

TERRA TAP

Optimierung der Wassernutzung



Fokko Vos
SOL 2 Dokumentation

1 Technical Summary

Projektname: TerraTap

Das TerraTap-Projekt wurde ins Leben gerufen, um die Wassernutzung in urbanen Ökosystemen zu optimieren. Ziel ist es, ein intelligentes Bewässerungssystem zu entwickeln, das den Wasserbedarf von Pflanzen effizient und bedarfsgerecht steuert.

Hauptvorteile:

- **Wassersparnis:**
Reduzierung des Wasserverbrauchs durch präzise und bedarfsgerechte Bewässerung.
- **Nachhaltigkeit:**
Förderung der nachhaltigen Pflege städtischer Grünflächen.
- **Ökonomische Effizienz:**
Minimierung der Kosten durch optimierten Ressourceneinsatz.

Wichtige Ergebnisse:

- **Präzise Messung der Bodenfeuchtigkeit:**
Die Sensoren haben zuverlässig die Feuchtigkeit gemessen, um eine bedarfsgerechte Bewässerung zu ermöglichen.
- **Effiziente Steuerung:**
Die Bewässerungssysteme wurden so programmiert, dass sie den Wasserbedarf basierend auf den Sensordaten steuern und dadurch Über- und Unterbewässerung vermeiden

Zeitplan:

Das Projekt wurde innerhalb des geplanten Zeitrahmens von fünf Wochen abgeschlossen. Wichtige Meilensteine wurden termingerecht erreicht, einschliesslich der Entwicklung und Testphase des Prototyps.

Budget:

Die Gesamtkosten des Projekts beliefen sich auf 90 CHF für Sachmittel, was innerhalb des geplanten Budgets lag.

Risiken und Herausforderungen:

- **Systemintegration:**
Die Verbindung und Kommunikation zwischen verschiedenen Hardware- und Softwarekomponenten stellte eine Herausforderung dar.
- **Technische Probleme:**
Probleme bei der Einrichtung und Programmierung der Mikrocontroller wurden durch intensive Tests und Debugging gelöst.

Empfehlungen:

- **Weiterentwicklung und Skalierung:** Das System sollte für grössere Grünflächen und unterschiedliche Einsatzszenarien weiterentwickelt werden.
- **Sicherheitsmassnahmen:** Implementierung zusätzlicher Sicherheitsmassnahmen, wie die Authentifizierung von MQTT-Nachrichten.
- **Langzeittests:** Durchführung weiterer Langzeittests zur Validierung der Systemstabilität und Effizienz.

Inhaltsverzeichnis

1	TECHNICAL SUMMARY	1
2	VERSIONIERUNG	5
3	AUFTRAG.....	6
3.1	AUSGANGSLAGE	6
3.2	AUFTRAGSBESCHREIBUNG	6
3.3	ZIELE	6
3.4	ABGRENZUNG	6
3.5	LIEFEROBJEKTE	6
3.6	PROJEKTORGANISATION	6
3.7	MEILENSTEINE.....	7
3.8	HILFSMITTEL UND KOSTEN.....	7
3.9	VORKENNTNISSE	7
3.10	NEUE LERNINHALTE	7
4	ZEITPLANUNG.....	8
5	ARBEITSRAPPORTE	9
5.1	Woche 1: 13.05 – 19.05.....	9
5.2	Woche 2: 20.05 – 26.05.....	11
5.3	Woche 3: 27.05 – 02.06.....	13
5.4	Woche 4: 03.05 – 09.06.....	15
5.5	Woche 5: 10.06 – 14.06.....	16
6	PROTOKOLLE	17
6.1	KICKOFF-MEETING.....	17
6.2	1. MEETING	17
6.3	2. MEETING	18
6.3.1	Teil 1.....	18
6.3.2	Teil 2.....	19
7	SYSTEMDOKUMENTATION.....	20
7.1	SYSTEMARCHITEKTUR	20
7.1.1	Sensor.....	20
7.1.2	Wassersteuerung	20
7.1.3	Hub.....	20
7.2	SYSTEMANFORDERUNGEN	21
7.2.1	Funktionale Anforderungen.....	21
7.2.2	Nicht-funktionale Anforderungen.....	21
7.3	SICHERHEITSASPEKTE	21
7.4	ENTWICKLUNGSUMGEBUNG	22
7.4.1	Einrichtung Arduino.....	22
7.4.2	Einrichtung Visual Studio Code.....	23
7.4.3	Weitere Entwicklungswerkzeuge	23
7.5	KONZEPTION SOFTWARE	24
7.5.1	Steuerungssystem.....	24
7.5.2	Hub.....	25
7.6	KONZEPTION HARDWARE.....	26
7.6.1	Sensor Schaltkreis.....	26

7.6.2	Wassersteuerung Schaltkreis	26
7.7	KOMMUNIKATION	27
7.8	HARDWARE KOMPONENTEN	28
7.8.1	Sensor.....	28
7.8.2	Wassersteuerung	28
7.8.3	HUB	28
8	BAU	29
8.1	SENSOR	29
8.2	WASSERSTEUERUNG	30
9	PROGRAMMCODE: MODULE (CLIENTS).....	31
9.1	LIB.....	32
9.1.1	clients/lib/SimpleConnect/SimpleConnect.cpp.....	32
9.1.2	clients/lib/SimpleConnect/SimpleConnect.h.....	33
9.1.3	clients/lib/SimpleDebug/SimpleDebug.h.....	34
9.1.4	clients/lib/SimpleSleep/SimpleSleep.cpp	35
9.1.5	clients/lib/SimpleSleep/SimpleSleep.h.....	36
9.2	SENSOR	37
9.2.1	clients/sensor/sensor.ino	37
9.3	WATERING	41
9.3.1	clients/watering/watering.ino.....	41
10	PROGRAMMCODE: HUB	45
10.1	HUB.....	45
10.1.1	src/main.rs	46
10.1.2	src/mqttc.rs	49
10.1.3	src/settings.rs.....	51
10.1.4	src/state.rs	52
10.1.5	src/core/serde.rs.....	53
10.1.6	src/core/manager.rs.....	55
10.1.7	src/core/macros.rs	59
10.1.8	src/core/traits/config.rs	59
10.1.9	src/core/traits/module.rs	61
10.1.10	src/modules/prelude.rs.....	63
10.1.11	src/modules/sensor.rs.....	64
10.1.12	src/modules/watering.rs.....	66
10.2	MQTTD	68
10.2.1	src/main	68
10.2.2	rumqtttd.toml	69
11	SCREENSHOTS	71
12	TESTKONZEPT.....	73
12.1	TESTSTRATEGIE	73
12.1.1	Testclients.....	73
12.2	TEST-CASES.....	74
12.3	UNIT-TESTS FÜR DEN HUB.....	77
12.4	E2E TESTING.....	77
13	TESTS.....	78

13.1	UNIT-TEST RESULTATE	78
13.2	E2E RESULTATE.....	79
14	REFLEKTION	80
15	ABSCHLUSS UND AUSBLICK.....	81
16	VERWEISE	82
16.1	TABELLENVERZEICHNIS	82
16.2	ABBILDUNGSVERZEICHNIS	83
17	QUELLENVERZEICHNIS	83
18	GLOSSAR	84
19	ANHANG.....	87
19.1	ARDUINO BIBLIOTHEKVERSIONEN	87
19.2	RUST PAKETVERSIONEN	87
19.3	RUST EXTENSION VERSIONEN	87
19.4	KOSTEN	88

2 Versionierung

Version	Autor	Datum	Änderung
1.0	Fokko Vos	17.05.2024	Erstellung und Grundlegendes Layout
1.1	Fokko Vos	17.05.2024	Arbeitsrapport für Recherche und Tools führen
1.2	Fokko Vos	18.05.2024	Schreiben des Projektauftrages, Grundlegende Dokumentationsinhalte
1.3	Fokko Vos	18.05.2024	Nachführen der Protokolle und Arbeitsrapporte
1.4	Fokko Vos	19.05.2024	Anforderungen Dokumentiert
1.5	Fokko Vos	22.05.2024	Dokumentation der Hardware & Planung sowie das Konzept für die Software und allgemeine Architektur.
1.6	Fokko Vos	23.05.2024	Nachbesserung der Softwarespezifizierung
1.7	Fokko Vos	23.05.2024	Zeitplan EXCEL eingebunden
1.8	Fokko Vos	25.05.2024	Grafische Aufarbeitung der Systemarchitektur
1.9	Fokko Vos	26.05.2024	Dokumentation des 3D Druck Prototyps
2.0	Fokko Vos	27.05.2024	Dokumentation des Testkonzeptes
2.1	Fokko Vos	28.05.2024	Entscheidungsfindung für Hard- und Software genauer festhalten und Entscheidung für das Kommunikationsprotokoll Dokumentieren.
2.2	Fokko Vos	03.06.2024	Dokumentation für 3D Druck Modelle erweitert, Model Aktualisiert
2.3	Fokko Vos	04.06.2024	Dokumentation der Software für die Steuerung
2.4	Fokko Vos	05.06.2024	Dokumentation des HUB Programms
2.5	Fokko Vos	06.06.2024	Aktualisierung der Hardware Dokumentation und Allgemeiner Feinschliff
2.6	Fokko Vos	07.06.2024	Technische Dokumentation
2.7	Fokko Vos	08.06.2024	Finalisierung der Technischen Dokumentation
2.8	Fokko Vos	10.06.2024	Dokumentation der Testergebnisse
2.9	Fokko Vos	12.06.2024	Entwurf Reflektion & Fazit
3.0	Fokko Vos	13.06.2024	Ausarbeitung Fazit & Reflektion
3.1	Fokko Vos	13.06.2024	Finalisierung der Dokumentation
3.2	Fokko Vos	13.06.2024	Schreiben des Management Summary

Arbeitsrapport Änderungen wurden meistens zwischendurch getätigt und nicht genau dokumentiert.

1 - Versionierung

3 Auftrag

3.1 Ausgangslage

Das TerraTap-Projekt entstand aus der dringenden Notwendigkeit, die Wassernutzung in urbanen Ökosystemen zu optimieren. Aktuelle Bewässerungssysteme für städtische Grünflächen, Parks und private Gärten arbeiten oft ineffizient, da sie vordefinierte Zeitpläne nutzen, die die tatsächliche Bodenfeuchtigkeit und den spezifischen Wasserbedarf der Pflanzen nicht berücksichtigen. Dies führt sowohl zu Überbewässerung, die Wasser verschwendet, als auch zu Unterbewässerung, die Pflanzen schädigen kann. Angesichts globaler Trends zunehmender Trockenheit und Ressourcenknappheit ist eine fortschrittliche und umweltbewusste Lösung erforderlich, die sowohl ökonomische als auch ökologische Aspekte des Wasserverbrauchs berücksichtigt.

3.2 Auftragsbeschreibung

Das TerraTap-Projekt zielt darauf ab, ein intelligentes Bewässerungssystem zu entwickeln, das Bodenfeuchtigkeitssensoren und adaptive Steuerungstechnologie verwendet, um eine präzise und bedarfsgerechte Wasserversorgung zu gewährleisten. Das System wird den Wasserverbrauch reduzieren, Ressourcen effizienter nutzen und die Pflege städtischer Grünflächen nachhaltiger gestalten.

3.3 Ziele

Primäres Ziel: Entwicklung und Implementierung eines intelligenten Bewässerungssystems im Rahmen eines Prototyps, das durch den Einsatz von Bodenfeuchtigkeitssensoren eine präzise und bedarfsgerechte Wasserversorgung ermöglicht.

Sekundäre Ziele: Reduzierung des Wasserverbrauchs, Verbesserung der Ressourceneffizienz und Förderung der nachhaltigen Pflege städtischer Grünflächen.

3.4 Abgrenzung

Das Projekt konzentriert sich auf kleinere bis mittelgrosse urbane und suburbane Grünflächen. Grossflächige landwirtschaftliche Felder und Indoor-Anwendungen wie Gewächshäuser sind ausgeschlossen.

3.5 Lieferobjekte

- Ein funktionsfähiger Prototyp des TerraTap-Bewässerungssystems.
- Dokumentation des Entwicklungsprozesses und der Testergebnisse.

3.6 Projektorganisation

Das Projekt wird im Rahmen der Schule von Fokko Vos allein durchgeführt. Verantwortlich für die Gesamtkoordination des Projekts, die Konzeption, Programmierung, Hardware- und Softwareentwicklung, den Bau, die Tests und die Dokumentation.

3.7 Meilensteine

Die Meilensteine dieses Projektes definieren sich durch den Abschluss einer Phase des Projekts Plans.

Meilenstein 1: Recherche abgeschlossen und Datenbasis für das Projekt etabliert

Meilenstein 2: Detaillierter Projektplan mit definierten Zielen und Zeitrahmen erarbeitet.

Meilenstein 3: Auswahl der Konzeption und Technologie für TerraTap.

Meilenstein 4: Prototyp von TerraTap entwickelt und bereit für erste Testphase.

Meilenstein 5: Durchführung und Analyse der Tests bestätigt, Anpassungen vorgenommen.

Meilenstein 6: Projektergebnisse und Lernerfahrungen ausgewertet und dokumentiert.

3.8 Hilfsmittel und Kosten

Das Projekt wird innerhalb von 5 Wochen durchgeführt und umfasst etwa 48.5 Stunden. Die Gesamtkosten belaufen sich auf 90 CHF für Sachmittel. Eine detaillierte Kostenaufstellung ist im Anhang zu finden.

3.9 Vorkenntnisse

Die Vorkenntnisse von Fokko Vos belaufen sich hauptsächlich auf die Programmierung. In einem früheren Modul wurden bereits ESP32-Chips programmiert und in einem minimalen Umfang zusammengebaut.

3.10 Neue Lerninhalte

Während des Projekts wird Fokko Vos vertiefte Kenntnisse über den Zusammenbau und die Integration von Hardwarekomponenten wie ESP8266-Mikrocontrollern, Bodenfeuchtigkeitssensoren und pneumatischen Ventilen erlangen. Das Verständnis der gesamten Systemarchitektur wird erweitert, einschliesslich der Kommunikation zwischen den verschiedenen Hardware- und Softwarekomponenten.

Ein zentrales neues Lernfeld ist die Verwendung von Kommunikationsprotokollen zur zuverlässigen und skalierbaren Kommunikation zwischen den Systemkomponenten. Ausserdem wird die Fähigkeit zur Modellierung und Herstellung von Bauteilen mit 3D-Drucktechniken vertieft. Dies umfasst das Design von Halterungen und Gehäusen für Sensoren und Elektronik. Die Entwicklung in Rust und die Nutzung der Arduino IDE für die Programmierung der Mikrocontroller sind weitere wichtige Lerninhalte.

