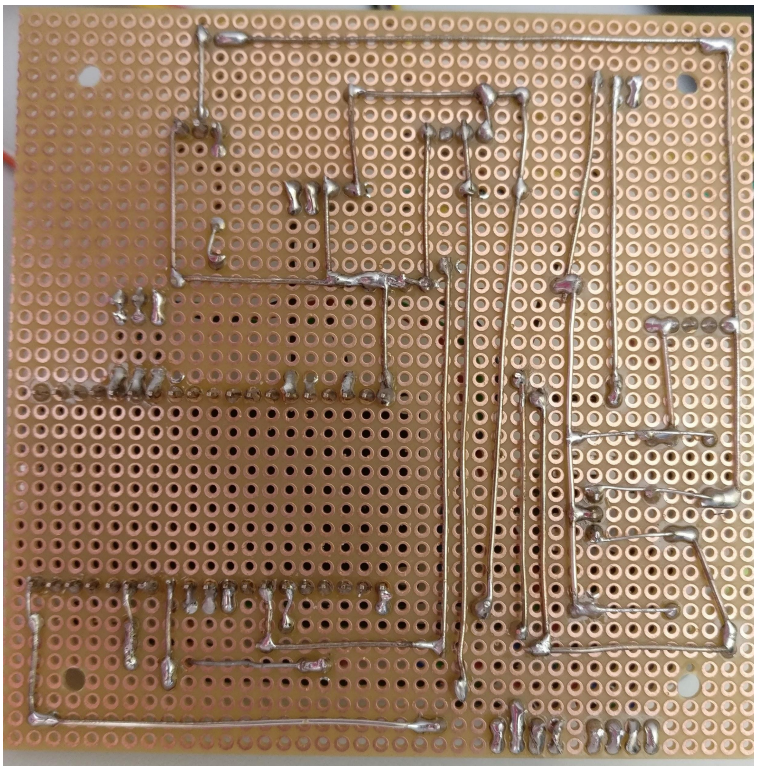


Abbildung 13: Gelötete Schaltung unten

3. Umsetzung

3.1. Die Entwicklungsumgebung

Geplant war zunächst die Verwendung des “Espressif IoT Development”-Frameworks (ESP-IDF), welches auf dem Echtzeit-Betriebssystem FreeRTOS basiert. Nach einem kleinen Testlauf mit dem Espressif Framework wurde die Entscheidung getroffen, das Arduino-Framework zu verwenden, da die Entwicklungszeit für das Projekt damit deutlich reduziert werden konnte.

Als IDE wird PlatformIO verwendet, da diese im Gegensatz zur Arduino IDE auf allen gängigen Plattformen (Linux/MacOS/Windows) läuft und die Verwendung von C/C++ Quellcode erlaubt, statt den Arduino-Sketch-Files. Darüber hinaus bietet diese Entwicklungsumgebung Code-Vervollständigung, komfortables Package-Management und Integration von Build- und Debugging-Tools. Ebenfalls wird der ESP32 ohne weitere Einrichtung von Haus aus unterstützt. Die komplizierte Einrichtung der Toolchain entfällt.

Der größte Mehrwert hierbei ist, dass einzelne Teammitglieder frei in der Wahl Ihres Betriebssystems sind und trotzdem auf eine einheitliche und funktionierende Toolchain zurückgreifen können.

Weiterhin wurden folgende Open-Source Libraries verwendet:

- BH1750
- DHT sensor library
- AutoConnect@^1.1.7
- ArduinoJson@^6.15.2
- PubSubClient@^2.8
- ArduinoNvs@^2.5
- ESPRandom@^1.3.3

3.2. WLAN Verbindung herstellen

Um die WLAN-Verbindung herzustellen, ohne die Zugangsdaten im Quellcode zu hinterlegen und es dem Benutzer so einfach wie möglich zu gestalten eine Verbindung mit einem beliebigen WLAN herzustellen, kamen folgende Optionen in die engere Auswahl:

- WPS-Funktion (WiFi Protected Setup)
- Konfiguration per WiFi Access Point und Captive-Portal

Erstere Variante erscheint zunächst sehr komfortabel, da keinerlei Zugangsdaten eingegeben werden müssen. In der Praxis hat die Erfahrung jedoch gezeigt, dass dies nicht immer reibungslos funktioniert und nicht jeder Router unterstützt diese Funktion.

Zudem muss am Router eine Taste gedrückt werden, um das WPS zu aktivieren. Wenn der Router an einem unzugänglichen Ort steht, erweist sich dies als ungeschickt.

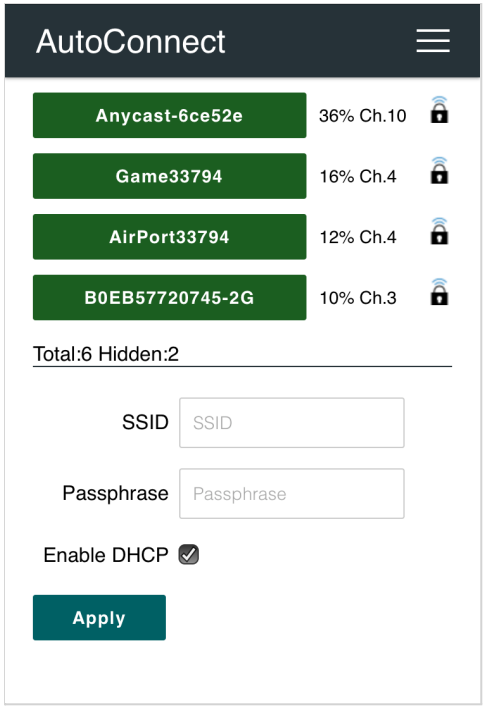


Abbildung 14: Captive-Portal auf einem mobilen Endgerät

Die zweite und von uns bevorzugte Variante erlaubt die Konfiguration von einem beliebigen Gerät mit WLAN und Webbrowser. Die Entfernung zum ESP32 sollte dafür nicht zu groß sein.
Wenn das Gerät in Betrieb genommen wird und keine gültigen Zugangsdaten im Flash-Speicher des Gerätes vorhanden sind, schaltet das WLAN-Modul des ESP32 in den AP-Modus um und startet das Captive-Portal. Wird nun eine Verbindung mit diesem Access Point hergestellt, öffnet sich der Webbrowser und es sind alle Netzwerke in Reichweite des ESP32 zu sehen nachdem das gewünschte Netzwerk ausgewählt und der Schlüssel eingegeben wurde, schaltet der ESP wieder in den Station-Modus und verbindet sich mit dem neu eingerichteten WLAN. Die Zugangsdaten werden auf dem Non-Volatile-Storage (NVS) des ESP32 gespeichert und überstehen somit einen Neustart. Beim zukünftigen Einschalten des Geräts wird somit die Verbindung automatisch wiederhergestellt. Für nähere Informationen steht unser Code und die Dokumentation der verwendeten Library AutoConnect online bereit.

Wird die Verbindung unterbrochen, startet der ESP einen Timer der in regelmäßigen Abständen (momentan alle 2 Sekunden) versucht die Verbindung wiederherzustellen.

Hinweis:
Die verwendeten Timer werden vom Betriebssystem bereitgestellt und nicht durch Interrupts gesteuert. Stattdessen werden diese von einem unabhängigen und nebenläufigen Task verwaltet.
FreeROTS Tasks | *FreeRTOS Timer*

3.3. MQTT Nachrichten senden und empfangen

Die Kommunikation zum Mikrocontroller findet ausschließlich über MQTT statt. Nach jedem Messzyklus published der Mikrocontroller die gemessenen Werte (Licht, Temperatur, Luftfeuchtigkeit und Bodenfeuchtigkeit) in Form eines JSON-Objekts auf dem Topic **smartgarden/updates/{device_id}/data**

Das Anpassen der Einstellungen des Mikrocontrollers geschieht ebenfalls über MQTT-Nachrichten. Die folgende Tabelle zeigt die möglichen Befehle und die erwartete Struktur der Daten. Weitere Details zu den Parametern sind in den entsprechenden Kapiteln zu finden.