4 Zeitplanung

Terra Tap

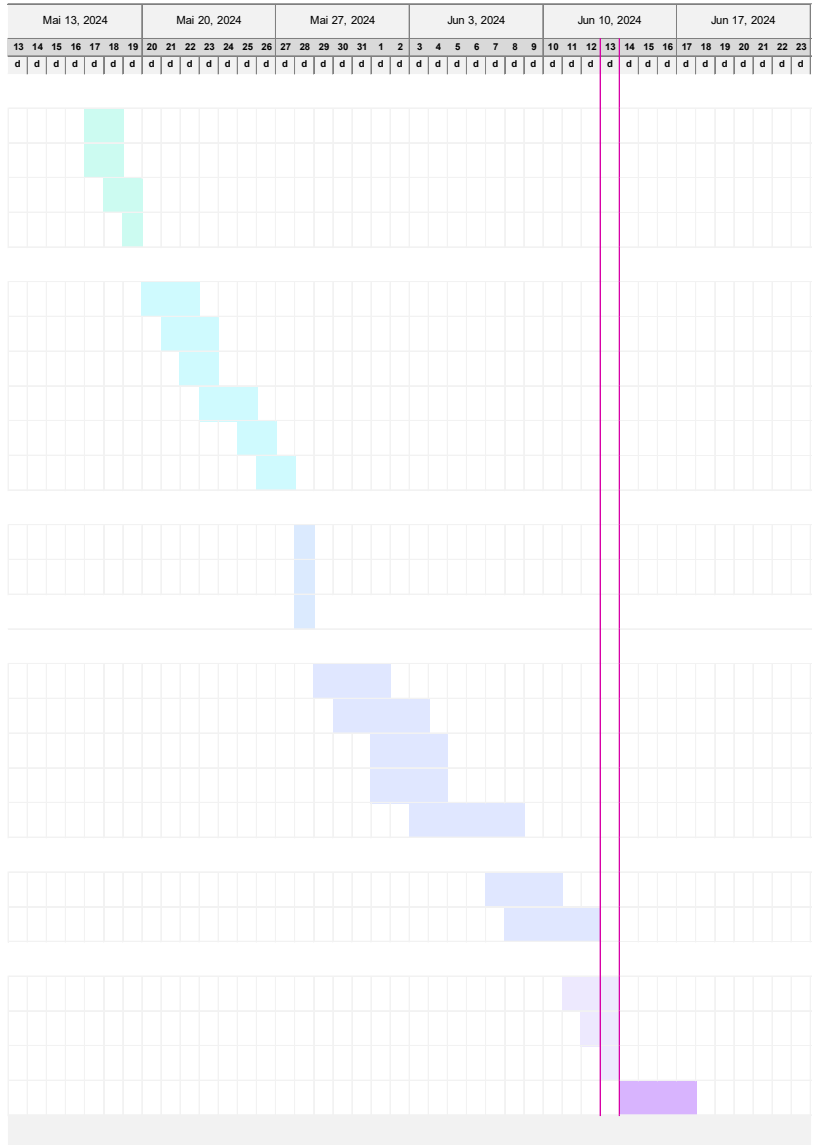
Prototyp

Projektleiter: Fokko Vos

Project start: **Freitag, 17. Mai 2024**

Display week: **1**

TASK	SOLL	IST	PROGRESS	START	END
Informieren	6	5.5	100%	17.05.24	19.05.24
Recherche	2	2	100%	17.05.24	18.05.24
Tools finden	1	1	100%	17.05.24	18.05.24
Erstellung der Dokumentation	1	1	100%	18.05.24	19.05.24
Anforderungsanalyse	2	1.5	100%	19.05.24	19.05.24
Planen	10.5	10.5	100%	20.05.24	27.05.24
Spezifikationen Hardware	3	3	100%	20.05.24	22.05.24
Spezifikationen Software	1	2	100%	21.05.24	23.05.24
Detaillierten Zeitplan erstellen	1.5	2	100%	22.05.24	23.05.24
Ausarbeitung der Systemarchitektur	1	0.5	100%	23.05.24	25.05.24
Initial Entwurf von 3D Druckmuster	2	1	100%	25.05.24	26.05.24
Testkonzept	2	2	100%	26.05.24	27.05.24
Entscheiden	1.5	1.5	100%	28.05.24	28.05.24
Entscheidung Hardware	1	1	100%	28.05.24	28.05.24
Entscheidung Software	0.5	0.5	100%	28.05.24	28.05.24
Entscheidung Kommunikationsprotokoll	0.5	0.5	100%	28.05.24	28.05.24
Realisieren	18	18.5	100%	29.05.24	08.06.24
Zusammenbau der Hardware	3	2	100%	29.05.24	01.06.24
Entwicklung der Steuerungssoftware	3	2	100%	30.05.24	03.06.24
Weiterentwicklung der 3D Druck Modelle	2	1	100%	01.06.24	04.06.24
Entwicklung des Hub's	4	7	100%	01.06.24	04.06.24
Technische Dokumentation	6	6.5	100%	03.06.24	08.06.24
Kontrollieren	5	3.5	100%	07.06.24	12.06.24
Durchführung von Integrations- und Funktionstests	3	3	100%	07.06.24	10.06.24
ggf. Anpassungen Vornehmen	2	0.5	100%	08.06.24	12.06.24
Auswerten	7	7	75%	11.06.24	17.06.24
Abgleich mit den Zielen	1	1.5	100%	11.06.24	13.06.24
Reflektion und Fazit	1	1.5	100%	12.06.24	13.06.24
Finalisierung der Dokumentation	2	4	100%	13.06.24	13.06.24
Vorbereitung der Präsentation	3	0	0%	14.06.24	17.06.24
	48	46.5		5.17.24	6.17.24



2 – Zeitplanung

Während des gesamten Projektes wird nebenbei die Dokumentation geführt.
Jede der Aufgabe beinhaltet immer auch die Dokumentation.

5 Arbeitsrapporte

5.1 Woche 1: 13.05 – 19.05

Aktivität	Aufwand geplant (h)	Aufwand effektiv (h)
Informieren, Beschaffen der Benötigten Informationen & Abgleich	6	5.5
Wochenablauf		
<ul style="list-style-type: none"> - Grundlegende Informationen über Automatik und Strom sammeln - Tools für die Simulation von Technik suchen. - Tools für 3D Druck Modellierung suchen. - Tools für die Entwicklung suchen. - Dokumentation Erstellen, Grundlayout einrichten - Anforderungen erneut sauber durchlesen 		
Hilfestellung		
<ul style="list-style-type: none"> - Unterlagen für SOL - GPT - Internet 		
Reflektion		
<p>In der vergangenen Woche habe ich mich intensiv mit verschiedenen Aspekten des SOL-Projekts auseinandergesetzt. Der Schwerpunkt meiner Arbeit lag auf dem Sammeln grundlegender Informationen über Strom. Ich habe mich zuerst grundlegend über Strom informiert, dann über Relays, Downstepper-Module und die Chips an sich sowie über die allgemeine Funktionsweise dieser Komponenten. Dies war eine essentielle Basis, um ein fundiertes Verständnis für die weiteren Aufgaben zu entwickeln. Ein weiterer bedeutender Teil meiner Woche bestand darin, geeignete Tools für die Simulation von Technik zu suchen. Hierbei habe ich verschiedene Optionen in Betracht gezogen, darunter Fritzing, Proteus und Tinkercad. Nach einer eingehenden Prüfung und dem Vergleich der Funktionalitäten habe ich mich letztlich für Tinkercad entschieden, da es eine benutzerfreundliche Oberfläche bietet und gut für meine Anforderungen geeignet ist.</p> <p>Die Recherche nach Tools für die 3D-Druck-Modellierung gestaltete sich ähnlich. Ich habe mehrere Tools geprüft, darunter Fusion 360, FreeCAD und Tinkercad. Aufgrund seiner einfachen Handhabung und der nahtlosen Integration in die Entwicklungsumgebung habe ich mich auch hier für Tinkercad entschieden. Für die Entwicklungstools habe ich mich mit verschiedenen Optionen auseinandergesetzt, darunter PlatformIO, Visual Studio Code und die Arduino-IDE. Nach sorgfältiger Abwägung habe ich mich für die Arduino-IDE entschieden, da sie speziell für die Programmierung von Mikrocontrollern entwickelt wurde und eine grosse Community-Unterstützung bietet.</p> <p>Die Erstellung der Dokumentation und das Einrichten eines Grundlayouts gehörten zu den strukturellen Aufgaben dieser Woche. Diese Tätigkeiten fielen mir vergleichsweise leicht, da sie stark methodisch geprägt waren und klare Anforderungen erfüllten. Das erneute saubere Durchlesen der Anforderungen war ebenfalls ein wichtiger Schritt. Ich bin zuversichtlich, dass ich das Projekt wie erwartet oder vorgegeben abschliessen kann.</p> <p>Während der gesamten Woche erwies sich die Nutzung der zur Verfügung stehenden Hilfsmittel als äusserst hilfreich. Die Unterlagen für SOL, GPT und das Internet boten eine breite Informationsbasis und unterstützten mich dabei, die Aufgaben effizient zu bewältigen. Besonders herausfordernd war es, die Vielzahl an Informationen zu filtern und die relevantesten Inhalte für meine Arbeit herauszusuchen. Rückblickend war die Woche intensiv und lehrreich. Ich habe viel über die verschiedenen Tools und deren Einsatzmöglichkeiten gelernt und konnte meine Fähigkeiten in der Informationsbeschaffung und -verarbeitung weiterentwickeln. Trotz einiger Herausforderungen, insbesondere bei der Tool-Recherche, war die Woche insgesamt erfolgreich.</p>		

Nächster Schritt
<ul style="list-style-type: none">- Übertragen meines mentalen Modells in eine nachvollziehbare Form- Generelle grundlegende Planung für das Projekt

3 - Arbeitsrapport 13.05 – 19.05

5.2 Woche 2: 20.05 – 26.05

Aktivität	Aufwand geplant (h)	Aufwand effektiv (h)
Planung, Abstecken der Rahmenbedingungen	7.5	7.5
Wochenablauf		
<ul style="list-style-type: none"> - Hardware Spezifizieren - Software Spezifizieren - Detaillierten Zeitplan erstellen - Ausarbeiten der Systemarchitektur - Entwurf eines Testdrucks um Abmessungen zu überprüfen 		
Hilfestellung		
<ul style="list-style-type: none"> - GPT - CratesIO - DrawIO - Internet - Tinkercad 		
Reflektion		
<p>In der vergangenen Woche habe ich mich intensiv mit der Spezifizierung von Hardware und Software, der Erstellung eines detaillierten Zeitplans, der Ausarbeitung der Systemarchitektur und dem Entwurf eines Testdrucks beschäftigt. Der Arbeitsprozess war insgesamt positiv, aber nicht ohne Herausforderungen. Die Spezifizierung der Hardware stellte keinen grossen Aufwand dar, da die Komponenten bereits im Vorfeld festgelegt und bestellt worden waren. Diese Phase bestand hauptsächlich aus dokumentarischem Festhalten der vorhandenen Hardware, um die Nachvollziehbarkeit zu gewährleisten.</p> <p>Die Softwarespezifizierung verlief ebenfalls relativ einfach. Dank eines grundlegenden Modells, das ich bereits im Kopf hatte, konnte ich die Planung für die einzelnen Softwarekomponenten schnell dokumentarisch festhalten. Mithilfe von Crates.io suchte ich notwendige Libraries zusammen und machte mich schlau über geeignete Libraries für Arduino, wie zum Beispiel den PubSubClient für die MQTT-Kommunikation. Anfangs hatte ich Schwierigkeiten, die Programmierung des Rust-Clients für MQTT zu verstehen, doch durch das gründliche Lesen der Dokumentation lösten sich diese Unklarheiten schnell.</p> <p>Die Erstellung eines detaillierten Zeitplans war eine etwas komplexere Aufgabe, da es mir oft schwerfällt, passende Bezeichnungen für Arbeitspakete zu finden. Dennoch verlief der Prozess insgesamt reibungslos und half mir, einen klaren Überblick über den Projektverlauf zu behalten.</p> <p>Die Ausarbeitung der Systemarchitektur basierte stark auf der zuvor spezifizierten Software. Dieser Schritt war grösstenteils dokumentarisch, und mithilfe von Draw.io konnte ich die Systemarchitektur visuell darstellen, was den Prozess erleichterte.</p> <p>Der Entwurf eines Testdrucks, um die Abmessungen zu überprüfen, war eine neue Erfahrung für mich, da ich noch nie zuvor mit Tinkercad gearbeitet hatte. Die Plattform erwies sich jedoch als sehr benutzerfreundlich, und ich konnte meinen ersten Sketch problemlos erstellen.</p> <p>Insgesamt war die Woche geprägt von einer guten Vorbereitung und klaren Vorstellungen, was die einzelnen Schritte erleichterte. Trotz kleinerer Herausforderungen konnte ich alle Aufgaben erfolgreich bewältigen und dabei neue Erfahrungen und Kenntnisse sammeln.</p>		

Nächster Schritt
<ul style="list-style-type: none">- Die Dokumentation der Planung nochmal etwas Abrunden- Die Entscheidung für die Komponenten nachvollziehbar Dokumentieren- Begin der Realisierung

4 - Arbeitsrapport 20.05 – 26.05

5.3 Woche 3: 27.05 – 02.06

Aktivität	Aufwand geplant (h)	Aufwand effektiv (h)
Planung abschliessen, Entscheiden und mit der Realisierung Beginnen	10.8	12.5
Wochenablauf		
<ul style="list-style-type: none"> - Testkonzept für das fertige System erstellen - Entscheidungen erläutern - Schaffung des Behälters für die Wassersteuerung - Schreiben der initial Steuerungssoftware für beide Komponenten - Initialversion des MQTT Servers 		
Hilfestellung		
<ul style="list-style-type: none"> - Erstellte Planung - GPT - Internet - Unterlagen des Testing Moduls 		
Reflektion		
<p>In dieser Woche habe ich intensiv an verschiedenen Aspekten meines Projekts gearbeitet, was sowohl Herausforderungen als auch wertvolle Lernerfahrungen mit sich brachte. Zu Beginn habe ich das Testkonzept für das fertige System erstellt. Dabei konzentrierte ich mich darauf, sinnvolle Test Cases zu entwickeln, die als Anhaltspunkte während der Systementwicklung dienen. Diese Methode half mir sicherzustellen, dass die Kernpunkte meiner Applikation immer funktionieren.</p> <p>Während der Dokumentation erläuterte ich die getroffenen Entscheidungen detailliert, besonders im Bereich des Messagings und der verwendeten Tools. Dieser Prozess erforderte eine klare und logische Darstellung meiner Gedanken und Entscheidungen, was mir gut gelungen ist.</p> <p>Die Schaffung des Behälters für die Wassersteuerung war eine besonders interessante Aufgabe. Ich verwendete ein altes Tupperware, das ich entsprechend präparierte, um die Schaltung vor Witterungseinflüssen zu schützen. Dieser Prozess ermöglichte es mir, kreativ zu sein und eine praktische sowie kosteneffiziente Lösung zu finden. Die Arbeit daran war für mich einfach und bereichernd.</p> <p>Beim Schreiben der initialen Steuerungssoftware für beide Komponenten konnte ich meine Fähigkeiten weiterentwickeln. Obwohl es sich vorerst nur um eine Prototyp-Entwicklung handelte, war die grundlegende Funktionalität gegeben, und ich musste lediglich kleine Anpassungen vornehmen. Ein anfängliches Problem mit den LEDs des ESP8266-Chips konnte ich durch sorgfältiges Studium der Dokumentation lösen.</p> <p>Die Erstellung der Initialversion des MQTT-Servers stellte eine weitere Herausforderung dar. Durch das gründliche Durchlesen der Dokumentation der MQTT-Library für Rust konnte ich jedoch schnell ein grundlegendes Konzept und einen funktionsfähigen Code für den Broker entwickeln. Dieser ist nun in einem separaten Projekt implementiert und soll gemäss der Planung über den Hub gestartet werden.</p> <p>Zu guter Letzt nutzte ich verschiedene Hilfsmittel, die mir bei der Arbeit sehr halfen. Die Unterlagen des Testingsmoduls halfen mir, wichtige Aspekte der Testdeklaration abzudecken. GPT und das Internet dienten als allgemeine Ressourcen für nahezu jede Aufgabe. Die erstellte Planung war besonders für die Erschaffung des Testkonzepts sowie das Schreiben der Initialsteuerungssoftware und des MQTT-Servers von grosser Bedeutung.</p> <p>Insgesamt war die Woche sehr produktiv und lehrreich. Ich konnte sowohl technische als auch kreative Fähigkeiten weiterentwickeln und habe wertvolle Erfahrungen gesammelt, die mir in zukünftigen Projekten zugutekommen werden.</p>		

Nächster Schritt
<ul style="list-style-type: none">- Prototyp Realisieren- Begin der Auswertung

5 - Arbeitsrapport 27.05 – 02.06

5.4 Woche 4: 03.05 – 09.06

Aktivität	Aufwand geplant (h)	Aufwand effektiv (h)
Realisierung, beginn der Auswertung	13.6	12
Wochenablauf		
<ul style="list-style-type: none"> - Zusammenbau der Wassersteuerung - Zusammenbau des Sensors - Programmierung des Sensors - Programmierung der Wassersteuerung - Weiterentwicklung der 3D Modelle - Programmierung des Hubs 		
Hilfestellung		
<ul style="list-style-type: none"> - Erstellte Planung - GPT - Internet 		
Reflektion		
<p>In der vergangenen Woche habe ich intensiv am Zusammenbau und der Programmierung der Wassersteuerung sowie des Sensors gearbeitet. Der Zusammenbau der Wassersteuerung verlief dank einer Anleitung von GPT relativ reibungslos. Anfangs hatte ich Schwierigkeiten mit dem Relay, da ich nicht sofort verstand, wie es funktioniert. Nach einigen Tests und Experimenten konnte ich jedoch die Funktionsweise isolieren und die Probleme in meiner Schaltung beheben.</p> <p>Die Verbindung des Sensors zum Microcontroller gestaltete sich unkompliziert, da nur drei Verbindungen erforderlich waren: Plus, Minus und der Output für die Readings. Die Programmierung des Sensors verlief ebenfalls problemlos. Ich konnte die Protokollnachrichten mit einem lokalen MQTT-Client testen und sicherstellen, dass die erwarteten Aktionen bei den entsprechenden Eingaben erfolgen. Nachdem der Sensor funktionsfähig war, konnte ich die Programmdatei kopieren und für die Wassersteuerung anpassen.</p> <p>Die Entwicklung der 3D-Modelle verlief weitgehend reibungslos. Ich überprüfte die Abmessungen aller Teile erneut und entwickelte ein Gehäuse aus mehreren Teilen, das den Chip, den Sensor und eine Powerbank aufnehmen soll. Diese Teile werden anschliessend zusammengeleimt.</p> <p>Die Programmierung des Hubs war aufgrund meiner Vorkenntnisse in Rust relativ einfach. Ich konnte schnell ein funktionierendes Grundprogramm erstellen und die Kommunikation mit den MQTT-Clients auf meinem Desktop testen. Herausforderungen gab es bei der Modulkonfiguration, da ich unsicher war, wie ich diese sinnvoll programmieren und erweitern kann. Schliesslich entwickelte ich eine Modulmanager-Struktur in Kombination mit dem Client-Modul, die erweiterbar und effizient ist. Mit dieser Lösung bin ich sehr zufrieden, da sie übersichtlich und leistungsfähig ist.</p> <p>Zusammenfassend war die Arbeit in dieser Woche sowohl herausfordernd als auch befriedigend. Die grössten Schwierigkeiten bestanden darin, mich wieder in C++ einzuarbeiten und die Modulkonfiguration des Hubs zu optimieren. Dennoch konnte ich durch Geduld und iterative Tests alle Probleme lösen und bin stolz auf das erreichte Ergebnis.</p>		
Nächster Schritt		
<ul style="list-style-type: none"> - Vollständige Auswertung - Finalisierung der Dokumentation 		

6 - Arbeitsrapport 03.05 – 09.06

5.5 Woche 5: 10.06 – 14.06

Aktivität	Aufwand geplant (h)	Aufwand effektiv (h)
Auswertung, Finalisierung und Abgabe	7.1	4
Wochenablauf		
<ul style="list-style-type: none"> - Durchführung der Tests - Abgleichen der Ziele - Reflektion und Fazit ausarbeiten - Finalisierung der Dokumentation 		
Hilfestellung		
<ul style="list-style-type: none"> - Erstellte Planung - Arbeitsrapporte 		
Reflektion		
<p>Diese Woche war für mich geprägt von routinierten und strukturierten Aufgaben, die jedoch unterschiedliche Herausforderungen mit sich brachten. Zu Beginn der Woche habe ich die Tests für die Applikationen durchgeführt, um sicherzustellen, dass alle Funktionen wie geplant arbeiten. Diese Aufgabe war routinemässig, aber dennoch essenziell, um die Stabilität und Funktionalität der Anwendungen zu bestätigen. Das Dokumentieren der Testergebnisse verlief reibungslos und stellte sicher, dass der gesamte Prozess nachvollziehbar und gut dokumentiert ist.</p> <p>Beim Abgleich der Ziele musste ich etwas genauer arbeiten. Hier habe ich die Anforderungen nochmals durchgelesen und mein Projekt dahingehend überprüft und angepasst. Obwohl dies ebenfalls eine Routineaufgabe war, erforderte sie mehr Konzentration, um sicherzustellen, dass alle Anforderungen korrekt umgesetzt und dokumentiert sind.</p> <p>Die Reflexion und das Fazit auszuarbeiten war eine der anstrengendsten Aufgaben dieser Woche. Das Zusammenfassen der Arbeitsrapporte und das Schreiben einer prägnanten Reflexion fiel mir schwer, da ich oft dazu neige, abzuschweifen und zu ausführlich zu werden. Dennoch habe ich festgestellt, dass meine Reflexionen im Vergleich zu früheren Arbeiten deutlich besser strukturiert und fokussierter sind.</p> <p>Die Finalisierung der Dokumentation stellte den letzten Schritt dar. Hier habe ich nochmal alle Teile durchgelesen, Rechtschreibfehler korrigiert und sichergestellt, dass die Struktur sauber und lesbar ist. Dabei habe ich auch alle Verzeichnisse und Quellen aktualisiert, um eine vollständige und kohärente Dokumentation zu gewährleisten.</p> <p>Insgesamt war die Woche zwar anspruchsvoll, aber auch sehr produktiv. Ich konnte alle Aufgaben erfolgreich abschliessen und habe dabei wertvolle Erfahrungen gesammelt, die mir in zukünftigen Projekten sicherlich helfen werden.</p>		
Nächster Schritt		
<ul style="list-style-type: none"> - Erstellung der Präsentation 		

7 - Arbeitsrapport 10.06 – 14.06

6 Protokolle

6.1 Kickoff-Meeting

Nicht anwesend, Entschuldigt. Protokoll aus Nachriten und PDF.

Ablauf:

- Einführung SOL
- Festlegung der Meilensteine
- Begründung, Motivation und Aufwand rahmen
- Grobe Bewertungsübersicht
- Auftrag für den Projektantrag, Erklärung des Budgets
- Benötigte Inhalte für die erfolgreiche Durchführung von SOL
- Formelle und Inhaltliche Vorgaben für die Dokumentation
- Besprechung der Präsentation und des Umfeldes für die Bewertung

6.2 1. Meeting

Ablauf:

- Repetition: Einleitung in SOL
- Vorstellung der Dokumentationsstruktur
- Abgrenzung des Inhalts für die Dokumentation
- Besprechung der Projektanträge
- Vorstellung der Bewertungskriterien
- Weitere Zeit für die Arbeit an dem Projekt
- Statusbericht bis am Ende des Tages

Statusbericht: 07.05.2024

Projektauftrag

Der Projektantrag wurde formuliert und genehmigt. Die Teile für den Test sind bestellt und sollen bis zum 15.05.2024 eintreffen. Danach wird die Projektzeitschiene (SOLL-Zeit) detailliert geplant.

Projektverlauf

Eine detaillierte Projektplanung fehlt noch; es existiert nur ein grobes Konzept. Die genaue Planung wird nach Eintreffen der Bauteile erstellt.

Arbeitsrapport

Die benötigten Komponenten wurden bestellt. Eine detaillierte Erfassung der Arbeitsstunden und Fortschritte beginnt nach Eintreffen dieser Komponenten.

Aktueller Stand der Dokumentation

Die Projektdokumentation hat noch nicht begonnen. Sie wird gemeinsam mit der detaillierten Planung entwickelt.

6.3 2. Meeting

6.3.1 Teil 1

Ablauf:

- Besprechung des Aktuellen Projektes
- Festlegung der Abgabe
- Erläuterungen zu Präsentationen
 - o 10-15min
 - o Nicht vor der Ganzen Klasse, nur Experten
 - o Hochdeutsch, mind. 2 Medienmittel
 - Handout als 2. Medienmittel
 - o Grober Inhalt
 - Ausgangslage
 - Antrag
 - Auftrag
 - Vorgehen Technisch/Organisatorisch
 - Gute/Schwere dinge
 - Probleme
 - Lösungen
 - Fazit & Reflektion
- Erläuterungen zu Demo
 - o Vorbereitet, Mundart
 - o Empfehlung 3 Punkte
 - o Ca. 5min (nicht bewertet)
- Fachgespräch
 - o 3 Themengebiet
 - o Ca. 10min
- Kurzes Feedback mündlich
 - o Note schriftlich am Ende des Semesters

Statusbericht 17.05.2024

Projektverlauf

Heute beginnt das Projekt, es gibt also noch keinen Zeitlichen Rahmen, diesen werde ich im Verlauf der Nächsten Woche erarbeiten.

Arbeitsrapport

Für alle Berichte wurden Leere Tabellen in der Dokumentation angelegt.

Aktueller Stand der Dokumentation

Mit der Dokumentation wurde heute begonnen, aktuell besteht sie nur aus dem Grundlegenden Layout.

6.3.2 Teil 2

Ablauf:

- Besprechung des laufenden Projektes
- Repetition: Abgabe der Dokumentation
- Repetition: Präsentation und Demo
- Repetition: Bewertung
- Genauere Erklärung zum Meeting Protokoll
 - o Grundsätzlich einfach der Statusbericht

Statusbericht: 28.05.2024

Projektverlauf

Eine detaillierte Projektplanung wurde erstellt, Heute endet die Planungsphase ein Konzept wurde erstellt. Ich befinde mich Zeitlich gut in der Planung.

Arbeitsrapport

Die Arbeitsrapporte für die erledigten aufgaben wurden direkt in der Dokumentation erstellt, der Projekt Start wurde auf den 17.05.2024 gelegt. An diesem Tag sind die Benötigten Komponenten eingetroffen.

Aktueller Stand der Dokumentation

Die Projektdokumentation wurde erstellt und mit Inhalten bis zum Aktuellen Zeitpunkt befüllt.

Eigene Punkte

Projektorganisation für Dokumentation:

- Ohne SOL spezifische Inhalte

Tech. Summary

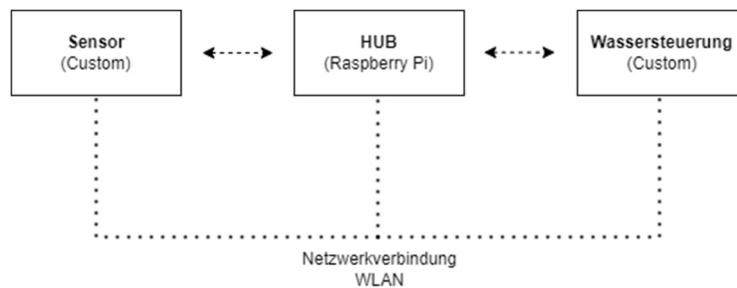
- Positionierung: nach Titelblatt

7 Systemdokumentation

7.1 Systemarchitektur

Das System besteht aus vernetzten Komponenten, die über WLAN kommunizieren. Der Hub fungiert als zentraler Knotenpunkt, der die Koordination vom Sensor und der Wassersteuerung ermöglicht.

Ziel dieser Architektur ist es, die Erweiterbarkeit zu gewährleisten, damit in späteren Versionen mehrere Komponenten in das System eingebunden werden können.



1 - Systemarchitektur

7.1.1 Sensor

Der Sensor bildet das Herzstück des Systems und spielt eine zentrale Rolle bei der Ermittlung des Wasserbedarfs. Mithilfe eines Bodenfeuchtigkeitssensors misst er regelmässig die Feuchtigkeit im Boden. Kurz vor der geplanten Bewässerung überprüft der Sensor, ob der Wassergehalt unter einem festgelegten Schwellenwert liegt. Ist dies der Fall, erkennt der Sensor einen Wasserbedarf und sendet eine entsprechende Nachricht an den Hub. Um den Energieverbrauch zu minimieren, versetzt sich der Sensor nach jeder Messung in den Schlafmodus und wird nur zu den festgelegten Überprüfungszeiten aktiviert. Dies gewährleistet eine langfristige und energieeffiziente Betriebsdauer.

7.1.2 Wassersteuerung

Die Wassersteuerung ist für die Regulierung des Wasserflusses verantwortlich. Sie besteht grob aus einem Ventil, das von einem Mikrocontroller gesteuert wird. Bei einer Bewässerungsanfrage vom Hub, schaltet der Mikrocontroller über ein Relais-Modul das Ventil. Nach Ablauf der festgelegten Bewässerungsdauer schliesst der Mikrocontroller das Ventil wieder, indem er das Relais deaktiviert, wodurch der Wasserfluss gestoppt wird und eine Überbewässerung verhindert wird. Auch die Wassersteuerung wechselt zwischen den Bewässerungszyklen in den Schlafmodus, um den Stromverbrauch zu reduzieren.

7.1.3 Hub

Der Hub fungiert als zentrale Steuer- und Kommunikationsschnittstelle des gesamten Systems. Er stellt den Broker für die Kommunikation. Der Hub empfängt die Feuchtigkeitsmessungen des Sensors und speichert diese Intern. Wird dieser Status abgefragt sendet der Hub den aktuellen Stand. Einstellungen sollen über das Hub erfolgen. Dies ermöglichen eine zentrale Verwaltung und einfache Anpassung des Systems. Aktuell unternimmt der Hub keine Spezifischen Massnahmen für die Minderung des Energieverbrauchs, abgesehen von effizienter Programmierung.

7.2 Systemanforderungen

7.2.1 Funktionale Anforderungen

Das TerraTap-Bewässerungssystem besteht aus mehreren Komponenten, die spezifische Aufgaben erfüllen. Das Sensor-Element und das Wassersteuerungs-Element sind zentrale Bestandteile des Systems. Beide Komponenten sollen zu festgelegten Zeiten Überprüfungen durchführen und ihre Ergebnisse an den Hub übermitteln. Der Hub ist dafür verantwortlich, die Anfragen für Bewässerungen zu verarbeiten und zu speichern. Auf Anfrage sendet der Hub diese Informationen an das Wassersteuerungs-Element, das daraufhin die Bewässerung durchführt. Darüber hinaus muss das Wassersteuerungs-Element aktiv nach ausstehenden Bewässerungsanfragen beim Hub nachfragen, um sicherzustellen, dass alle Pflanzen bedarfsgerecht versorgt werden.

7.2.2 Nicht-funktionale Anforderungen

Neben den funktionalen Anforderungen muss das System auch eine Reihe nicht-funktionaler Anforderungen erfüllen. Es ist essenziell, dass das TerraTap-System zugänglich ist, sodass Personen mit den nötigen Kenntnissen in der Lage sind, Modifikationen vorzunehmen. Dies fördert die Anpassungsfähigkeit und Erweiterbarkeit des Systems. Zudem wurde besonderer Wert auf die Energieeffizienz gelegt. Da viele Komponenten batteriebetrieben sind, soll das System möglichst stromsparend arbeiten, um die Betriebsdauer zu maximieren und Wartungsintervalle zu minimieren.

7.3 Sicherheitsaspekte

In der aktuellen Prototyp-Konfiguration wurden keine erweiterten Sicherheitsmassnahmen implementiert. Der Fokus lag zunächst auf der Sicherstellung der grundlegenden Funktionalität. Für zukünftige Iterationen wird jedoch die Implementierung zusätzlicher Sicherheitsvorkehrungen empfohlen. Beispielsweise könnte die Authentifizierung für MQTT-Nachrichten eingeführt werden, um sicherzustellen, dass Nachrichten nur von autorisierten Quellen stammen. Dies würde die Sicherheit und Zuverlässigkeit des Systems weiter erhöhen.

7.4 Entwicklungsumgebung

Die Entwicklungsumgebung des Projekts umfasst verschiedene Tools und Plattformen, die sowohl die Hardware- als auch die Softwareentwicklung unterstützen. Diese wurden sorgfältig ausgewählt, um die Anforderungen des Projekts bestmöglich zu erfüllen. Der Entscheidungsprozess für die Auswahl der Tools basiert auf umfassender Recherche und Abwägung verschiedener Optionen hinsichtlich ihrer Funktionalitäten, Benutzerfreundlichkeit und Integration in den Entwicklungsprozess.

7.4.1 Einrichtung Arduino

Für die Entwicklung mit dem ESP8266-Chip wird die Arduino IDE verwendet. Diese bietet eine benutzerfreundliche Oberfläche und unterstützt die einfache Integration von Bibliotheken durch die richtigen Treiber. Um das Board für den ESP8266 hinzuzufügen, muss die folgende URL unter „Additional Boards Manager URLs“ in den Einstellungen der Arduino IDE hinterlegt werden:

http://arduino.esp8266.com/stable/package_esp8266com_index.json

Für dieses Projekt wird hauptsächlich eine Bibliothek dritter verwendet, welche einer zusätzlichen Installation bedarf: «PubSubClient» diese kann über den Library Manager in der Arduino IDE installiert werden.