Tabelle 1:

Command:	Payload:	Description:

smartgarden/commands/{device_id}/soil	JSON: {
	„pwp“: int,
	„fc“: int,
	„sat“: int
	}
		Schwellenwerte für die Ventilsteuerung
		(siehe Kapitel 3.9)

smartgarden/commands/{device_id}/light	JSON: {
	„minLX“: int,
	„nm“: int
	}
		Schwellenwert für die Beleuchtung und Lichtfarbe
		(siehe Kapitel 3.8)

smartgarden/commands/{device_id}/valve *(wird momentan nicht verwendet)*	–	Manuelle Ventilsteuerung

smartgarden/commands/{device_id}/automatic	JSON: {
	„irrigation“: bool,
	„light“: bool
	}
		Ein- (true) und Ausschalten (false) der Gieß- und Beleuchtungsautomatik

Der MQTT-Client auf dem ESP32 läuft in einem eigenen Task, damit der Programmfluss nicht blockiert, sollte einmal die Verbindung abbrechen.

3.4. Eindeutige Identifikation der Geräte

Damit auch viele Geräte bei der Verbindung mit dem Server eindeutig identifizierbar bleiben, wird beim ersten Start des Geräts eine Device-ID (UUID) erzeugt, die der UUID Version 4 nach RFC4122 entspricht. Die dazu benötigten Zufallszahlen werden von der ESPRandom Library mithilfe des WiFi-Moduls erzeugt.

3.5. Persistieren von Einstellungen

Um die Schwellenwerte nach der Konfiguration über die Weboberfläche und die Device-ID auf dem ESP zu speichern, wurde ein Store implementiert, der sich den NVS des ESP32 zunutze macht. Die Daten werden von der Library ArduinoNVS im Key-Value Format im Flash-Speicher des ESP32 abgelegt.

3.6. Sensor-Loop

Der Sensor-Loop ist ebenfalls ein eigener Task, der in regelmäßigen Abständen die Messungen durchführt. Die Daten werden an den MQTT-Task delegiert und von dort aus an den Server gesendet. Bewässerungs- und Beleuchtungstasks werden ebenfalls durch den Sensor-Loop getriggert.

3.7. Temperatur und Luftfeuchtigkeit messen

Diese beiden Parameter werden durch den Sensor DHT11 gemessen und sind bisher lediglich zu informativen Zwecken implementiert. Das heißt, es wird durch diese Parameter keine Ansteuerung von Aktoren getriggert.
Für bestimmte Pflanzen oder Pflanzenarten sind jedoch Temperatur-Schwellenwerte (Min- und Max-Temperatur) hinterlegt, mit denen der Benutzer dann in der GUI gewarnt wird, wenn für die Pflanze lebensfeindliche Bedingungen herrschen. Weitere Verbesserungen sind auch hier noch geplant (siehe Kapitel 5: Ausblick).

3.8. Helligkeit messen & Beleuchtungssteuerung

Die Helligkeit wird mit dem Sensor-Bauteil BH1750 gemessen im Kontext des Sensor-Loops gemessen. Wenn der Lichtwert unter einen für die Pflanze definierten Schwellenwert fällt, wird ein neuer Task auf dem ESP32 getriggert, der die Beleuchtung einschaltet und kurz bevor eine erneute Bewertung der Lichtsituation ansteht, wieder ausschaltet. Dies geschieht aus dem Grund, dass während die Beleuchtung aktiv ist, keine Bewertung der natürlichen Lichtsituation möglich ist, da die Beleuchtung und der Lichtsensor in der Regel nicht voneinander isoliert sind und diese die Messung des Umgebungslichts verfälschen würde. Die Zusatzbeleuchtung wird nur bei Bedarf, zwischen 8 und 20 Uhr aktiviert, um den natürlichen Rhythmus der Pflanze nicht durcheinanderzubringen. Zusätzlich kann über die GUI das Beleuchtungsspektrum den Bedürfnissen der Pflanze entsprechend angepasst werden.

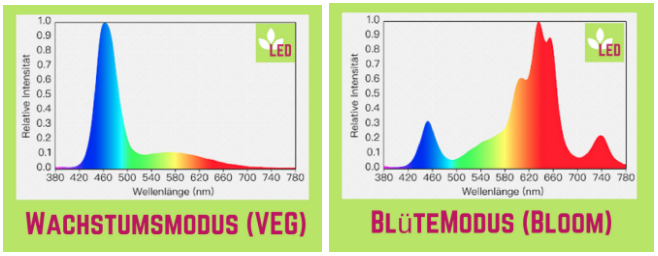


Abbildung 15: Modi für die unterschiedlichen Wachstumsphasen nach led-grow-lampe.com

Weitere Verbesserungen für diesen Mechanismus sind geplant (siehe Kapitel 5: Ausblick).

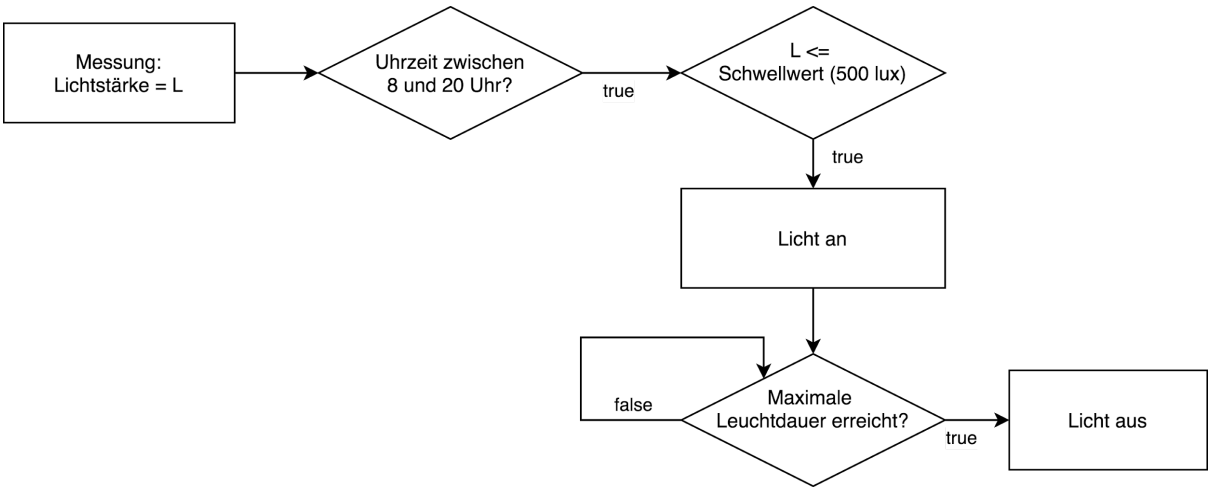


Abbildung 16: Aktivitätsdiagramm Helligkeitssensor Logik

3.9. Bodenfeuchtigkeit messen & Bewässerung der Pflanze

Für die Schwellenwerte der Bewässerung hat sich gezeigt, dass es hier mehr auf den Boden als auf die Pflanze ankommt, da die Pflanze letztendlich selbst das Wasser, das sie benötigt aus dem Boden zieht. Es ist lediglich sicherzustellen, dass dafür genug Feuchtigkeit zu Verfügung steht.
Mit dem permanenten Welkepunkt (PWP) lässt sich die Feuchtigkeit ermitteln, die der Boden mindestens haben muss, damit die Pflanze in der Lage ist Wasser daraus zu gewinnen. Gleichzeitig ist darauf zu achten, dass nicht zu viel gegossen wird, da der Boden nur in der Lage ist, eine bestimmte Menge an Wasser gegen die Schwerkraft zu halten. Alles darüber hinaus würde versickern oder aus dem Topf laufen und wäre damit verschwendetes Wasser. Dieser Schwellenwert wird Feldkapazität (FC) genannt. Der Wert *SoilSaturation (SAT)* beschreibt den Wert, bei dem der Boden vollständig mit Wasser gesättigt ist und keine Luft mehr enthält. Wenn dieser Wert über längere Zeit erreicht oder überschritten wird, ist die Gefahr, dass die Wurzeln faulen sehr hoch.