Zusätzlich werden eigens entwickelte Bibliotheken verwendet. Diese werden im **clients/lib** Ordner des Repositories abgelegt und können mithilfe der „install.ps1“ installiert werden, nach der Installation muss die Arduino IDE neugestartet werden, damit die Bibliotheken richtig geladen werden.

Begründung der Auswahl: Die Arduino IDE ist speziell für Mikrocontroller-Programmierung entwickelt und bietet eine grosse Community-Unterstützung sowie eine einfache Integration von Bibliotheken.

7.4.2 Einrichtung Visual Studio Code

Für die Entwicklung des HUBs wird Visual Studio Code Verwendet. Zur Arbeit mit Rust werden folgende Erweiterungen Verwendet:

Erweiterung	Beschreibung
rust-analyzer	Rust language support for Visual Studio Code
crates	Helps Rust developers managing dependencies with Cargo.toml.
Dracula For Rust Theme	Bietet ein auf die Programmiersprache Rust zugeschnittenes Dracula-Theme.
Even Better TOML	Umfassende TOML-Unterstützung
Prettier - Code formatter (Rust)	Prettier Rust ist ein Code-Formatierer, der schlechte Syntax automatisch korrigiert
Rust Test Explorer	Anzeigen und Ausführen Ihrer Rust-Tests in der Seitenleiste von Visual Studio Code
Error Lens	Verbesserte Hervorhebung von Fehlern, Warnungen und anderen Sprachdiagnosen.
Todo Tree	TODO-, FIXME- usw. Kommentar-Tags in einer Baumansicht anzeigen
GitHub Copilot	«Your AI pair programmer»

8 - Erweiterungen Visual Studio Code

Begründung der Auswahl: Visual Studio Code ist ein vielseitiger Editor mit umfangreicher Erweiterbarkeit und Unterstützung für verschiedene Programmiersprachen. Die Verfügbarkeit von Extensions wie rust-analyzer und GitHub Copilot erhöht die Effizienz der Entwicklung erheblich.

7.4.3 Weitere Entwicklungswerkzeuge

Tinkercad

Tinkercad wird für die Simulation von elektronischen Schaltungen und die 3D-Modellierung verwendet.

Begründung der Auswahl: Tinkercad bietet eine benutzerfreundliche Oberfläche und ermöglicht eine schnelle und einfache Erstellung von Schaltungsdesigns sowie 3D-Modellen. Die Integration der beiden Funktionen in einem Tool erleichtert den Entwicklungsprozess und reduziert den Wechsel zwischen verschiedenen Programmen.

7.5 Konzeption Software

7.5.1 Steuerungssystem

Die Chips für Sensor und Wassersteuerung arbeiten beide mit CP2102.

Was eine Verbindung über USB erlaubt. Folgende Bibliotheken werden verwendet:

Bibliothek	Beschreibung
ESP8266WIFI	Ist eine Arduino-Bibliothek, die die Integration und Steuerung von ESP8266 WiFi-Modulen ermöglicht. Sie vereinfacht die Konfiguration von Netzwerken, das Herstellen von Verbindungen und die Verwaltung von WiFi-Funktionen für IoT-Anwendungen.
PubSubClient	Ist eine Arduino-Bibliothek, die das MQTT-Protokoll unterstützt. Sie ermöglicht die einfache Kommunikation und den Nachrichtenaustausch zwischen Geräten in IoT-Netzwerken, indem sie als MQTT-Client fungiert.

Grundlegende Beschreibungen von GPT

9 – Steuerungssystem Bibliotheken

Zusätzlich werden eigens entwickelte Bibliotheken verwendet welche Funktionalität die für beide der Systeme Benötigt wird zur Verfügung stellen. Diese sind im **clients/lib** Ordner des Repositories.

7.5.2 Hub

Das HUB wird in Rust Programmiert und fungiert als Zentraler Kontaktpunkt für alle Komponenten. Hier werden Einstellungen getätigt und bei Jeder Überprüfung an Entsprechende Komponenten Übermittelt.

Das HUB verwendet den ModuleManager zur Verwaltung der Module und deren Einstellungen. Alle Module sollten beim Manager registriert werden und die `handle_message` Funktion sollte von der Hauptschleife aufgerufen werden. Der Manager hält eine Liste von Modulen und deren Konfigurationen.

Die Module implementieren das `ClientModule Trait`, das Methoden für das Handhaben von Nachrichten und das Abrufen von Einstellungen bereitstellt. Es gibt spezifische Module wie das `WateringModule` und das `SensorModule`, die spezifische Funktionen für die Bewässerung und Sensorik bereitstellen.

Für Rust werden die folgenden Pakete verwendet:

Bibliothek	Beschreibung
<code>serde</code>	Framework für Serialisierung und Deserialisierung.
<code>serde_json</code>	JSON-Serializer und -Deserialzer auf Basis von Serde.
<code>tokio</code>	Asynchrone Laufzeitumgebung für asynchrone I/O in Rust.
<code>tracing</code>	Bibliothek für ereignisgesteuertes Debugging und Logging.
<code>tracing-subscriber</code>	Abonnement für Tracing-Daten, Unterstützung Filtering und Formatierung.
<code>rumqttc</code>	MQTT-Client für asynchrone Kommunikation.
<code>once_cell</code>	Thread-sichere Initialisierung von statischen Variablen.
<code>chrono</code>	Bibliothek für Datums- und Zeitmanipulationen.
<code>async-trait</code>	Unterstützung für asynchrone Funktionen in Traits.

Grundlegende Beschreibungen von GPT

10 - Hub Bibliotheken

Für die Kommunikation wird ein Broker benötigt, dieser wird mithilfe des HUBs gestartet und gestoppt allerdings als separate Rust Applikation umgesetzt. Diese Applikation verwendet die Folgenden Bibliotheken:

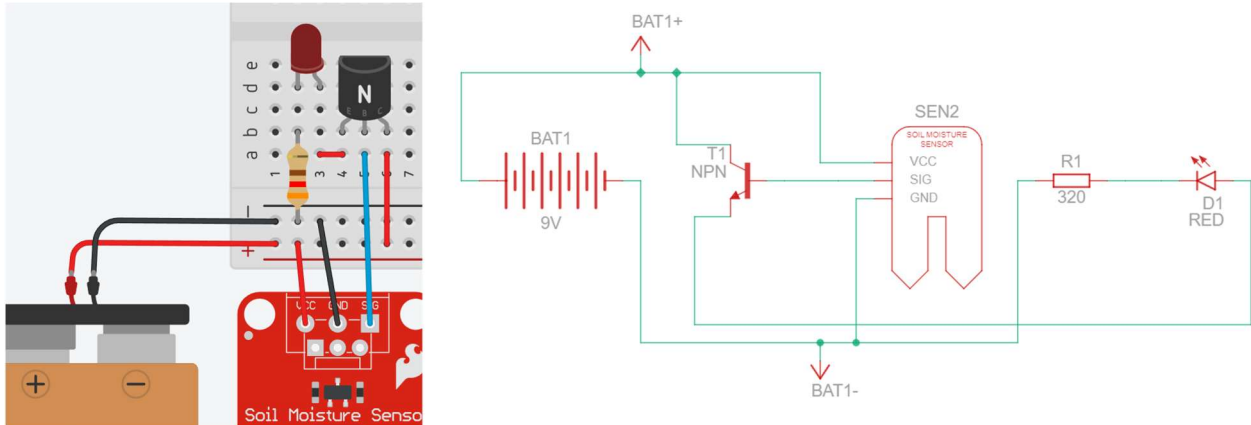
Bibliothek	Beschreibung
<code>rumqttd</code>	MQTT-Server zur Verarbeitung und Weiterleitung von Nachrichten.
<code>tracing</code>	Bibliothek für ereignisgesteuertes Debugging und Logging.
<code>tracing-subscriber</code>	Abonnement für Tracing-Daten, Unterstützung für Filtering und Formatierung.
<code>tokio</code>	Asynchrone Laufzeitumgebung für asynchrone I/O in Rust.
<code>config</code>	Bibliothek zur Verwaltung von Konfigurationsdateien. Wird verwendet von <code>rumqttd</code> .

Grundlegende Beschreibungen von GPT

11- MQTDD Bibliotheken

7.6 Konzeption Hardware

7.6.1 Sensor Schaltkreis

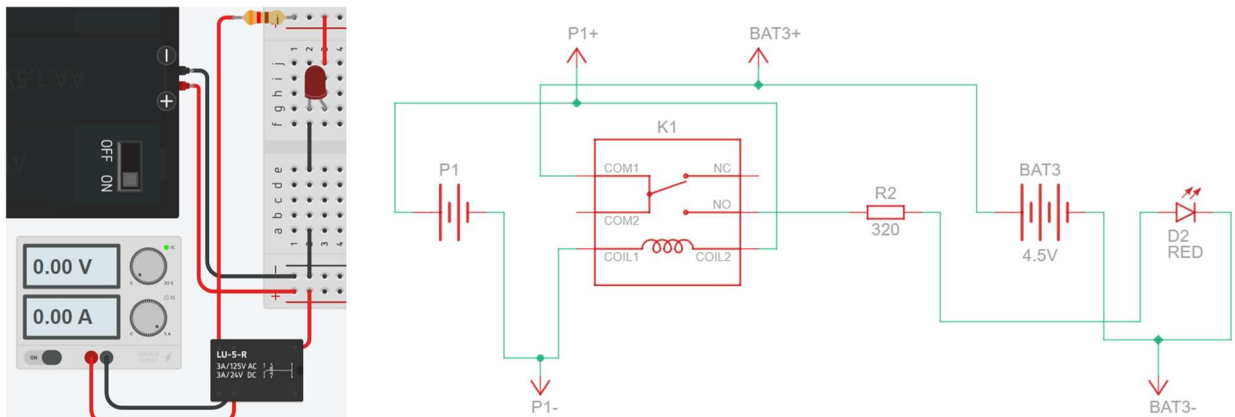


2 - Sensor Schaltkreis (Visuell links, Technisch rechts)

Beschreibung: Dieses Bild stellt einen Schaltkreis dar, die Batterie liefert Strom, der Sensor misst die Bodenfeuchtigkeit. Wenn die Feuchtigkeit steigt wird ein Signal auf den Transistor gegeben welcher dann den Stromfluss zum LED ermöglicht.

In der Realität wird das Signal des Feuchtigkeitssensors nicht in ein Transistor->LED sondern direkt in einen Micro Chip gehen welcher dann über MQTT mit dem HUB Kommunizieren kann.

7.6.2 Wassersteuerung Schaltkreis



3 - Wassersteuerung Schaltkreis (Visuell links, Technisch rechts)

Beschreibung: Dieses Bild stellt einen Schaltkreis dar, er beinhaltet 2 Stromquellen. Die einstellbare Stromquelle (0.00V im Bild) schaltet bei 5V das Relay Modul um, was den Stromfluss für das LED vollendet und somit dessen Aktivierung erlaubt.

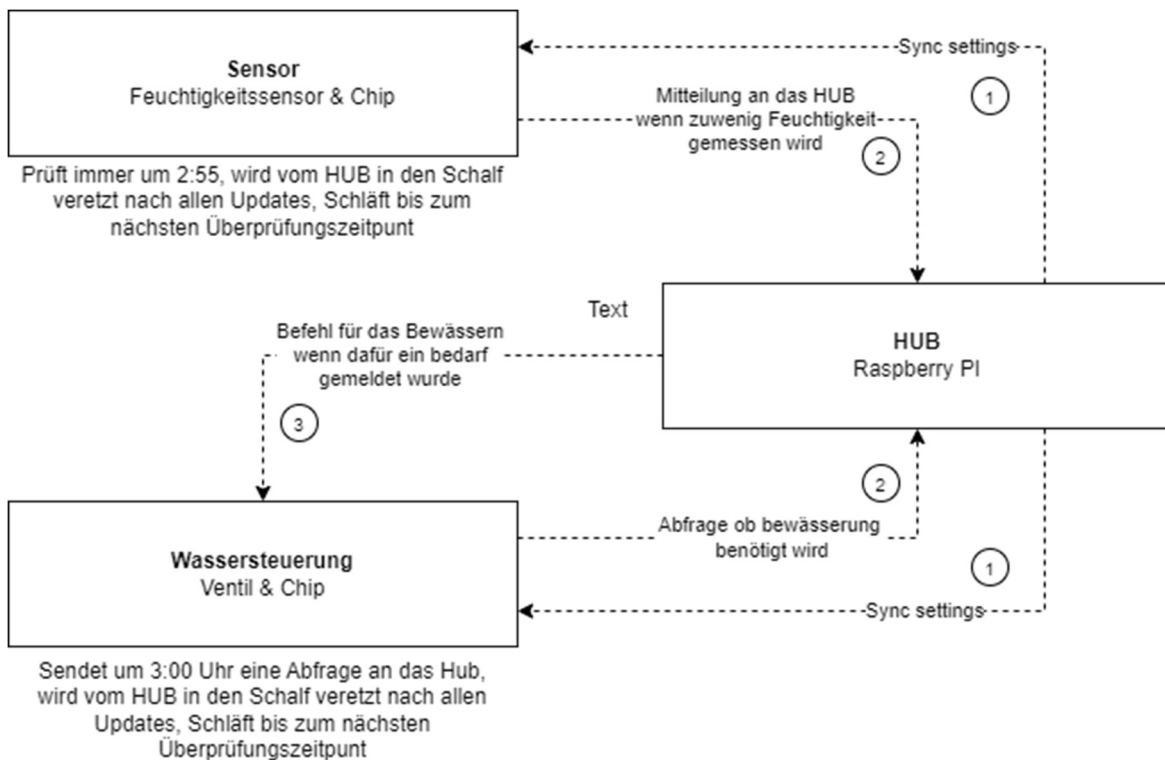
In der Realität wird das Umschalten des Relais nicht Manuell sondern mithilfe eines Micro Chips erledigt. Das LED spiegelt das Ventil wieder, denn für dessen Schaltung ist das Relay verantwortlich.

7.7 Kommunikation

MQTT wird als Kommunikationsprotokoll verwendet. Dieses Protokoll ermöglicht eine zuverlässige und skalierbare Kommunikation zwischen den Systemkomponenten. Das Protokoll läuft über WLAN und setzt voraus das alle Komponenten in WLAN Reichweite sind.

MQTT wurde anstelle von gRPC gewählt da es Flexibler in der Definition und Nutzung von Befehlen ist. Wo gRPC ausdefinierte Abläufe verlangt kann MQTT einfache Kanal Namen zur Unterscheidung anbieten. Auch die Verfügbarkeit von Test-Clients ist für MQTT besser gegeben als für gRPC.

Der MQTT Server soll auf dem HUB gehostet werden, er ist nicht das HUB selbst sondern eine separate Instanz welche mit dem HUB gestartet wird.



----- Kommunikation über MQTT

Der Prüfzeitpunkt ist Abhängig von der Einstellung und wird vom HUB an andere Elemente übermittelt. Er dient der Einsparung von Strom der Chips.

4 - Kommunikationsdiagramm

7.8 Hardware Komponenten

7.8.1 Sensor

Bezeichnung	Beschreibung
ESP8266 NodeMCU CP2102	Vielseitiges Entwicklungsboard basierend auf dem ESP8266, inklusive CP2102 Chip für Programmierung und Debugging, ideal für IoT-Projekte.
Moisture Sensor v2.0	Bodenfeuchtigkeitssensor, ideal für Garten- und automatisierte Bewässerungssysteme, zur einfachen Steuerung von Bewässerungssystemen

Grundlegende Beschreibungen von GPT

12 - Hardware Komponenten: Sensor

7.8.2 Wassersteuerung

Bezeichnung	Beschreibung
ESP8266 NodeMCU CP2102	Vielseitiges Entwicklungsboard basierend auf dem ESP8266, inklusive CP2102 Chip für Programmierung und Debugging, ideal für IoT-Projekte.
12V 1-Kanal Relaismodul	Zuverlässiges Relaismodul zur Steuerung von Hochleistungsgeräten mittels Niederspannungssignalen, geeignet für Automatisierungs- und Steuerungssysteme.
12V-zu-5V Spannungsregler-Modul	DC-DC Buck Converter, der 12V DC auf eine stabile 5V DC reduziert, perfekt für Projekte, die eine zuverlässige 5V-Versorgung benötigen.
Pneumatisches 12V-Wasserventil	12V Wasserventil, robust und zuverlässig für die Steuerung von Wasserflüssen, öffnet bei Stromzufuhr

Grundlegende Beschreibungen von GPT

13 - Hardware Komponenten: Wassersteuerung

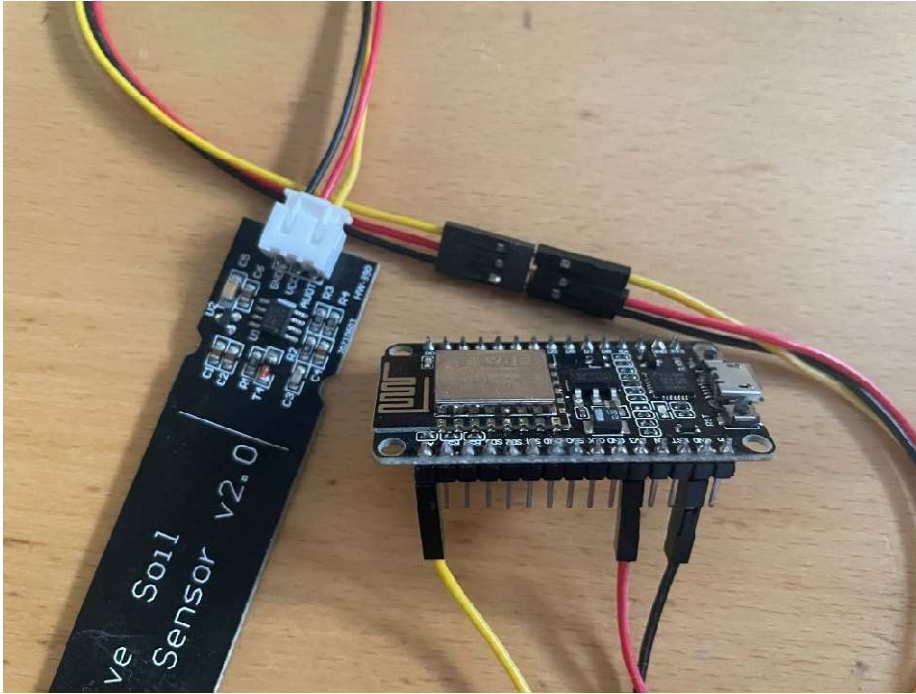
7.8.3 HUB

Bezeichnung	Beschreibung
Raspberry Pi	Ist ein Vielseitig einsetzbarer Einplatinencomputer.

14 - Hardware Komponenten: Hub

8 Bau

8.1 Sensor

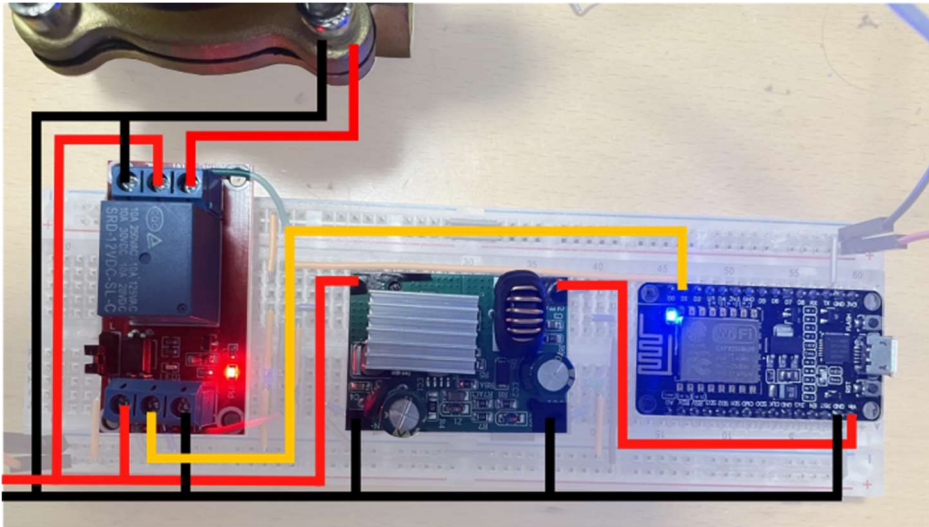


5 - Sensor Schaltkreis

Der Schaltkreis des Sensors ist sehr simpel. Der Sensor braucht eine Plus (Rot) und Minus (Schwarz) Verbindung, damit er arbeiten kann. Damit Daten empfangen werden können muss zusätzlich der AOUT-Pin des Sensors mit dem Mikrocontroller verbunden werden (Gelb). In diesem Aufbau werden die Informationen des Sensors über AO empfangen.

Dieses Modul soll mithilfe einer Powerbank betrieben werden. Diese wird über ein Micro-USB Kabel direkt mit dem Mikrocontroller verbunden.

8.2 Wassersteuerung



6 - Wassersteuerung Schaltkreis

+ (Rot) / - (Schwarz)

Während der Mikrochip 5V benötigt, braucht das Ventil 12V. Damit die Wassersteuerung mithilfe einer einzigen Stromquelle versorgt werden kann, wird mithilfe eines DC-DC Buck Converters von 12V auf 5V Konvertiert.

Der Mikrochip kann mithilfe der gelben Verbindung das Relais Modul Aktivieren. Wird das Relaismodul Aktiviert Schliesst sich der Kreislauf wodurch sich das Ventil Öffnet.

In der Aktuellten Version wird dieses Modul durch eine Konstante 12V Stromquelle Versorgt.

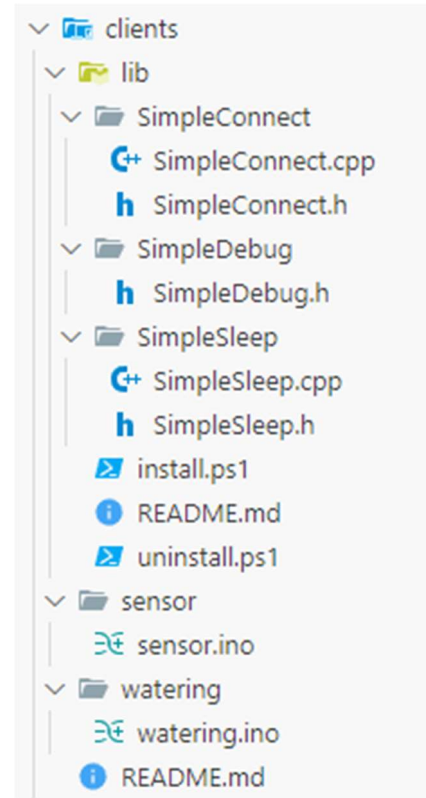
9 Programmcode: Module (Clients)

Programmcode des der Module.

Der clients/lib Ordner enthält die Bibliotheken, die von den verschiedenen Komponenten des Projekts verwendet werden. Diese Bibliotheken bieten gemeinsame Funktionalitäten, die von mehreren Komponenten genutzt werden.

Die Bibliotheken in diesem Ordner sind:

- **SimpleDebug:** Eine einfache Debugging-Bibliothek, die das Drucken von Debug-Nachrichten auf den seriellen Monitor ermöglicht. Sie bietet eine Syntax, die der console.log Funktion von JavaScript ähnelt.
- **SimpleConnect:** Eine einfache Bibliothek, die die Verbindung zu einem WiFi-Netzwerk ermöglicht. Sie bietet auch eine zusätzliche Hilfsfunktion, um den Prozess der Generierung einer Client-ID zu vereinfachen.
- **SimpleSleep:** Eine einfache Bibliothek, die es dem ESP32 ermöglicht, in den Deep Sleep Modus zu wechseln. Die Funktion sleepUntil erwartet einen Zeitstring, z.B. HH:MM, und versetzt den ESP32 bis zu diesem Zeitpunkt am nächsten Tag in den Schlaf.



Zur Installation der Bibliotheken führen Sie das Skript install.ps1 im Wurzelverzeichnis dieses Ordners aus. Da dies lediglich für die Entwicklung relevant ist wurde dieser Code nicht in die Dokumentation aufgenommen.

Die Beschreibungen für die Dateien wurden mithilfe von GPT grundlegend entworfen, von Fokko Vos kontrolliert und überarbeitet.

9.1 Lib

9.1.1 clients/lib/SimpleConnect/SimpleConnect.cpp

```
#include "SimpleConnect.h"

// WiFi and MQTT client instances
WiFiClient espClient;

// Configure the correct time
void setupTime() {
    debug("Syncing time...");
    // setoff for 2 hours
    configTime(2*3600, 0, "pool.ntp.org", "time.nist.gov");
    // Wait until time is set
    while (time(nullptr) < 8 * 3600 * 2) {
        delay(100);
    }

    time_t now = time(nullptr);
    struct tm* timeinfo;
    timeinfo = localtime(&now);
    char timeStr[9];
    strftime(timeStr, sizeof(timeStr), "%H:%M:%S", timeinfo);
    info("System time:", timeStr);
}

// Function to connect to Wi-Fi
void setup_wifi(const char* ssid, const char* password) {
    info("Connecting to WiFi:", ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        // fallback to Serial.print to not clutter the debug output
        Serial.print(".");
    }
    Serial.println();

    info("WiFi connected, IP address:", WiFi.localIP());
}

// Function to generate a random alphanumeric string
String generateRandomID(int length) {
    const char charset[] =
        "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
    String randomString = "";
    // Initialize random number generator seed from an unconnected pin
```

```
randomSeed(analogRead(0));

for (int i = 0; i < length; i++) {
    int index = random(0, sizeof(charset) - 1);
    randomString += charset[index];
}

return randomString;
}
```

Die Datei SimpleConnect.cpp enthält Funktionen zur Konfiguration der Systemzeit, zum Herstellen einer WiFi-Verbindung und zum Generieren einer zufälligen alphanumerischen Zeichenkette. Die Funktion setupTime() synchronisiert die Systemzeit mit einem NTP-Server. setup_wifi() stellt eine Verbindung zu einem WiFi-Netzwerk her, wobei SSID und Passwort als Parameter übergeben werden. Die Funktion generateRandomID() erzeugt eine zufällige alphanumerische Zeichenkette einer bestimmten Länge, die beispielsweise als eindeutige Geräte-ID verwendet werden kann.

9.1.2 clients/lib/SimpleConnect/SimpleConnect.h

```
#ifndef SIMPLE_CONNECT_H
#define SIMPLE_CONNECT_H

#include <ESP8266WiFi.h>
#include <time.h>
#include <Arduino.h>
#include <SimpleDebug.h>

// WiFi and MQTT client instances
extern WiFiClient espClient;

// Function declarations
void setupTime();
void setup_wifi(const char* ssid, const char* password);
String generateRandomID(int length);

#endif // SIMPLE_CONNECT_H
```

Die Datei SimpleConnect.h ist die Header-Datei für SimpleConnect.cpp und definiert die Schnittstelle für die in SimpleConnect.cpp implementierten Funktionen. Sie enthält die notwendigen Bibliotheken und Vorwärtsdeklarationen der Funktionen setupTime(), setup_wifi() und generateRandomID().

9.1.3 clients/lib/SimpleDebug/SimpleDebug.h

```
#ifndef SIMPLE_DEBUG_H
#define SIMPLE_DEBUG_H

#include <Arduino.h>

// Allow the debug flag to be defined externally
extern bool DEBUG_ENABLED;

// Empty function definitions for base cases
inline void print() {
    // Base case for recursion
}

// Empty function definitions to print a newline
inline void println() {
    Serial.println();
}

// Recursive function to print multiple arguments
template<typename T, typename... Args>
void print(T first, Args... args) {
    Serial.print(first);
    Serial.print(' ');
    print(args...);
}

// Recursive function to print multiple arguments with a newline
template<typename T, typename... Args>
void println(T first, Args... args) {
    Serial.print(first);
    Serial.print(' ');
    print(args...);
    Serial.println();
}

// Debug function that only prints if debugging is enabled
template<typename... Args>
void debug(Args... args) {
    if (DEBUG_ENABLED) {
        println(args...);
    }
}

// Info function that always prints
template<typename... Args>
void info(Args... args) {
    println(args...);
}
```

```
}
```

```
#endif // SIMPLE_DEBUG_H
```

Die Datei SimpleDebug.h bietet Funktionen für das Debugging und die Informationsausgabe. Sie definiert mehrere Funktionen wie debug() und info(), die zur Ausgabe von Debugging-Informationen auf der seriellen Konsole verwendet werden können. Die Funktion debug() gibt Informationen nur dann aus, wenn das Debugging aktiviert ist, während info() immer Informationen ausgibt. Diese Funktionen sind nützlich, um den Status und die Ausführung des Programms während der Entwicklung und Fehlersuche zu überwachen.

9.1.4 clients/lib/SimpleSleep/SimpleSleep.cpp

```
#include "SimpleSleep.h"

// Sleep until next check
void sleepUntil(const String& check_time) {
    // we expect the time to be in the format "HH:MM"
    int targetHour = check_time.substring(0, 2).toInt();
    int targetMinute = check_time.substring(3, 5).toInt();

    time_t now = time(nullptr);
    struct tm *now_tm = localtime(&now);

    // Setup target time: from config
    // add one day
    struct tm next_tm = *now_tm;

    next_tm.tm_hour = targetHour;
    next_tm.tm_min = targetMinute;
    next_tm.tm_sec = 0;
    next_tm.tm_mday += 1;

    time_t next_time = mktime(&next_tm);
    double secondsUntilTarget = difftime(next_time, now);

    // Format next time as HH:MM
    char formattedTime[6];
    strftime(formattedTime, sizeof(formattedTime), "%H:%M", &next_tm);
    info("Next check time: ", formattedTime, " - see you in ", secondsUntilTarget /
60, " minutes");

    // Set deep sleep in microseconds
    debug("Going to sleep now... (", secondsUntilTarget / 60, "minutes)");
    delay(1000);
    ESP.deepSleep((uint64_t)(secondsUntilTarget * 1000000));
}
```

Die Datei SimpleSleep.cpp enthält eine Funktion namens sleepUntil(), die das Gerät in den Tiefschlaf versetzt, bis eine bestimmte Uhrzeit erreicht ist. Die Funktion erwartet eine Uhrzeit im Format "HH:MM" als

Eingabe. Sie berechnet die Differenz zwischen der aktuellen Zeit und der Zielzeit und setzt das Gerät für diese Dauer in den Tiefschlaf. Vor dem Schlafengehen gibt die Funktion die nächste Überprüfungszeit und die Schlafdauer in Minuten aus. Diese Funktion ist nützlich für batteriebetriebene Geräte, die nur zu bestimmten Zeiten aktiv sein müssen, um Energie zu sparen.

9.1.5 clients/lib/SimpleSleep/SimpleSleep.h

```
#ifndef SIMPLE_SLEEP_H
#define SIMPLE_SLEEP_H

#include <time.h>
#include <Arduino.h>
#include <SimpleDebug.h>

void sleepUntil(const String& check_time);

#endif // SIMPLE_SLEEP_H
```

Die Datei SimpleSleep.h ist die Header-Datei für SimpleSleep.cpp und definiert die Schnittstelle für die in SimpleSleep.cpp implementierte Funktion. Sie enthält die notwendige Bibliothek und die Vorwärtsdeklaration der Funktion sleepUntil().