> **Hinweis:**
> Unter folgenden Links sind ausführlichere Informationen über Wasserhaltefähigkeit des Bodens zu finden. Außerdem eine Wertetabelle für PWP und FC.
> - https://www.dwd.de/DE/fachnutzer/landwirtschaft/dokumentationen/agrowetter/Schlagkonfiguration.pdf?__blob=publicationFile&v=2 (Seite 3)
> - https://www.bodenkunde-projekte.hu-berlin.de/boku_online/pcboku10.agrar.hu-berlin.de/cocoon/boku/sco_6_wasserhaushalt_127cf8.html?section=N100CL

Die Logik der Bewässerung wird vollständig auf dem ESP ausgeführt, um die Bewässerung auch ohne Verbindung zum Server zu ermöglichen. Bei jeder regelmäßigen Messung des Sensor-Loops wird ein neuer nebenläufiger Task getriggert, der den in Abbildung 16 zu sehenden Programmfluss implementiert. Um dem Wasser Zeit zu geben sich im Boden zu verteilen, wird nur für eine kurze Dauer (5 Sek.) gegossen. Um trotz dieser kurzen Bewässerungsdauer die vollständige Feldkapazität des Bodens ausnutzen zu können, wird die Bewässerung trotz Überschreitung des Welkepunkts in den nachfolgenden Zyklen fortgeführt, bis die Feldkapazität erreicht ist.

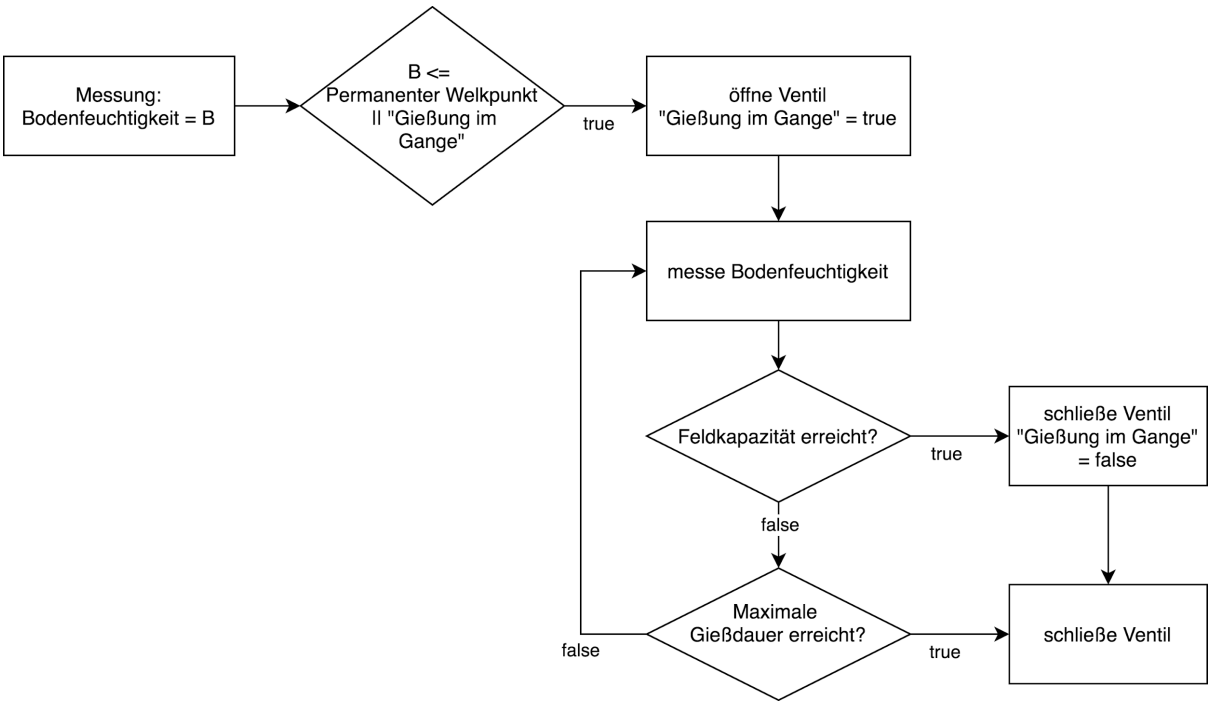


Abbildung 17: Smart Irrigation Aktivitätsdiagramm

Um Messrauschen zu glätten und nutzbare Werte zu erhalten wird der Sensor bei einer Messung mehrmals im Abstand von 2 Millisekunden ausgelesen.

3.9.1. Kapazitiver Bodenfeuchtesensor *(Integrated Sensors)*

3.9.1.1. Physikalische Grundlagen

Dieser Bodenfeuchtesensor basiert auf dem Funktionsprinzip eines Kondensators (Streufeldkondensator) bei der das Wasser bzw. die feuchte Erde die Rolle des Dielektrikums einnimmt. Die beiden Kupferplatten des Sensors, die durch Lack vor Verschleiß geschützt sind, nehmen die Funktion der Elektroden ein.

Je nach Feuchtigkeit des Bodens ändert sich nun die relative Permittivität des Dielektrikums (dielektrische Leitfähigkeit) und da Abstand und die Fläche der Elektroden, die auf dem Sensor aus zwei Kupferplatten bestehen, konstant ist, kann somit die dielektrische Leitfähigkeit (Permittivität ε) des Mediums zwischen den Elektroden gemessen werden.

Während Wasser bzw. feuchte Erde eine höhere relative Permittivität hat, liegt diese bei Luft bzw. trockener Erde eher im niedrigen Bereich (siehe *Tabelle 3*). Durch diese hohen Differenzen zwischen feuchtem und trockenem Boden bzw. Luft und Wasser, lassen sich Rückschlüsse auf die Feuchtigkeit des Bodens bestimmen.

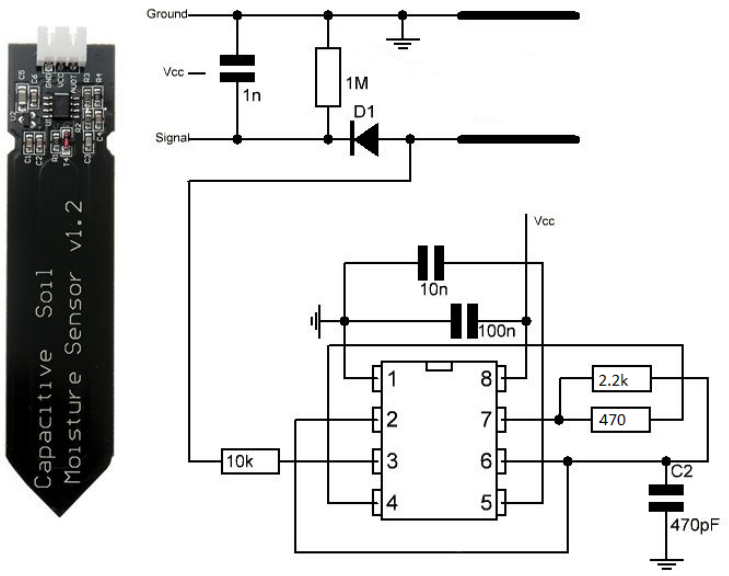
Tabelle 3: Permittivitätswerte

ε_{Vakuum}	ε_{Luft}	ε_{Wasser}	$\varepsilon_{TrockeneErde}$	$\varepsilon_{FeuchteErde}$
1	**~1,00059**	**~80**	**~3,9**	**~29**

Da auch noch andere Faktoren (wie z. B. Temperatur) Einfluss auf die relative Permittivität von Wasser haben, empfiehlt es sich, den Sensor regelmäßig zu kalibrieren. Insbesondere bei Wechsel des Mediums oder einem Ortswechsel.

Da das Ziel ist, die relative Feuchtigkeit des Bodens zu messen, sollten um den Sensor zu kalibrieren, mehrere Messungen in einem Glas Wasser und mehrere Messungen an der Umgebungsluft durchzuführen und diese dann auf die Werte von 0 bis 100 Prozent zu Mappen.

3.9.1.2. Technische Umsetzung / Sensoraufbau



Bauteile:

- NE555 Timer (DIL 8)
- 3.3V Linearregler
- 100 nF Kondensator
- 10 nF Kondensator
- 1 nF Kondensator
- 470 pF Kondensator
- 2.2 kOhm Widerstand
- 470 Ohm Widerstand
- 10 kOhm Widerstand
- 1 MOhm Widerstand
- Diode

Abbildung 18: Schaltplan des Bodenfeuchtigkeitssensors

Der Sensor wird mit 3,3 – 5 V versorgt und gibt ein analoges Signal aus. Das Signal ist pulsweitenmoduliert bei einer Frequenz von ca. 400 kHz. Durch Änderung der Feuchtigkeit ändert sich durch die Schaltung die Spannung am Ausgang. Nachfolgend eine detaillierte Beschreibung der Schaltung.

Zu Beginn wird über den VCC Eingang, 5 V Strom eingeführt. Dieser fließt erst einmal in zwei Kondensatoren (100 nF und 10 nF) unterschiedlicher Größe und Frequenz, um den Strom zu filtern. Der Strom fließt anschließend in ein 3.3V Linearregler (662K), heißt die Spannung wird auf 3.3 V gebracht und damit läuft die Schaltung stabil. Anschließend wird ein NE555 Timer über dessen VCC-Eingang (Pin acht) mit den 3.3 V versorgt, sowie dessen RST (Reset) Eingang. Zunächst werden Pins zwei und sechs (TRG und THR) des NE555 Timers direkt miteinander verbunden, sowie Pins zwei und sieben (TRG und DIS) jedoch zusätzlich mit einem 2.2 kOhm Widerstand dazwischen. Dazu wird noch Pin sieben (DIS) mit einem 470 Ohm Widerstand zum VCC Eingang (Pin acht) verbunden. Nach den direkt verbundenen Pins zwei und sechs (TRG und THR) wird ein Kondensator der Größe 470pF gegen Ground geschaltet. Der NE555 Timer mit den zwei genannten Widerständen und dem Kondensator bilden somit ein Oszillator. Dieser gibt eine Frequenz von circa 400 kHz mit Abweichungen aus, da die Bauteile eine gewisse Toleranz besitzen. Die Frequenz wird dann über den Output Pin (Pin drei) ausgegeben und zunächst in ein 10 kOhm Widerstand eingeflossen, welcher abschließend mit dem mittleren Kupferteil des Sensors verbunden ist.

Der äußere Kupferteil des Sensors ist direkt mit Ground verbunden. Zu guter Letzt, ist zusätzlich nach dem 10 kOhm Widerstand am Output (Pin drei) des NE555 Timers, eine Diode angebracht, sowie danach ein 1nF Kondensator, der parallel zu einem 1 MOhm Widerstand geschaltet ist, diese fließen dann in den AOUT Ausgang, welcher die Daten zum Mikrocontroller sendet.

3.10. Luftqualität messen

Der ursprüngliche Plan sah vor, mit dem Umgebungsluft-Sensor MQ-135 eine weitere Sensor Komponente zu implementieren. Nach näherer Betrachtung hat sich das Team jedoch dagegen entschieden, da die Auswertung der Luftqualität keinen Mehrwert für das Monitoring und die automatische Pflege der Pflanze gebracht hätte. Für Beurteilung von Behaglichkeit und Raumluftklima wäre dies ein interessanter Wert gewesen, jedoch wurde keine machbare Lösung gefunden, eine Korrelation zwischen der Pflanze und der Qualität der Umgebungsluft herzustellen. Dazu gibt es zu viele Einflüsse und Störfaktoren und eine Anzeige und Auswertung des Werteverlaufs hätten keinerlei Bezug zum Kern unseres Projektes gehabt.

3.11. Web-Applikation mit React

Wie bereits im Abschnitt Konzept erwähnt, wurde das Frontend mithilfe von React umgesetzt. Das Arbeiten mit React erfolgt über Komponentenklassen. Beim Aufbauen der Web App werden die verschiedenen Ansichten deshalb in Komponenten aufgeteilt. Diese einzelnen Komponente werden jeweils über eine eigene Klasse erstellt. Dadurch können die unterschiedlichen Komponente sehr flexibel und skalierbar eingesetzt werden.
Für die Styles wurde React-Boostrap verwendet.

3.11.1. Navigation

Für die Orientierung wurde eine Navigation Bar eingefügt um zwischen den Pages Smart Garden/Home, Overview und About navigieren zu können.

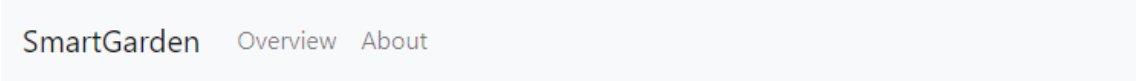


Abbildung 19: Navigationbar

3.11.2. Smart Garden Page

Auf der Smart Garden/Home Page wird eine Tabelle gezeigt, die die konfigurierten Geräte (ESPs) zeigen.

#	Device Nickname	Device ID	Plant Type	Soil Type	Mode	Permanent Wilting Point	Field Capacity	Min Temperature	Max Temperature	Min Brightness
0	Timo's Chili	cb1d463d-71c6-4e8a-a397-b393c6c6d0c1	Vegetables	Humus	Bloom	25 %	44 %	5 °C	35 °C	500 lux
1	ESP-Sebastian	6ae0ae46-4776-4c46-86a8-8ee50e342102	Vegetables	Humus	Growth	25 %	44 %	5 °C	35 °C	500 lux
2	ESP-Max	1aa91e2f-943a-488b-822e-fcc684581827	Vegetables	Humus	Bloom	25 %	44 %	5 °C	35 °C	500 lux
3	ESP-Andy	c1b1c870-30bd-416d-b4b1-d88689e35522	Vegetables	Humus	Growth	25 %	44 %	5 °C	35 °C	500 lux

Abbildung 20: Tabelle mit konfigurierten Geräten

Ist der ESP noch nicht konfiguriert, kann dies über das darunter liegende Formular gemacht werden.

Abbildung 21: Form für das Konfigurieren eines Gerätes

Beispielhaft wurden in der obigen Abbildung die Parameter bereits ausgefüllt. Bei Device wird über ein Dropdown Menü die Device ID des eigenen Gerätes ausgewählt. Bei Type kann zwischen Vegetables, Cacti und Flowers ausgewählt werden, je nachdem welcher Type hier gewählt wird, wird die Pflanze unterschiedlich bewässert.
Dasselbe gilt für die Auswahl von Mode, hier kann zwischen Growth und Bloom gewählt werden und damit wird die Beleuchtung der Pflanze beeinflusst.

3.11.3. Overview Page

Die Overview Page zeigt zunächst welches Gerät gerade ausgewählt und damit angezeigt wird.

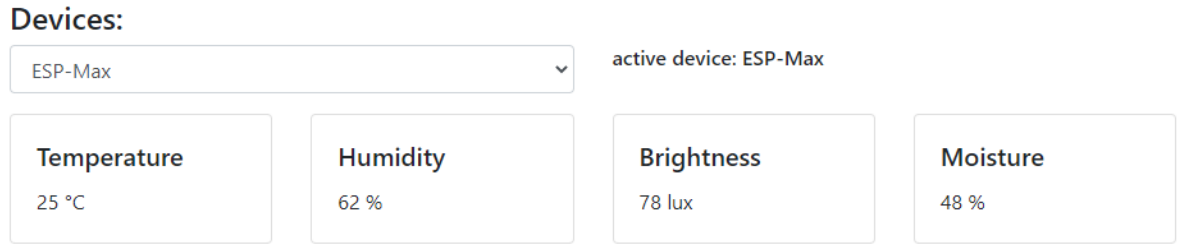


Abbildung 22: Ansicht für aktiviertes Gerät und Sensordaten

Darunter werden Card Komponenten abgebildet, die die aktuell gemessenen Sensordaten zeigen.