9.2 Sensor

9.2.1 clients/sensor/sensor.ino

```
#include <PubSubClient.h>
#include <SimpleConnect.h>
#include <SimpleSleep.h>
#include <SimpleDebug.h>

// #####
// # Global declarations

bool DEBUG_ENABLED = true;
const int threshold = 700;
const char *ssid = "Apple Network 785";
const char *password = "";
const char *mqtt_server = "192.168.1.80";
const char *prefix = "ttap_sensor_";
const int settings_count = 2;
int settings_set = 0;

// TRACKING
String client_name;
unsigned long startTime;
unsigned long gracePeriodStart;
bool checking;
bool publishNow;
bool needs_water;

// SETTINGS
int check_duration = 30;
String check_time = "03:00";

PubSubClient client(espClient);

// #####
// # Functions

// Reset the state of the Application
// (should also be run at the start to normalize behavior)
void reset(){
    debug("State resetting");
    client_name = prefix + generateRandomID(6);
    info("Client name:", client_name);

    // normalize state
    startTime = 0;
    gracePeriodStart = 0;
}
```

```

    checking = false;
    publishNow = false;
    needs_water = false;
    settings_set = 0;
}

// Callback function to handle incoming messages
void callback(char* topic, byte* payload, unsigned int length) {
    String message;
    for (unsigned int i = 0; i < length; i++) {
        message += (char)payload[i];
    }
    String topicStr = String(topic);

    debug("Message arrived [", topicStr, "]: ", message);

    // Update settings from MQTT
    if (topicStr.startsWith("settings/home/sensor")) {
        if (topicStr.endsWith("/check_time")) {
            check_time = message;
            debug("Updated check time:", check_time);
            settings_set++;
        }
        if (topicStr.endsWith("/check_duration")) {
            check_duration = message.toInt();
            debug("Updated check duration:", check_duration);
            settings_set++;
        }
    }
}

// Function to reconnect to the MQTT broker
void reconnect() {
    while (!client.connected()) {
        debug("Attempting MQTT connection...");
        if (client.connect(client_name.c_str())) {
            debug("MQTT connected");
            client.subscribe("settings/home/sensor/#");
        } else {
            debug("Failed to connect to MQTT rc=", client.state(), "try again in 1
second");
            delay(1000);
        }
    }
}

// #####
// # Main

```

```
void setup() {
  Serial.begin(9600);
  // Give time to boot
  while (!Serial) { }

  setup_wifi(ssid, password);
  setupTime();

  // Initialize LED as Output
  pinMode(LED_BUILTIN, OUTPUT);

  // Initialize MQTT
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);

  reset();
}

void loop() {
  long now = millis();

  if (checking) {
    // turn on the LED while we operate
    digitalWrite(LED_BUILTIN, LOW);
    // extract the sensor value from the A0 Channel
    // the value will normally be between 300 and 700
    // while 700 is close to dry
    int sensorValue = analogRead(A0);
    // compare to threshold
    needs_water = (sensorValue > threshold);

    // Check for 0.5 minutes (30000 milliseconds)
    if (now - startTime >= 30000) {
      checking = false;
      publishNow = true;
      debug("Check completed, needs water:", needs_water);
    }
    return;
  }

  // ensure MQTT is always connected
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  // do not continue if settings are not set (updated)
  if (settings_set < settings_count) {
    return;
  }
}
```



```

}

if (!publishNow && !checking){
    // update starttime to ensure the system can check for moisture over time
    // start when the settings have been received
    startTime = millis();
    checking = true;
    info("Starting check for moisture (30 seconds)");
    debug("Check duration: 30s, Started at:", startTime);
}

if (publishNow){
    if (gracePeriodStart == 0){
        // Publish the watering state
        String payload = needs_water ? "true" : "false";
        client.publish("home/sensor/watering_needed", payload.c_str());
        info("Published watering state:", payload);

        digitalWrite(LED_BUILTIN, HIGH);
        gracePeriodStart = now;
    }else if (now - gracePeriodStart > 1000){
        gracePeriodStart = 0;
        publishNow = false;

        // Sleep until next check time
        sleepUntil(check_time);
    }
}
}

```

Die Datei sensor.ino ist das Hauptprogramm für einen Feuchtigkeitssensor, der mit MQTT kommuniziert. Es initialisiert eine WiFi- und MQTT-Verbindung, liest den Feuchtigkeitssensor und veröffentlicht den Bewässerungsbedarf an einen MQTT-Broker. Die setup() Funktion initialisiert die Verbindungen und setzt die Systemzeit. Die loop() Funktion überwacht den Feuchtigkeitssensor und steuert die Veröffentlichung der Daten. Zusätzlich gibt es Funktionen wie reset(), callback() und reconnect(), die zum Zurücksetzen des Systemzustands, zum Verarbeiten eingehender MQTT-Nachrichten und zum Wiederverbinden mit dem MQTT-Broker verwendet werden. Die globalen Variablen speichern die Systemeinstellungen und den aktuellen Zustand.

9.3 Watering

9.3.1 clients/watering/watering.ino

```
#include <PubSubClient.h>
#include <SimpleConnect.h>
#include <SimpleSleep.h>
#include <SimpleDebug.h>

// #####
// # Global declarations

bool DEBUG_ENABLED = true;
const int threshold = 700;
const char *ssid = "Apple Network 785";
const char *password = "";
const char *mqtt_server = "192.168.1.80";
const char *prefix = "ttap_watering_";
const int settings_count = 2;
int settings_set = 0;

// TRACKING
String client_name;
unsigned long startTime;
bool needs_water;
bool requested;

// SETTINGS
int open_duration = 5 * 60;
String check_time = "03:00";

PubSubClient client(espClient);

// #####
// # Functions

// Reset the state of the Application
// (should also be run at the start to normalize behavior)
void reset(){
    debug("State resetting");
    client_name = prefix + generateRandomID(6);
    info("Client name:", client_name);

    // normalize state
    startTime = 0;
    needs_water = false;
    requested = false;
    settings_set = 0;
}
```

```

    digitalWrite(LED_BUILTIN, HIGH);
    digitalWrite(D1, LOW);
}

// Callback function to handle incoming messages
void callback(char* topic, byte* payload, unsigned int length) {
    String message;
    for (unsigned int i = 0; i < length; i++) {
        message += (char)payload[i];
    }
    String topicStr = String(topic);

    debug("Message arrived [", topicStr, "]: ", message);

    // Update settings from MQTT
    if (topicStr.startsWith("settings/home/watering")) {
        if (topicStr.endsWith("/check_time")) {
            check_time = message;
            debug("Updated check time:", check_time);
            settings_set++;
        }
        if (topicStr.endsWith("/open_duration")) {
            open_duration = message.toInt();
            debug("Updated open duration:", open_duration);
            settings_set++;
        }
    } else if (topicStr.startsWith("home/watering")) {
        if (topicStr.endsWith("/response")) {
            message.toLowerCase();
            if (message == "true") {
                needs_water = true;
            } else {
                needs_water = false;
            }
        }

        startTime = millis();
    }
}

// Function to reconnect to the MQTT broker
void reconnect() {
    while (!client.connected()) {
        debug("Attempting MQTT connection...");
        if (client.connect(client_name.c_str())) {
            debug("MQTT connected");
            client.subscribe("settings/home/watering/#");
            client.subscribe("home/watering/#/response");
        } else {

```

```

        debug("Failed to connect to MQTT rc=", client.state(), "try again in 1
second");
        delay(1000);
    }
}
}

// #####
// # Main

void setup() {
    Serial.begin(9600);
    // Give time to boot
    while (!Serial) { }

    setup_wifi(ssid, password);
    setupTime();

    // Initialize LED as Output
    pinMode(LED_BUILTIN, OUTPUT);
    // Pin connected to relais
    pinMode(D1, OUTPUT);

    // Initialize MQTT
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);

    reset();
}

void loop() {
    long now = millis();
    if (startTime > 0) {
        if (needs_water){
            digitalWrite(LED_BUILTIN, LOW);
            digitalWrite(D1, HIGH);

            if (now - startTime > open_duration * 1000){
                reset();
                sleepUntil(check_time);
            }
        }else{
            reset();
            sleepUntil(check_time);
        }
        return;
    }

    // ensure MQTT is always connected

```

```
if (!client.connected()) {  
    reconnect();  
}  
client.loop();  
  
if (settings_set < settings_count){  
    // wait for settings to be set  
    return;  
}  
  
if (!requested){  
    debug("Checking for water needs...");  
    // request watering state  
    client.publish("home/watering/watering_needed", "");  
    requested = true;  
}  
}
```

Die Datei watering.ino ist das Hauptprogramm für ein automatisches Bewässerungssystem. Es verwendet MQTT, um Einstellungen zu empfangen und den Bewässerungsstatus zu senden. Die setup() Funktion initialisiert die WiFi- und MQTT-Verbindungen, setzt die Systemzeit und konfiguriert die benötigten Pins. Die loop() Funktion überwacht den Bewässerungsstatus und steuert die Bewässerung entsprechend. Zusätzlich gibt es Funktionen wie reset(), callback() und reconnect(), die zum Zurücksetzen des Systemzustands, zum Verarbeiten eingehender MQTT-Nachrichten und zum Wiederverbinden mit dem MQTT-Broker verwendet werden. Die globalen Variablen speichern die Systemeinstellungen und den aktuellen Zustand.

10 Programmcode: Hub

Jeder Ordner enthält eine mod.rs Datei. Diese Datei verwaltet Importe und Module auf Ordner Level. Diese Dateien wurden nicht in diese Dokumentation aufgenommen da sie keine tatsächliche Logik enthalten. Der vollständige Programmcode kann im GitHub Repository gefunden werden.

<https://github.com/fokklz/sol-2-terra-tap>

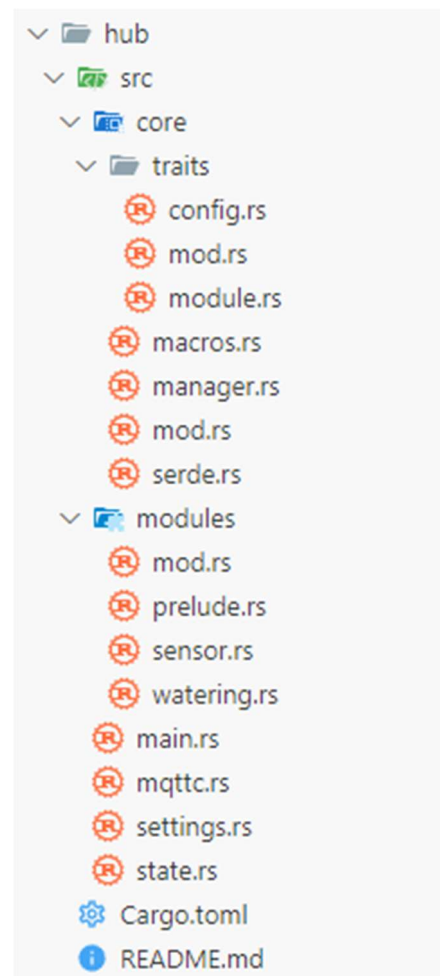
10.1 Hub

Programmcode des Hubs.

Der Hub-Ordner ist das Herzstück des Projekts und enthält den Code für die zentrale Kommunikationsstelle zwischen den Clients. Er ist verantwortlich für die Verwaltung der Verbindungen und der Nachrichten, die zwischen ihnen gesendet werden. Er behält den Zustand der Clients im Auge und ermöglicht das Anfordern von Zuständen.

Die Hauptdatei in diesem Ordner ist main.rs, die den Hub startet und konfiguriert. Darüber hinaus gibt es eine Reihe von Unterordnern und Dateien, die spezifische Aspekte des Hubs behandeln:

- Der core Ordner enthält den Code für die grundlegende Funktionalität des Hubs, einschliesslich des Modulmanagers und der Serde-Funktionen für die Serialisierung und Deserialisierung von Daten.
- Der modules Ordner enthält die Endpunktdefinitionen für die Module, die mit dem Hub kommunizieren sollen. Jedes Modul hat seine eigene Datei, z.B. watering.rs für das Bewässerungsmodul und sensor.rs für das Sensormodul.
- Die mqttc.rs Datei enthält den Code für den MQTT-Client, der für die Kommunikation mit dem MQTT-Broker verwendet wird.
- Die settings.rs und state.rs Dateien enthalten den Code für die Verwaltung der Einstellungen und des Zustands des Hubs.



Die Beschreibungen für die Dateien wurden mithilfe von GPT grundlegend entworfen, von Fokko Vos kontrolliert und überarbeitet.

10.1.1 src/main.rs

```
use modules::{SensorModule, WateringModule};
use once_cell::sync::{Lazy, OnceCell};
use std::{
    io,
    process::{Child, Command, Stdio},
};
use tokio::{
    signal,
    sync::{broadcast, Mutex},
};
use traits::ConfigFile;

mod core;
pub use core::*;

mod modules;
mod mqttc;
mod settings;
mod state;

pub use settings::Settings;
pub use state::State;

// Re-export the client for easy access
pub use mqttc::CLIENT;

/// Check if the program is running in release mode and disallow it.
/// Currently only running from source is intended.
fn check_release_mode() {
    if cfg!(debug_assertions) {
        return;
    }
    eprintln!("Error: Running in release mode is not supported.");
    std::process::exit(1);
}

/// Spawn the MQTT broker as a child process
fn spawn_broker() -> io::Result<Child> {
    Command::new("cargo")
        .arg("run")
        .arg("--bin")
        .arg("mqtttd")
        .stdout(Stdio::inherit())
        .stderr(Stdio::inherit())
        .spawn()
}
```

```
// Global handle to the MQTT broker
static BROKER: OnceCell<Mutex<Child>> = OnceCell::new();
// Global handle to the settings
static SETTINGS: OnceCell<Mutex<Settings>> = OnceCell::new();
// Global handle to the state
static STATE: OnceCell<Mutex<State>> = OnceCell::new();
// Global handle to the module manager
static MODULE_MANAGER: Lazy<Mutex<ModuleManager>> = Lazy::new(||
Mutex::new(ModuleManager::new()));

#[tokio::main]
async fn main() {
    std::env::set_var("RUST_LOG", "debug");
    check_release_mode();

    tracing_subscriber::fmt::init();

    // Shutdown channel to ensure graceful shutdown
    let (shutdown_tx, shutdown_rx) = broadcast::channel::<()>(1);

    // Spin up a task to listen for Ctrl+C
    let ctrl_c_task = tokio::spawn(async move {
        signal::ctrl_c().await.expect("Failed to listen for Ctrl+C");
        // Send shutdown signal
        let _ = shutdown_tx.send(());
    });

    // #####
    // # Code above should not be modified, as it ensures the proper operation of
the program.
    // # It also guarantees that the program will log anything it does

    // create settings and state handles
    let settings = Settings::load();
    let state = State::load();
    let _ = STATE.set(Mutex::new(state)).unwrap();
    let _ = SETTINGS.set(Mutex::new(settings.clone())).unwrap();

    // Register the modules
    let mut manager = MODULE_MANAGER.lock().await;
    manager.register_module(SensorModule::from(&settings));
    manager.register_module(WateringModule::from(&settings));
    // ensure the manager is available for the client
    drop(manager);

    tracing::info!("TerraTap running... Press Ctrl+C to exit.");
    tracing::info!("Starting MQTT broker... This may take a few seconds.");
    let broker = spawn_broker().expect("Failed to spawn broker");
    let _ = BROKER.set(Mutex::new(broker)).unwrap();
}
```



```

let client_task = mqttc::run(shutdown_rx);

// Wait for either Ctrl+C or the client task to finish
// The client task should not finish
// TODO: auto-restart mechanism instead of just exiting
loop {
    tokio::select! {
        _ = ctrl_c_task => {
            break;
        }
        _ = client_task.await => {
            break;
        }
    }
}

// kill the broker to be sure all tasks are cleaned up
let _ = BROKER.get().unwrap().lock().await.kill();
// Save the state and settings before exiting
STATE.get().unwrap().lock().await.save();
SETTINGS.get().unwrap().lock().await.save();

tracing::info!("Thank you for using TerraTap! Until next time!");
}

```

Die Datei `main.rs` ist die Hauptdatei des Programms und enthält den Einstiegspunkt der Anwendung. Sie beginnt mit der Definition einiger Funktionen, wie `check_release_mode()`, die überprüft, ob das Programm im Release-Modus läuft und dies verbietet, und `spawn_broker()`, die den MQTT-Broker als Child-Prozess startet. Der Hauptteil des Programms wird in der asynchronen Funktion `main()` ausgeführt. Hier werden verschiedene Einstellungen und Zustände initialisiert, Module registriert und der MQTT-Broker gestartet. Das Programm läuft in einer Schleife, bis es durch das Drücken von Ctrl+C oder das Beenden der Client-Aufgabe gestoppt wird. Vor dem Beenden werden der Zustand und die Einstellungen gespeichert. Die Datei enthält auch die Definitionen einiger globaler Variablen, die im gesamten Programm verwendet werden.