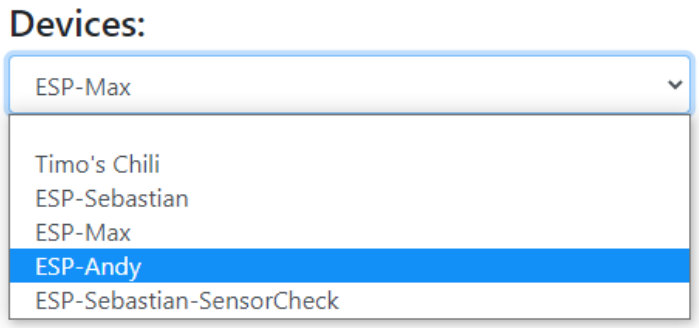


Abbildung 23: Dropdown Menü für die Auswahl des aktiven Gerätes

Über das Dropdownmenü kann zwischen den konfigurierten Geräten ausgewählt werden. Beim Ändern des aktiven Geräts aktualisiert sich die Overviewpage.

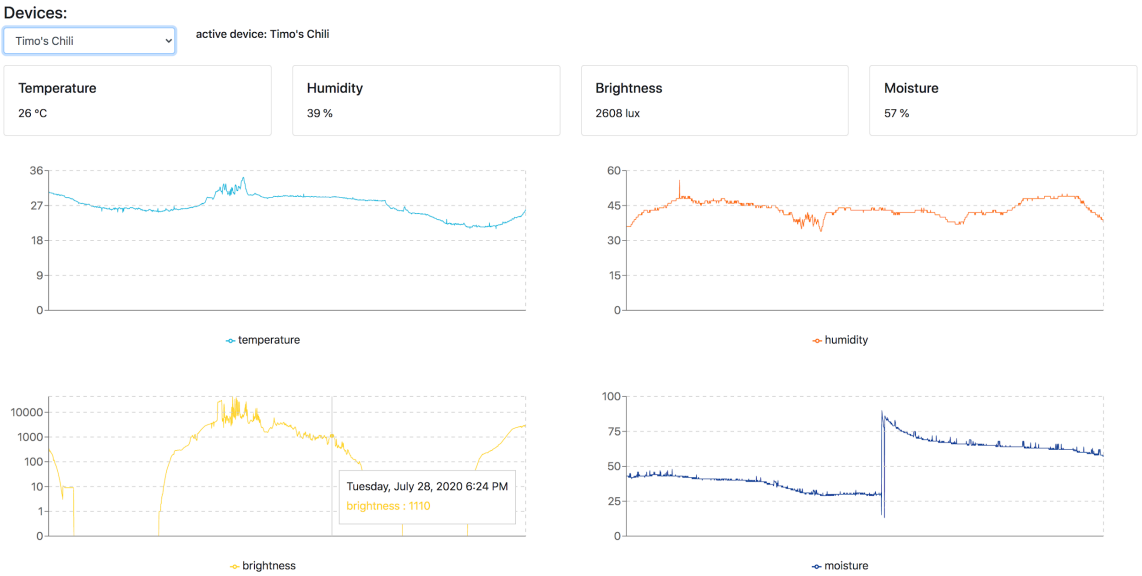


Abbildung 24: Overviewpage

Um dem Benutzenden einen langfristigen Überblick über die Sensordaten seiner Pflanze geben zu können, wurden Line Charts von recharts eingefügt.

Außerdem werden dem Benutzenden bei den Temperatur- und Feuchtigkeit-Daten Abweichungen markiert.
In der folgenden Abbildung wird zum Beispiel der Schwellenwert für die Feuchtigkeit überschritten und damit rot markiert. Der Schwellenwert ist abhängig von dem eingestellten Typ der Pflanze.

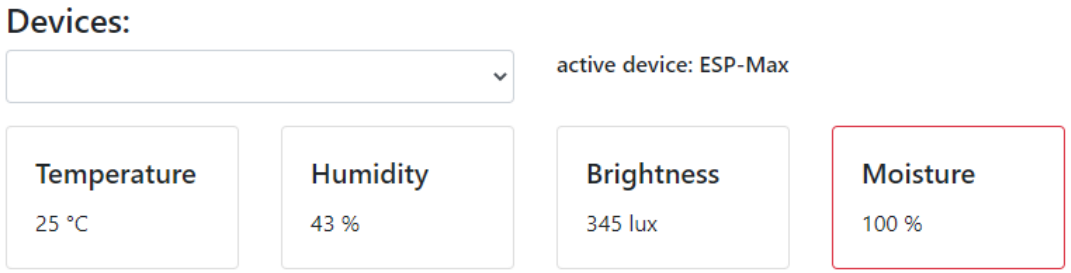


Abbildung 25: Abweichung der Sensordaten aus dem Sollbereich

3.12. Meteor Plattform

3.12.1. Was ist Meteor?

Meteor ist eine JavaScript Plattform für die Entwicklung von Web-Apps für verschiedene Endgeräte. Diese beinhaltet eine vielzahl von Node.js Paketen sowie Pakete aus der JavaScript Community.
Meteor funktioniert als Server und sendet keine HTML, sondern Daten, die vom Client gerendert werden. Die Implementierung erfolgt oft mithilfe von React im Frontend, da Meteor viele Schnittstellen bietet, mit denen das Frontend auf live Daten zugreift und bei Änderungen auch live im Frontend aktualisiert, beziehungsweise rendert.

3.12.2. Wieso Meteor?

Eines der primären Gründe wieso für die Umsetzung die Meteor Plattform verwendet wurde, ist die oben genannte Einfachheit mittels Websockets und Meteors Publish/Subscribe System, dem Client dynamische Live-Daten im Frontend bieten zu können. Dieser Grund ist essentiell für die Funktionalität des Projektes, da eine Anzeige von veralteten Daten im Frontend, einer Pflanze das Leben kosten könnte.
Der zweite ausschlaggebende Grund für die Verwendung von Meteor, ist die überaus gute Kompatibilität mit den weiteren genutzten Technologien. React für ein visuell-angenehmes Frontend und MongoDB als Datenbank für die dynamische Persistenz der gesammelten Pflanzen-Werte.
Meteor generiert zudem automatisch eine API. Dieser Punkt ist zwar nicht ausschlaggebend, jedoch erwähnenswert.

3.12.3. Wie funktioniert Meteor?

Meteor ist in zwei wesentlichen Bereichen unterteilt, die Serverseite und die Clientseite. Die Serverseite startet den Server und die Clientseite bereitet das Frontend vor. Der Client greift jedoch nicht direkt auf die Daten der Datenbank zu, sondern der Server übermittelt diese dem Client. Dies geschieht mithilfe der Meteor "Publish" Methode. Diese holt sich die jeweilige MongoDB Collection aus der Datenbank und bereitet diese in zugänglicher Form für den Client vor.
Damit der Client auf die gepublishten Daten zugreifen kann, muss es diese mithilfe der "Subscribe" Methode subscriben und im Client zwischenspeichern.

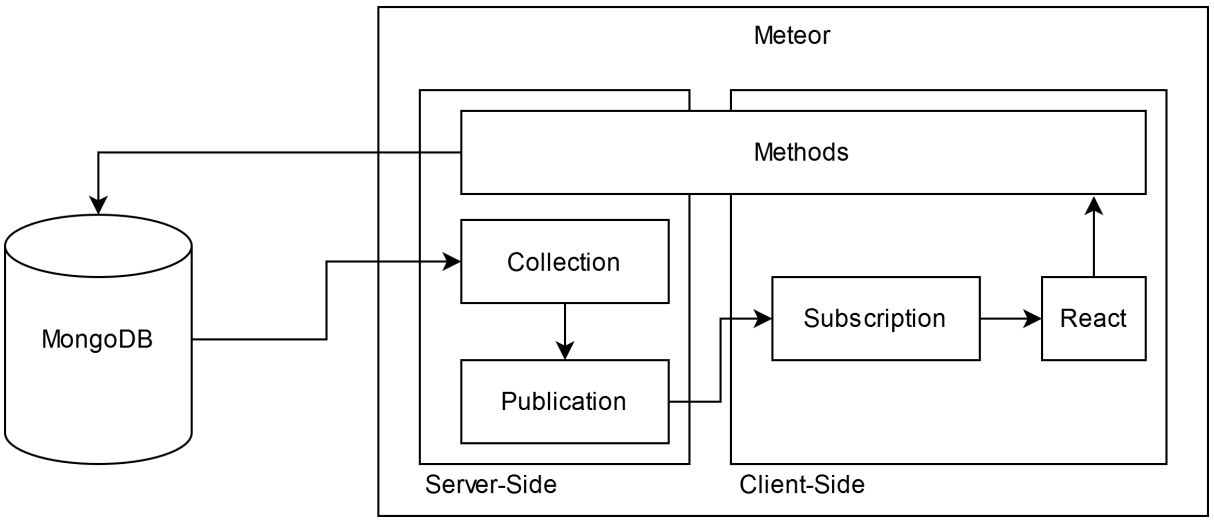


Abbildung 26: Meteor Architektur

Um Live-Daten im Frontend bereitzustellen, werden die zwischengespeicherte Collections im Client mithilfe der "Tracker" Methode getrackt. Dies bedeutet, dass sobald sich die getrackte Collection innerhalb der Datenbank verändert, das Frontend neu gerendert wird.

3.13. Mongo Datenbank

MongoDB ist eine NoSQL Datenbank, heißt, eine nicht-relationale Datenbank. Diese speichert Daten in Form von JSON-ähnlichen Objekten (BSON) und besitzt eine eigene Query-Sprache. MongoDB wurde ausgesucht, da sie einerseits die bevorzugte Datenbank bei Verwendung der Meteor Plattform ist und andererseits, da diese durch ihre Flexibilität und Skalierbarkeit für unsere Zwecke eine bevorzugte Wahl ist. Die Datenbank läuft ebenfalls als zentrale Instanz in Docker auf dem Server.

3.14. Data-Collector

Der Data-Collector ist eine kleine JavaScript-Applikation, die den einzigen Zweck hat, die Daten von den Smart Garden-Geräten vom MQTT-Broker entgegenzunehmen. Der Collector kann im Kontext von Meteor laufen aber auch als unabhängige Standalone als eigene Anwendung, um so auch mit einer hohen Anzahl an Clients (Smart Garden-Geräte und Meteor-Server) fertig zu werden (Skalierbarkeit durch Modularität).

3.15. Docker

In der Produktivumgebung bzw. dem Live-Test-Server laufen alle Module des Server-Systems in sogenannten Docker-Containern. Diese Virtualisierungsschicht ermöglicht Isolation und einfache Skalierung, erleichtert die Konfiguration und die Deployment Prozesse. Die Entwicklung kann jedoch lokal ohne diese Schicht erfolgen, wenn gewünscht.

3.16. MQTT-Broker & Docker

Der MQTT-Broker ist ein Mosquitto-Server. Dieser ist ebenfalls über ein bereits existierendes Docker-Image direkt zu Beginn der Entwicklung auf einem Server bereitgestellt worden.

4. Komplikationen

4.1. Hartnäckige Exceptions

“*Task watchdog got triggered. The following tasks did not feed the watchdog in time”*
Der Task Watchdog ist ein Mechanismus des ESP-IDFs um unbeabsichtigt lang laufende Tasks zu erkennen. Wird der Timer für einen lang laufenden Task nicht rechtzeitig durch Yielding zurückgesetzt, wird eine Panic Exception geworfen, die die Codeausführung auf dem ESP 32 stoppt. Dieses Verhalten konnte deaktivieren werden, ohne dass das Gerät im Langzeittest abnormales Verhalten zeigte, jedoch die Ursache für das Auftreten nicht identifizieren. Möglicherweise liegt diese in einer von uns benutzten Library oder des zugrunde liegenden Arduino- bzw. ESP-IDF-Frameworks.

4.2. WiFi Verbindung verursachte Reboot

Beim Verbindungsaufbau des WLANs hat sich der ESP 32 nach einem Timeout jedes zweite mal neugestartet. Leider ließ sich hier bisher keine Lösung finden die das Problem komplett behebt. Da das Problem während des Hochfahrens auftritt, konnten die Symptome des Fehlers durch ein gezwungenen Reboot beim Auftreten des Fehlers, auf ein Minimum reduziert werden. Ein offener GitHub-Issue zeugt davon, dass auch andere auf dieses Problem gestoßen sind und bisher noch keine saubere Lösung dafür existiert.

4.2. ESP-IDF vs. Arduino

Zu Beginn des Projektes war es vorgesehen die ESP-IDF Entwicklungsumgebung für die Umsetzung zu nutzen. Grund hierfür war, dass diese vom selben Hersteller (Espressif) wie die vom verwendeten Mikrocontroller (ESP 32) ist. Nach langer Recherche und ersten Implementierungen, stellte sich jedoch heraus, dass Arduino die bessere Variante für die Umsetzung des Projektes ist. Arduino ist einerseits weiter verbreitet, bietet eine bessere Dokumentation und eine einfachere Implementierung.

4.3. Lokal integrierte MongoDB in Meteor

Eines der vielen Features die Meteor bietet, ist eine integrierte Mongo Datenbank für die zu entwickelnde Web-Applikation. Für das Projekt stellte sich dies jedoch eher als ein Hindernis, anstatt eines Features. Vorgesehen war die Verwendung einer remote Mongo Datenbank, welche auf einem eigenen Server läuft. Um Meteor dazu zu bringen die remote Datenbank zu verwenden anstatt der lokalen, musste die Umgebungsvariable “MONGO_URL” verändert werden. Dies geschah vorerst im Code, doch als ersichtlich wurde, dass Meteor bei Gelegenheit trotzdem auf die lokale Datenbank zugriff, musste eine andere Lösung her.

Als Endlösung bewährte sich die Variable direkt beim Bauen der Applikation zu setzen. Dies geschieht mithilfe eines Flags im Start-Befehl. Diese Variante ist jedoch nur möglich mit einer Unix-Konsole, weshalb für den Start der Applikation die Git Bash Konsole verwendet werden muss. Somit wird die lokale Datenbank von Meteor erst gar nicht gestartet.

4.4. Breadboard ungeeignet für 5V Versorgung

Mit dem Aufbau auf dem Breadboard gab es Probleme bei der 5V Stromversorgung. Was genau das Problem war ist unklar, allerdings kam es öfters vor, dass der ESP 32 nach Anschluss des Netzteils in einer Neustart-Schleife gefangen war. Nach leichtem Umstecken auf dem Breadboard und einem Reset, funktionierte die Versorgung meist stabil, und der ESP 32 funktionierte wie vorgesehen.

4.5. Fehlermeldung bei Konfiguration von Gerät im Frontend

Für die Konfiguration eines Gerätes im Frontend werden unter anderem zwei Werte verlangt, “Device” und “Type”. Beide dieser Werte müssen aus einem Drop-Down-Menü ausgewählt werden, welche Meteor aus der remote Mongo Datenbank holt. Wenn eines dieser Eingabefelder durch den User nicht ausgefüllt wird, wird eine Fehlermeldung ausgeworfen und der User wird aufgefordert, alle Felder sorgfältig auszufüllen. Es kann jedoch vorkommen, dass trotz Ausfüllen aller Felder, die Fehlermeldung dennoch geworfen wird. Dies liegt daran, dass der Server zu lange braucht, um das ausgewählte Device oder Type aus der Datenbank zu holen und das zu setzende Attribut somit bei der Verarbeitung der Methode leer bleibt.
Dieser Fehler wurde aus zeitlichen Gründen leider nicht behoben.

5. Ausblick

5.1. Batteriebetrieb & Stromsparmodus

Aktuell wird der Mikrocontroller und das Ventil über ein Netzteil versorgt. Dies ist für draußen stehende Pflanzen eher ungünstig, weshalb eine Stromversorgung über einer Batterie optimal wäre.
Um die Langlebigkeit der Batterie zu optimieren, wäre es möglich, die Deep-Sleep-Funktion des ESP 32 zu verwenden. Dieser Stromsparmodus bringt jedoch einige Probleme mit sich. Einerseits laufen mehrere Tasks nebenläufig auf dem ESP, wird einer dieser Tasks beendet, geht der Mikrocontroller in den Stromsparmodus ohne die anderen vollendet zu haben. Diese Problematik kann mit Semaphoren gelöst werden, einem Blockiermechanismus, bereitgestellt durch FreeRTOS, welcher verhindert, dass das Gerät abgeschaltet wird, solange noch Tasks ausgeführt werden.
Ein anderes Problem ist, dass der MQTT Broker Nachrichten an den ESP verschickt, während dieser im Stromsparmodus ist und somit verloren gehen. Hierfür gibt es die MQTT-Retain Funktion im Broker. Nachrichten werden dann im Broker zwischengespeichert und zugestellt, wenn ein Client auf das passende Topic subscribed.