10.1.2 src/mqttrc.rs

```
use std::time::Duration;

use once_cell::sync::OnceCell;
use rumqttrc::{ AsyncClient, MqttOptions };
use tokio::sync::{ broadcast::Receiver, Mutex };
use tracing::span;

pub static CLIENT: OnceCell<Mutex<AsyncClient>> = OnceCell::new();

pub async fn run(mut shutdown: Receiver<()>) -> tokio::task::JoinHandle<()> {
    let broker_span = span!(tracing::Level::INFO, "mqtt-client");
    let _ = broker_span.enter();

    let mut mqtt_options = MqttOptions::new("Hub", "127.0.0.1", 1883);
    mqtt_options.set_keep_alive(Duration::from_secs(5));

    let (client, mut eventloop) = AsyncClient::new(mqtt_options, 10);
    CLIENT.set(Mutex::new(client.clone())).unwrap();
    tracing::info!("Client created and stored in global static variable");

    tokio::spawn(async move {
        loop {
            tokio::select! {
                result = eventloop.poll() => {
                    if let Ok(notification) = result {
                        //dbg!("Received = {:#?}", notification.clone());
                        match notification {
                            rumqttrc::Event::Incoming(incoming) => {
                                match incoming {
                                    rumqttrc::Incoming::Publish(publish) => {
                                        let topic = publish.topic;
                                        let payload = publish.payload;

                                        // acquire the module manager and handle
                                        the message

                                        let manager =
                                        crate::MODULE_MANAGER.lock().await;

                                        manager.handle_message(&topic,
                                        std::str::from_utf8(&payload).unwrap()).await;
                                        // free the manager
                                        drop(manager);
                                    }
                                    rumqttrc::Incoming::ConnAck(_) => {
                                        // Initialize the modules once the
                                        connection is acknowledged

                                        let manager =
                                        crate::MODULE_MANAGER.lock().await;

```

irgnored

```
incoming);
```

TerraTap Dokumentation

10.1.3 src/settings.rs

```
use chrono::NaiveTime;
use serde::{Deserialize, Serialize};

use crate::traits::ConfigFile;

#[derive(Clone, Debug, Serialize, Deserialize)]
pub struct Settings {
    #[serde(
        serialize_with = "crate::serde::serialize_naive_time",
        deserialize_with = "crate::serde::deserialize_naive_time"
    )]
    pub check_time: NaiveTime,
    pub check_duration: u64,
    pub open_duration: u64,
}

impl Default for Settings {
    fn default() -> Self {
        Self {
            // Default time is 3:00 AM
            check_time: NaiveTime::from_hms_opt(3, 0, 0).unwrap(),
            // Default duration is 30 seconds
            check_duration: 30,
            // Default duration is 5 minutes
            open_duration: 5 * 60,
        }
    }
}

impl ConfigFile<&'static str> for Settings {
    const PATH: &'static str = "settings.json";
    type Config = Self;
}
```

Die Datei settings.rs definiert die Struktur Settings, die die Konfigurationseinstellungen der Anwendung enthält. Diese Einstellungen umfassen check_time, die Zeit, zu der eine Überprüfung durchgeführt wird, check_duration, die Dauer der Überprüfung in Sekunden, und open_duration, die Dauer, für die etwas geöffnet bleibt, ebenfalls in Sekunden. Die Struktur implementiert das Default-Trait, das Standardwerte für diese Einstellungen bereitstellt. Sie implementiert auch das ConfigFile-Trait, das es ermöglicht, die Einstellungen aus einer JSON-Datei zu laden und in diese zu speichern. Der Pfad zu dieser Datei ist settings.json. Die Datei verwendet die serde-Bibliothek für die Serialisierung und Deserialisierung der Einstellungen und die chrono-Bibliothek für die Zeitverwaltung.

10.1.4 src/state.rs

```
use serde::{ Deserialize, Serialize };

use crate::traits::ConfigFile;

#[derive(Debug, Deserialize, Serialize)]
pub struct State {
    pub watering_needed: bool,
}

impl Default for State {
    fn default() -> Self {
        Self {
            watering_needed: false,
        }
    }
}

impl ConfigFile<&'static str> for State {
    const PATH: &'static str = "state.json";
    type Config = Self;
}
```

Die Datei state.rs definiert die Struktur State, die den aktuellen Zustand der Anwendung speichert. Dieser Zustand umfasst nur eine einzige Variable, watering_needed, die angibt, ob eine Bewässerung benötigt wird. Die Struktur implementiert das Default-Trait, das einen Standardzustand bereitstellt, in dem keine Bewässerung benötigt wird. Sie implementiert auch das ConfigFile-Trait, das es ermöglicht, den Zustand aus einer JSON-Datei zu laden und in diese zu speichern. Der Pfad zu dieser Datei ist state.json. Die Datei verwendet die serde-Bibliothek für die Serialisierung und Deserialisierung des Zustands.

10.1.5 src/core/serde.rs

```
use std::fmt;

use chrono::{NaiveTime, Timelike};
use serde::{
    de::{self, Visitor},
    Deserializer, Serializer,
};

// Custom function to serialize NaiveTime to "HH:MM" format
pub fn serialize_naive_time<S>(time: &NaiveTime, serializer: S) -> Result<S::Ok, S::Error>
where
    S: Serializer,
{
    // Format the NaiveTime as "HH:MM"
    let formatted = format!("{:02}:{:02}", time.hour(), time.minute());
    // Serialize the formatted string
    serializer.serialize_str(&formatted)
}

// Custom function to deserialize "HH:MM" format to NaiveTime
pub fn deserialize_naive_time<'de, D>(deserializer: D) -> Result<NaiveTime, D::Error>
where
    D: Deserializer<'de>,
{
    // Visitor to help with deserialization
    struct NaiveTimeVisitor;

    impl<'de> Visitor<'de> for NaiveTimeVisitor {
        type Value = NaiveTime;

        fn expecting(&self, formatter: &mut fmt::Formatter) -> fmt::Result {
            formatter.write_str("a string in the format HH:MM")
        }

        fn visit_str<E>(self, value: &str) -> Result<NaiveTime, E>
        where
            E: de::Error,
        {
            NaiveTime::parse_from_str(value, "%H:%M").map_err(de::Error::custom)
        }
    }

    // Use the visitor to deserialize the string
    deserializer.deserialize_str(NaiveTimeVisitor)
}
```

```
// Tests for the custom serialization and deserialization functions
#[cfg(test)]
mod tests {
    use serde::{Deserialize, Serialize};

    use super::*;

    // Struct to test serialization and deserialization
    // Same as the actual usage in something like settings should be
    #[derive(Serialize, Deserialize)]
    struct Testing {
        #[serde(
            serialize_with = "serialize_naive_time",
            deserialize_with = "deserialize_naive_time"
        )]
        time: NaiveTime,
    }

    #[test]
    fn test_serialize_naive_time() {
        let time = NaiveTime::from_hms_opt(12, 34, 56).unwrap();
        let testing = Testing { time };
        let serialized = serde_json::to_string(&testing).unwrap();
        assert_eq!(serialized, r#"{"time":"12:34"}"#);
    }

    #[test]
    fn test_deserialize_naive_time() {
        let testing =
            serde_json::from_str::<Testing>(r#"{"time":"12:34"}"#).unwrap();
        assert_eq!(testing.time, NaiveTime::from_hms_opt(12, 34, 0).unwrap());
    }
}
```

Die Datei `serde.rs` enthält benutzerdefinierte Funktionen zur Serialisierung und Deserialisierung von `NaiveTime`-Objekten in das "HH:MM"-Format. Die Funktion `serialize_naive_time()` nimmt ein `NaiveTime`-Objekt und einen Serializer und gibt das serialisierte Ergebnis zurück. Sie formatiert das `NaiveTime`-Objekt als "HH:MM"-Zeichenkette und serialisiert diese Zeichenkette. Die Funktion `deserialize_naive_time()` nimmt einen Deserializer und gibt ein `NaiveTime`-Objekt zurück. Sie definiert einen Visitor, der eine Zeichenkette im "HH:MM"-Format in ein `NaiveTime`-Objekt umwandelt. Die Datei enthält auch Tests für diese beiden Funktionen. Der Test `test_serialize_naive_time()` überprüft, ob die Serialisierung korrekt funktioniert, und der Test `test_deserialize_naive_time()` überprüft, ob die Deserialisierung korrekt funktioniert. Diese Funktionen werden verwendet, um die `NaiveTime`-Felder in den Einstellungen und im Zustand zu serialisieren und zu deserialisieren.

10.1.6 src/core/manager.rs

```
use crate::{topic, ClientModule };
use std::collections::HashMap;

/// Core manager for handling modules and the settings for modules
/// All modules should be registered with the manager
/// The handle_message function should be called from the main loop
pub struct ModuleManager {
    modules: Vec<Box<dyn ClientModule>>,
    configs: HashMap<String, String>,
}

impl ModuleManager {
    /// Create a new ModuleManager
    pub fn new() -> Self {
        tracing::trace!("ModuleManager created");

        Self {
            modules: Vec::new(),
            configs: HashMap::new(),
        }
    }

    /// Register a module with the manager
    /// This will add the settings to the settings map
    pub fn register_module(&mut self, module: impl ClientModule + 'static) {
        let name = module.name();
        // prefix the setting with the module topic
        let settings = module
            .settings()
            .iter()
            .map(|(k, v)| (format!("{}/{}", module.topic(), k), v.clone()))
            .collect::<HashMap<String, String>>();

        self.configs.extend(settings);
        self.modules.push(Box::new(module));
        tracing::debug!("Registered module '{}'", name);
    }

    /// Handle a message from the MQTT broker
    /// Will filter the modules based on the topic and call the handle function
    of each matching module
    pub async fn handle_message(&self, topic: &str, payload: &str) {
        for ele in self.modules.iter().filter(|ele| topic.contains(&ele.topic()))
        {
            tracing::debug!("Handling message for module '{}' on topic '{}'",
            ele.name(), topic);
            ele.handle(topic, payload).await;
        }
    }
}
```



```

    }
}

/// Initialize the modules and settings
/// Subscribes to the topics of the modules and publishes the settings as
retained messages
pub fn initialize(&self, client: &rumqttc::AsyncClient) {
    for (topic, value) in &self.configs {
        let new_topic = format!("settings/{}", topic);
        let mut payload = value.to_string();
        if payload.is_empty() {
            continue;
        }
        // clean up the payload for strings
        payload = payload.replace("\\", "");

        // Publish the setting to the MQTT broker
        let res = client.try_publish(
            new_topic,
            rumqttc::QoS::ExactlyOnce,
            true,
            payload.as_bytes()
        );

        if res.is_ok() {
            tracing::debug!(
                "Published settings for '{}' retain = true, value = '{}'",
                topic,
                value
            );
        } else {
            tracing::error!(
                "Failed to publish settings for '{}' retain = true, value =
'{}'",
                topic,
                value
            );
        }
    }

    for module in &self.modules {
        let res = client.try_subscribe(topic!(module.topic(), "#"),
            rumqttc::QoS::ExactlyOnce);

        if res.is_ok() {
            tracing::debug!("Subscribed to '{}'", module.topic());
        } else {
            tracing::error!("Failed to subscribe to '{}'", module.topic());
        }
    }
}

```

```

    }
  }
}

/// Tests for the ModuleManager
/// The Initialize function is not tested as it requires a rumqttc::AsyncClient
/// The handle_message function is not tested as it returns nothing
#[cfg(test)]
mod tests {
    use super::*;

    #[derive(Debug, Default, serde::Deserialize, serde::Serialize)]
    struct TestModule {
        config_data: String,
    }

    #[async_trait::async_trait]
    impl ClientModule for TestModule {
        fn topic(&self) -> String {
            "test/topic".to_string()
        }

        async fn handle(&self, _topic: &str, _payload: &str) {}

        fn settings(&self) -> HashMap<String, String> {
            let mut settings = HashMap::new();

            let ser = serde_json::to_string(&self).unwrap();
            let deser = serde_json
                ::to_value(serde_json::from_str::<TestModule>(&ser).unwrap())
                .unwrap();

            settings.insert("config_data".to_string(),
deser["config_data"].to_string());

            settings
        }
    }

    #[test]
    fn test_module_manager_new() {
        let manager = ModuleManager::new();
        assert_eq!(manager.modules.len(), 0);
        assert_eq!(manager.configs.len(), 0);
    }

    #[test]
    fn test_register_module() {
        let mut manager = ModuleManager::new();

```

```

    let module = TestModule::default();
    manager.register_module(module);

    assert_eq!(manager.modules.len(), 1);
    assert_eq!(manager.configs.len(), 1);
}

#[test]
fn test_setting_prefix_correctly(){
    let mut manager = ModuleManager::new();
    let module = TestModule::default();
    manager.register_module(module);

    assert!(manager.configs.contains_key("test/topic/config_data"));
}
}

```

Die Datei `manager.rs` definiert die Struktur `ModuleManager`, die für die Verwaltung von Modulen und deren Einstellungen zuständig ist. Sie speichert die Module in einem Vektor und die Einstellungen in einer `HashMap`. Die Struktur bietet Methoden zum Registrieren von Modulen, zum Behandeln von Nachrichten von einem MQTT-Broker und zum Initialisieren der Module und Einstellungen. Beim Registrieren eines Moduls werden die Einstellungen des Moduls in die `HashMap` aufgenommen und das Modul selbst wird dem Vektor hinzugefügt. Beim Behandeln einer Nachricht werden alle Module, deren Thema in dem Thema der Nachricht enthalten ist, gefiltert und ihre Handle-Funktion wird aufgerufen. Beim Initialisieren werden die Einstellungen als zurückgehaltene Nachrichten an den MQTT-Broker gesendet und es wird ein Abonnement für die Themen der Module erstellt. Die Datei enthält auch Tests für die Methoden `new`, `register_module` und `handle_message`. Der Test `test_module_manager_new()` überprüft, ob ein neuer `ModuleManager` korrekt erstellt wird, der Test `test_register_module()` überprüft, ob ein Modul korrekt registriert wird, und der Test `test_setting_prefix_correctly()` überprüft, ob die Einstellungen korrekt mit dem Thema des Moduls vorangestellt werden.

10.1.7 src/core/macros.rs

```
/// A macro to generate a topic string from a prefix and an ending.
#[macro_export]
macro_rules! topic {
    ($prefix:expr, $ending:expr) => {
        format!("{}/{}", $prefix, $ending)
    };
}
```

Die Datei macros.rs definiert ein Makro namens `topic!`, das eine Zeichenkette aus einem Präfix und einem Ende generiert. Das Makro nimmt zwei Ausdrücke, `$prefix` und `$ending`, und gibt eine formatierte Zeichenkette zurück, die das Präfix, einen Schrägstrich und das Ende enthält. Dieses Makro kann verwendet werden, um MQTT-Themen zu generieren, indem ein Präfix, das den Kontext des Themas angibt, und ein Ende, das den spezifischen Teil des Themas angibt, bereitgestellt werden. Da das Makro mit `#[macro_export]` annotiert ist, kann es von anderen Modulen in der gleichen Crate verwendet werden.

10.1.8 src/core/traits/config.rs

```
use std::path::Path;

/// A trait for loading and saving configuration files
pub trait ConfigFile<P: AsRef<Path>> {
    type Config: serde::de::DeserializeOwned + Default + serde::Serialize;

    const PATH: P;

    /// Load settings from the file or return default settings
    fn load() -> Self::Config {
        std::fs::read_to_string(Self::PATH.as_ref())
            .map(|s| serde_json::from_str::<Self::Config>(&s).unwrap())
            .unwrap_or_default()
    }

    /// Save settings to the file
    fn save(&self)
    where
        Self: serde::Serialize,
    {
        let s = serde_json::to_string(self).unwrap();
        std::fs::write(Self::PATH.as_ref(), s).unwrap();
    }
}

/// Tests for the ConfigFile trait above
/// Creates a TestConfig struct that implements the ConfigFile trait
/// and tests the load and save methods
/// The test file is created in the current directory and removed after the test
#[cfg(test)]
mod tests {
```

```

use super::*;

#[derive(Debug, Default, serde::Deserialize, serde::Serialize)]
struct TestConfig {
    pub test: String,
}

impl ConfigFile<&'static str> for TestConfig {
    type Config = TestConfig;

    const PATH: &'static str = "test.json";
}

#[test]
fn test_config_file_load() {
    let config = TestConfig::load();
    assert_eq!(config.test, "");
}

/// This will also test the load from a file
#[test]
fn test_config_file_save() {
    let mut config = TestConfig::default();
    assert_eq!(config.test, "");

    config.test = "test".to_string();
    config.save();

    let config_loaded = TestConfig::load();
    assert_eq!(config_loaded.test, "test");

    // Clean up test file
    std::fs::remove_file("test.json").unwrap();
}
}

```

Die Datei config.rs definiert ein Trait namens ConfigFile, das für das Laden und Speichern von Konfigurationsdateien verwendet wird. Es definiert zwei Methoden, load() und save(), und einen assoziierten Typ Config, der die Konfiguration repräsentiert. Die Methode load() lädt die Einstellungen aus der Datei oder gibt die Standardeinstellungen zurück, und die Methode save() speichert die Einstellungen in der Datei. Die Datei enthält auch Tests für das Trait. Der Test test_config_file_load() überprüft, ob die Methode load() korrekt funktioniert, und der Test test_config_file_save() überprüft, ob die Methode save() und load() korrekt funktioniert. Die Tests löschen erstellte Dateien wieder.