5.2. Frontend

Im Frontend sind auch noch einige Erweiterungen möglich. Beispielsweise werden aktuell nur die Daten der letzten 48 Stunden im Frontend angezeigt. Möchte der User aber Werte eines anderen Zeitraums sehen, könnte eine Kalender Abfrage mit Start und Ende eingefügt werden, damit der User selbst entscheiden kann, aus welchem Zeitraum die Daten angezeigt werden sollen. Somit passt sich anhand des übergebenen Zeitraums, die Sensordaten query im code an.
Eine andere mögliche Erweiterung, ist das manuelle Gießen oder Beleuchten der Pflanze über das Frontend. Dies kann mithilfe von Buttons im Frontend getriggert werden, indem beim Klicken jeweils ein payload an den MQTT Broker geschickt wird, der das Gießen oder Beleuchten auslöst.

5.3 Mehrbenutzersystem

Um unabhängigen Benutzern ein Verwenden unserer Applikation in ihrem eigenen abgeschlossenen Kontext zu ermöglichen, sollte Authentifizierung und Autorisierung implementiert werden. Seine eigenen Devices lassen sich dann z. B. durch Scan eines QR-Codes (auf dem Gerät angebracht) mit der Device-ID seinem Benutzerkonto zugewiesen werden.

5.4. Caching und verzögertes Senden von Messdaten

Um zu verhindern, dass Messdaten verloren gehen, wenn der ESP 32 nicht mit dem Server verbunden ist, könnte man einen Cache im ESP 32 implementieren, um diese zwischenzuspeichern und zu einem späteren Zeitpunkt, wenn wieder eine Verbindung besteht, wieder an den Server auszuspielen. Beim Verbindungsaufbau muss dann überprüft werden ob Daten im Cache liegen. Weiter muss die Zeiterfassung der Messung, die bisher vom Server gehandhabt wird, auf dem ESP 32 stattfinden, damit die Messdaten nicht erst zum Zeitpunkt der Zustellung mit Zeitstempel versehen werden.

5.4. Aufbau

Ein Ausblick für die Zukunft war es, die Schaltung fest zu verlöten und ein Gehäuse für die Elektronik zu bauen. Mittlerweile wurde das umgesetzt.
Die Elektronik befindet sich auf einer 10x10cm großen Lochplatine. Für Sensoren, Ventil und Stromversorgung gibt es Steckverbindungen auf der Lochplatine.
Das Gehäuse wurde 3-D gedruckt. Der BH1750 und der DHT11 Sensor, so wie die Stromversorgung über einen Klinkenstecker sind im Gehäuse integriert. Für die Kabel des Bodenfeuchtigkeitssensors und des Ventils gibt es im Gehäuse entsprechende Löcher. Außerdem ist der ESP 32 von außen zugänglich, damit neue Software geflasht werden kann und auch die Knöpfe Boot und Reset erreichbar sind.

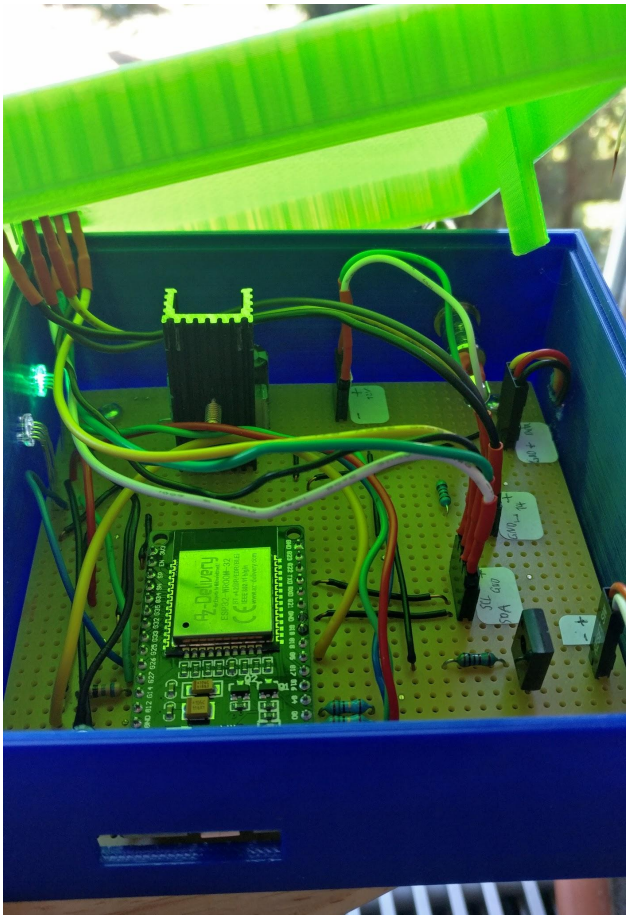


Abbildung 27: Gehäuse

Aktuell wird der Topf über einen Schlauch punktuell bewässert. Dadurch wird das Wasser nicht gut verteilt, und der Bodenfeuchtigkeitssensor kann je nachdem, wo dieser steckt unpräzise Messwerte liefern. Wenn alles richtig aufbaut ist, ist das zwar kein Problem aber dennoch nicht schön.
Als Lösung dafür könnten mehrere Löcher in den Schlauch gestochen werden, um so das Wasser in verschiedene Richtungen zu verteilen. Alternativ kann auch mit einer Pumpe gearbeitet werden. Das erfordert jedoch eine Überarbeitung der Schaltung.

5.5 Beleuchtung

Die Beleuchtung könnte noch an den realen Tag/Nacht Zyklus angepasst werden. Dabei könnte man die Helligkeitskurve über den Tag nachahmen und die Lichtfarbe über den Tagesverlauf anpassen bzw. Sonnenaufgang und -untergang simulieren. Zusätzlich könnte man die Beleuchtung auch nach Region und Jahreszeit anpassen um gewisse Bedingungen simulieren damit sich die Pflanzen heimisch fühlen.

5.6. Notifications

Aktuell werden Werte von (Temperatur, Luftfeuchtigkeit, Bodenfeuchtigkeit, Licht) auf der Weboberfläche rot dargestellt, wenn diese kritische Werte erreichen (zu hoch oder zu niedrig). Die Grenzwerte dafür sind zusammen mit dem Pflanzentyp in der Datenbank hinterlegt.
Eine mögliche Erweiterung wäre es nun den User über diese Grenzwertüberschreitungen in Form von SMS, E-Mail oder Telegram zu informieren.

Sonstiges

Abbildung 28: Aufbau Timo

Abbildung 29: Aufbau Sebastian

Abbildung 30: Aufbau Max

Abbildung 31: Aufbau Andy

Quellenangaben

Bilder

Abbildung 3: AZ-Delivery ESP32 Mikrocontroller -
https://cdn.shopify.com/s/files/1/1509/1638/products/Esp32DevkitCNewChipfront_500x.jpg?v=1583915972

Abbildung 4: DHT11 Sensor -
https://images-na.ssl-images-amazon.com/images/I/51-fjA52JRL._AC_SX425_.jpg

Abbildung 5: BH1750 Sensor -
https://raw.githubusercontent.com/Erriez/ErriezBH1750/master/extras/BH1750.png

Abbildung 6: Capacitive Soil Moisture Sensor v1.2 -
https://cdn.shopify.com/s/files/1/1509/1638/products/bodenfeuchtesensor-hygrometer-modul-v12-kapazitiv-fur-arduinosensoraz-deliveryaz-delivery-23418940_500x.jpg?v=1572809240

Abbildung 7: Ventil -
https://www.funduinoshop.com/WebRoot/Store14/Shops/78096195/5A4F/3B4D/12EA/D0FE/758F/0A0C/6D09/65A4/Ventil6mm.jpg

Abbildung 8: RGB-LED -
https://media.digikey.com/Photos/Kingbright%20Photos/WP154A4SUREQBFZGC.JPG

Abbildung 17: Schaltplan des Bodenfeuchtigkeitssensors -
https://www.instructables.com/id/Soil-Moisture-Sensor-Calibration/

Alle anderen Abbildungen wurden selbst erstellt.

Links, Querverweise im Text

Pull-up
https://www.elprocus.com/pull-up-and-pull-down-resistors-with-applications/&sa=D&ust=1596120400732000&usg=AOvVaw3FCxDjU63mqRMlBq5VpudZ

FreeRTOS
https://www.freertos.org/index.html&sa=D&ust=1596120400734000&usg=AOvVaw2V-wH7zEwpgkL6QNSzbX30

BH1750
https://platformio.org/lib/show/439/BH1750&sa=D&ust=1596120400735000&usg=AOvVaw0X9SWv7cHEqZ0OL6-BLqI1

DHT sensor library
https://platformio.org/lib/show/19/DHT%2520sensor%2520library&sa=D&ust=1596120400735000&usg=AOvVaw2MKdFwjBxY_i7IWtVYO4Mn

AutoConnect@^1.1.7
https://platformio.org/lib/show/2678/AutoConnect&sa=D&ust=1596120400736000&usg=AOvVaw1AQ9aZcDg2o0uBKWb35V2G

ArduinoJson@^6.15.2
https://platformio.org/lib/show/64/ArduinoJson&sa=D&ust=1596120400736000&usg=AOvVaw2CzjA319HJQMMT3-Xw39RE

PubSubClient@^2.8
https://platformio.org/lib/show/89/PubSubClient&sa=D&ust=1596120400736000&usg=AOvVaw25_YiqHFNOfL5NiVbO2saQ

ArduinoNvs@^2.5
https://platformio.org/lib/show/5989/ArduinoNvs&sa=D&ust=1596120400737000&usg=AOvVaw0--67fwJw26YMUXoYUrl_8

ESPRandom@^1.3.3
https://platformio.org/lib/show/6883/ESPRandom&sa=D&ust=1596120400737000&usg=AOvVaw1zvekJQ0zCskRQZuv8ry3l

WPS-Funktion
https://de.wikipedia.org/wiki/Wi-Fi_Protected_Setup&sa=D&ust=1596120400738000&usg=AOvVaw0JXK-e5J-ZeeAVSn2sxAdh

Captive-Portal
https://de.wikipedia.org/wiki/Captive_Portal&sa=D&ust=1596120400738000&usg=AOvVaw2-g7pMVMZ5TEVBL-wk1Aie

AutoConnect
https://hieromon.github.io/AutoConnect/index.html&sa=D&ust=1596120400740000&usg=AOvVaw1IwThnXEqS5X-XeDT-LXXt

FreeROTS Tasks
https://www.freertos.org/taskandcr.html&sa=D&ust=1596120400741000&usg=AOvVaw3LHDBbLlFlFQ2bZ3SFDfph

FreeRTOS Timer
https://www.freertos.org/RTOS-software-timer.html&sa=D&ust=1596120400741000&usg=AOvVaw1R8q0QLyFP9JapfcrmhWGl

UUID Version 4 nach RFC4122
https://de.wikipedia.org/wiki/Universally_Unique_Identifier&sa=D&ust=1596120400749000&usg=AOvVaw0KetTmeqP5lg6u7vMJIlVA

ESPRandom Library
https://github.com/protohaus/ESPRandom&sa=D&ust=1596120400749000&usg=AOvVaw2msLz71UuIIn4gD1SDxLg9

NVS
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/storage/nvs_flash.html&sa=D&ust=1596120400750000&usg=AOvVaw0w2lCNVSVezWD6iVGmWdQw

ArduinoNVS
https://github.com/rpolitex/ArduinoNvs&sa=D&ust=1596120400750000&usg=AOvVaw2fG1BecESLJPF01BOf0w4i

BH1750
https://components101.com/sensors/bh1750-ambient-light-sensor&sa=D&ust=1596120400751000&usg=AOvVaw2U8UND69FL6l5LTWHdtxJl

Permanenten Welkepunkt
https://de.wikipedia.org/wiki/Welkepunkt&sa=D&ust=1596120400752000&usg=AOvVaw3gTInv3iTS7p8s2tozpr7D

Feldkapazität
https://de.wikipedia.org/wiki/Feldkapazit%25C3%25A4t&sa=D&ust=1596120400753000&usg=AOvVaw2vnwhAPCnc2A2Iqu9CGcds

https://www.dwd.de/DE/fachnutzer/landwirtschaft/dokumentationen/agrowetter/Schlagkonfiguration.pdf?__blob%3DpublicationFile%26v%3D2&sa=D&ust=1596120400754000&usg=AOvVaw0AwMfFigHJhStE4fODPOu_

https://www.bodenkunde-projekte.hu-berlin.de/boku_online/pcboku10.agrar.hu-berlin.de/cocoon/boku/sco_6_wasserhaushalt_127cf8.html?section%3DN100CL&sa=D&ust=1596120400754000&usg=AOvVaw2dxbpF7nBLs7zwVFIeE5Bt

Permittivität
https://de.wikipedia.org/wiki/Permittivit%25C3%25A4t&sa=D&ust=1596120400756000&usg=AOvVaw1W3b-PkxCa5_H2cuLS1lPk

React
https://reactjs.org/docs/getting-started.html&sa=D&ust=1596120400761000&usg=AOvVaw35I33nfksemF0kNyj-WdOu

React-Boostrap
https://react-bootstrap.github.io/components/table/&sa=D&ust=1596120400762000&usg=AOvVaw3-cCKUvUu2JBiXkV6QDjaq

recharts
http://recharts.org/en-US/&sa=D&ust=1596120400765000&usg=AOvVaw0ivaBS_a9zIakzqXtZi5DF

Meteor
https://docs.meteor.com/&sa=D&ust=1596120400766000&usg=AOvVaw1kxbjHT_19dA7dn-WHGNhy

Publish
https://docs.meteor.com/api/pubsub.html%23Meteor-publish&sa=D&ust=1596120400768000&usg=AOvVaw0oNU-UNC6TGBZy3lnBA9MZ

Subscribe
https://docs.meteor.com/api/pubsub.html%23Meteor-subscribe&sa=D&ust=1596120400768000&usg=AOvVaw3TpxS5PXIJq6OHnigxwS06

Tracker
https://docs.meteor.com/api/tracker.html&sa=D&ust=1596120400769000&usg=AOvVaw0tURUEfwH0P_v6_GrFPj2v

MongoDB
https://www.mongodb.com/what-is-mongodb&sa=D&ust=1596120400769000&usg=AOvVaw20WslQEfxLAurmxj1vP52h

BSON
http://bsonspec.org/&sa=D&ust=1596120400770000&usg=AOvVaw0vJe8KaRWcKw2LvPj5bg5U

Docker-Containern
https://www.dev-insider.de/was-sind-docker-container-a-597762/&sa=D&ust=1596120400771000&usg=AOvVaw3fKbqSmX57pAIwPdSxmA7b

Mosquitto-Server
https://mosquitto.org/&sa=D&ust=1596120400771000&usg=AOvVaw1fJ1IJqifRefmMbqMnihes

Docker-Image
https://hub.docker.com/_/eclipse-mosquitto&sa=D&ust=1596120400772000&usg=AOvVaw3kGxKwTnLIzZNjXkZelad6

Task Watchdog
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/wdts.html&sa=D&ust=1596120400773000&usg=AOvVaw3hi07JfUqFF5UXpGJCuIHD

GitHub-Issue
https://github.com/espressif/arduino-esp32/issues/2501&sa=D&ust=1596120400774000&usg=AOvVaw0MvoSqn1QU7534WkMtdGD0

Deep-Sleep-Funktion
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html&sa=D&ust=1596120400777000&usg=AOvVaw2dsDex2uJefNonuT8Nxsur