10.1.9 src/core/traits/module.rs

```
use std::collections::HashMap;

/// A trait for modules that can be added to the Hub
#[async_trait::async_trait]
pub trait ClientModule: Send + Sync {
    /// The name of the module (last part of the topic by default)
    fn name(&self) -> String {
        self.topic()
            .clone()
            .split('/')
            .last()
            .unwrap_or_default()
            .to_string()
    }

    /// The topic the module is interested in
    fn topic(&self) -> String;

    /// The handlers for the module
    async fn handle(&self, topic: &str, payload: &str);

    /// The settings for the module
    fn settings(&self) -> HashMap<String, String>;
}

/// Test the ClientModule trait with a simple module
#[cfg(test)]
mod simple_module_tests {
    use super::*;

    #[derive(Debug, Default, serde::Deserialize, serde::Serialize)]
    struct SimpleTestModule;

    #[async_trait::async_trait]
    impl ClientModule for SimpleTestModule {
        fn topic(&self) -> String {
            "test/topic".to_string()
        }

        async fn handle(&self, _topic: &str, _payload: &str) {}

        fn settings(&self) -> HashMap<String, String> {
            HashMap::new()
        }
    }

    #[test]
```

```

fn test_simple_client_module() {
    let module = SimpleTestModule;
    assert_eq!(module.name(), "topic");
    assert_eq!(module.topic(), "test/topic");
    assert_eq!(module.settings().len(), 0);
}
}

/// Test the ClientModule trait with a module that has settings
#[cfg(test)]
mod module_tests {
    use super::*;

    #[derive(Debug, Default, serde::Deserialize, serde::Serialize)]
    struct TestModule {
        config_data: String,
    }

    #[async_trait::async_trait]
    impl ClientModule for TestModule {
        fn topic(&self) -> String {
            "test/topic".to_string()
        }

        async fn handle(&self, _topic: &str, _payload: &str) {}

        fn settings(&self) -> HashMap<String, String> {
            let mut settings = HashMap::new();

            let ser = serde_json::to_string(&self).unwrap();
            let deser =
                serde_json::to_value(serde_json::from_str::<TestModule>(&ser).unw
rap()).unwrap();

            for (key, value) in deser.as_object().unwrap() {
                settings.insert(key.clone(), value.to_string());
            }
            settings
        }
    }

    #[test]
    fn test_client_module() {
        let module = TestModule::default();
        assert_eq!(module.name(), "topic");
        assert_eq!(module.topic(), "test/topic");
        assert_eq!(module.settings().len(), 1);
    }
}

```

Die Datei `module.rs` definiert ein Trait namens `ClientModule`, das für Module verwendet wird, die zum Hub hinzugefügt werden können. Es definiert vier Methoden, `name()`, `topic()`, `handle()` und `settings()`. Die Methode `name()` gibt den Namen des Moduls zurück, die Methode `topic()` gibt das Thema zurück, an dem das Modul interessiert ist, die Methode `handle()` behandelt eine Nachricht mit einem bestimmten Thema und Inhalt, und die Methode `settings()` gibt die Einstellungen des Moduls zurück. Die Datei enthält auch Tests für das Trait mit zwei Modulen, `SimpleTestModule` und `TestModule`. Der Test `test_simple_client_module()` überprüft, ob das `SimpleTestModule` korrekt funktioniert, und der Test `test_client_module()` überprüft, ob das `TestModule` korrekt funktioniert.

10.1.10 `src/modules/prelude.rs`

```
pub use crate::{ ClientModule, Settings };
pub use serde::{ Deserialize, Serialize };

pub use crate::topic;
```

Die Datei `prelude.rs` ist eine Sammlung von häufig verwendeten Importen, die in anderen Modulen der Crate verwendet werden können. Sie enthält Importe für die Traits `ClientModule` und `Settings` aus der eigenen Crate, die Traits `Deserialize` und `Serialize` aus der Crate `serde` für die Serialisierung und Deserialisierung von Daten, und das Makro `topic` aus der eigenen Crate. Durch das Importieren dieser Elemente in der Prelude können sie in anderen Modulen der Crate ohne vollständigen Pfad verwendet werden.

10.1.11 src/modules/sensor.rs

```
use std::collections::HashMap;

use chrono::{Duration, NaiveTime };

use super::prelude::*;

#[derive(Serialize, Deserialize)]
pub struct SensorModule {
    #[serde(
        serialize_with = "crate::serde::serialize_naive_time",
        deserialize_with = "crate::serde::deserialize_naive_time"
    )]
    pub check_time: NaiveTime,
    pub check_duration: u64,
}

impl Default for SensorModule {
    fn default() -> Self {
        Self::from(&Settings::default())
    }
}

impl From<&Settings> for SensorModule {
    fn from(settings: &Settings) -> Self {
        Self {
            // remove 5 minutes from the check time for the sensor
            check_time: settings.check_time - Duration::minutes(5),
            check_duration: settings.check_duration,
        }
    }
}

#[async_trait::async_trait]
impl ClientModule for SensorModule {
    fn topic(&self) -> String {
        format!("{}/{}", super::PREFIX, "sensor")
    }

    async fn handle(&self, topic: &str, payload: &str) {
        match topic {
            t if t == topic!(self.topic(), "watering_needed") => {
                if payload.to_ascii_lowercase() == "true" {
                    let mut state_mut = crate::STATE.get().unwrap().lock().await;
                    state_mut.watering_needed = true;
                    tracing::info!("Watering needed: {}",
state_mut.watering_needed);
                } else {

```

```

        tracing::trace!(
            "Received message on topic '{}{}' with payload '{}'",
            topic,
            payload
        );
    }
}
_ => {}
}
}

fn settings(&self) -> HashMap<String, String> {
    let mut settings = HashMap::new();

    let ser = serde_json::to_string(&self).unwrap();
    let deser = serde_json
        ::to_value(serde_json::from_str::<SensorModule>(&ser).unwrap())
        .unwrap();

    for (key, value) in deser.as_object().unwrap() {
        settings.insert(key.clone(), value.to_string());
    }

    settings
}
}

```

Die Datei `sensor.rs` definiert eine Struktur namens `SensorModule`, die das Trait `ClientModule` implementiert. Die Struktur hat zwei Felder, `check_time` und `check_duration`, die die Zeit und Dauer der Überprüfung durch den Sensor darstellen. Die Struktur implementiert die Traits `Default` und `From<&Settings>` für die Erstellung von Instanzen aus den Standardeinstellungen oder spezifischen Einstellungen. Die Implementierung des Traits `ClientModule` definiert die Methoden `topic()`, `handle()` und `settings()` zur Interaktion mit dem Modul.

10.1.12 src/modules/watering.rs

```
use std::collections::HashMap;

use super::prelude::*;

use chrono::NaiveTime;
use rumqttn::QoS;

#[derive(Serialize, Deserialize)]
pub struct WateringModule {
    #[serde(
        serialize_with = "crate::serde::serialize_naive_time",
        deserialize_with = "crate::serde::deserialize_naive_time"
    )]
    pub check_time: NaiveTime,
    pub open_duration: u64,
}

impl Default for WateringModule {
    fn default() -> Self {
        Self::from(&Settings::default())
    }
}

impl From<&Settings> for WateringModule {
    fn from(settings: &Settings) -> Self {
        Self {
            check_time: settings.check_time,
            open_duration: settings.open_duration,
        }
    }
}

#[async_trait::async_trait]
impl ClientModule for WateringModule {
    fn topic(&self) -> String {
        format!("{}/{}", super::PREFIX, "watering")
    }

    fn settings(&self) -> HashMap<String, String> {
        let mut settings = HashMap::new();

        let ser = serde_json::to_string(&self).unwrap();
        let deser = serde_json
            ::to_value(serde_json::from_str::<WateringModule>(&ser).unwrap())
            .unwrap();

        for (key, value) in deser.as_object().unwrap() {
```

```

        settings.insert(key.clone(), value.to_string());
    }

    settings
}

async fn handle(&self, topic: &str, _payload: &str) {
    match topic {
        t if t == topic!(self.topic(), "watering_needed") => {
            let client = crate::CLIENT.get().unwrap().lock().await;
            let mut state = crate::STATE.get().unwrap().lock().await;
            // Publish the watering needed state to the MQTT broker so the
client can read it
            // TODO: Tie this to a client to allow for multiple watering
modules

            let res = client.try_publish(
                topic!(self.topic(), "watering_needed/response"),
                QoS::ExactlyOnce,
                false,
                state.watering_needed.clone().to_string().as_bytes()
            );

            // If the publish was successful, reset the watering needed state
            if res.is_ok() {
                state.watering_needed = false;
                tracing::trace!("Watering needed reset");
            }
        }
        _ => {}
    }
}
}

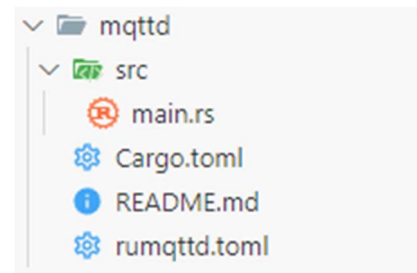
```

Die Datei watering.rs definiert eine Struktur namens WateringModule, die das Trait ClientModule implementiert. Die Struktur hat zwei Felder, check_time und open_duration, die die Zeit und Dauer der Bewässerung darstellen. Die Struktur implementiert die Traits Default und From<&Settings> für die Erstellung von Instanzen aus den Standardeinstellungen oder spezifischen Einstellungen. Die Implementierung des Traits ClientModule definiert die Methoden topic(), handle() und settings() zur Interaktion mit dem Modul. Die Methode handle() veröffentlicht den Bewässerungsbedarf an den MQTT-Broker und setzt den Bewässerungsbedarf zurück, wenn die Veröffentlichung erfolgreich war.

10.2 MQTTD

Programmcode des MQTTD.

Der MQTTD-Ordner enthält den Code für den MQTT-Broker, der für die Kommunikation zwischen den Clients verantwortlich ist. Der Broker wird vom HUB gestartet und ist ein wesentlicher Bestandteil des Systems, da er die Nachrichtenübermittlung zwischen den verschiedenen Komponenten ermöglicht.



Die Hauptdatei in diesem Ordner ist `main.rs`, die den Broker startet und konfiguriert. Darüber hinaus gibt es eine `rumqtttd.toml` Datei, die die Konfiguration des Brokers enthält.

Die Beschreibungen für die Dateien wurden mithilfe von GPT grundlegend entworfen, von Fokko Vos kontrolliert und überarbeitet.

10.2.1 src/main

```
use rumqtttd::{ Broker, Config };
use tracing::span;
use tracing_subscriber::{ EnvFilter, FmtSubscriber };

#[tokio::main]
async fn main() {
    // Use the RUST_LOG environment variable to set the log level
    let filter = EnvFilter::new(std::env::var("RUST_LOG").unwrap_or_else(|_|
"info".to_string()));
    let subscriber = FmtSubscriber::builder()
        .with_env_filter(filter)
        .with_level(true)
        .with_target(true)
        .with_thread_ids(false)
        .with_thread_names(false)
        .finish();

    tracing::subscriber::set_global_default(subscriber).expect("setting default
subscriber failed");

    // Create a span for the broker and enter it
    let broker_span = span!(tracing::Level::INFO, "mqtt-server");
    let _ = broker_span.enter();

    // Configure the broker using the rumqtttd configuration file
    // TODO: this should be handled differently to support for various
configuration sources
    // and to allow for more flexible configuration (not running from
source)
    let raw_mqtt_config = config::Config
        ::builder()
        .add_source(config::File::with_name("mqtttd/rumqtttd.toml"))
```

```

        .build()
        .unwrap();
let mqtt_config: Config = raw_mqtt_config.try_deserialize().unwrap();

// Create the final instance of the broker and start it
let mut broker = Broker::new(mqtt_config);
tokio
    ::spawn(async move {
        broker.start().unwrap();
    }).await
    .unwrap();
}

```

Die Datei main.rs ist der Einstiegspunkt für das Programm. Sie konfiguriert und startet den MQTT-Broker. Zunächst wird ein Logging-Subscriber mit Umgebungsvariablen konfiguriert und global gesetzt. Danach wird ein Span für den Broker erstellt und betreten. Anschliessend wird die Konfiguration des Brokers mit einer TOML-Datei geladen und deserialisiert. Schliesslich wird eine Instanz des Brokers erstellt und gestartet.

10.2.2 rumqttd.toml

```

id = 0

[router]
id = 0
max_connections = 10010
max_outgoing_packet_count = 200
max_segment_size = 104857600
max_segment_count = 10

[v4.1]
name = "v4-1"
listen = "0.0.0.0:1883"
next_connection_delay_ms = 1

[v4.1.connections]
connection_timeout_ms = 60000
max_payload_size = 20480
max_inflight_count = 100
dynamic_filters = true

# Configuration below is needed for MQTTX to work with the broker
# MQTTX is a cross-platform MQTT desktop client that supports MQTT 5.0 and 3.1.1
# https://mqttx.app/

[v5.1]
name = "v5-1"
listen = "0.0.0.0:1884"

```

```
next_connection_delay_ms = 1
```

```
[v5.1.connections]
```

```
connection_timeout_ms = 60000
```

```
max_payload_size = 20480
```

```
max_inflight_count = 100
```

Die Datei `rumqtttd.toml` ist die Konfigurationsdatei für den MQTT-Broker. Sie definiert die Einstellungen für den Broker, wie die maximale Anzahl von Verbindungen und die Grösse der Nachrichtenpakete. Sie enthält auch spezifische Einstellungen für verschiedene MQTT-Versionen (4.1 und 5.1), einschliesslich der IP-Adresse und des Ports, auf denen der Broker lauscht. Zusätzlich gibt es Einstellungen für die Verbindungsparameter, wie das Verbindungszeitlimit und die maximale Grösse der Nutzlast. Ein spezieller Abschnitt ist für die Konfiguration von MQTTeX, einem plattformübergreifenden MQTT-Desktop-Client, vorgesehen.

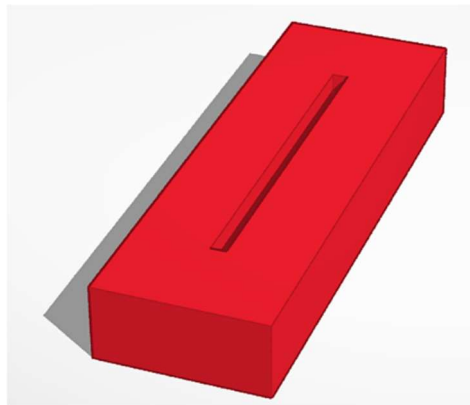
11 Screenshots



11 - Behälter Wassersteuerung



7 - Initial Testing: Sensor



10 - Initialversion 3D Druck Entwurf

```
INFO hub::mqttc: Sent = PingReq
INFO hub::mqttc: Received = PingResp
INFO hub::mqttc: Sent = PingReq
INFO hub::mqttc: Received = PingResp
INFO hub::mqttc: Sent = PingReq
INFO hub::mqttc: Received = PingResp
INFO hub::mqttc: Sent = PingReq
INFO hub::mqttc: Received = PingResp
INFO hub::mqttc: Received = b"WATERING_NEEDED_TRUE"
INFO hub::mqttc: Sent = PingReq
```

```
.....
WiFi connected
IP address: 192.168.1.212
Attempting MQTT connection...connected
starting checking period (30 seconds)...
Updated Watering State!
Going to sleep now... (1316.72 minutes)
```

9 - Verbindung zwischen Hub & Client

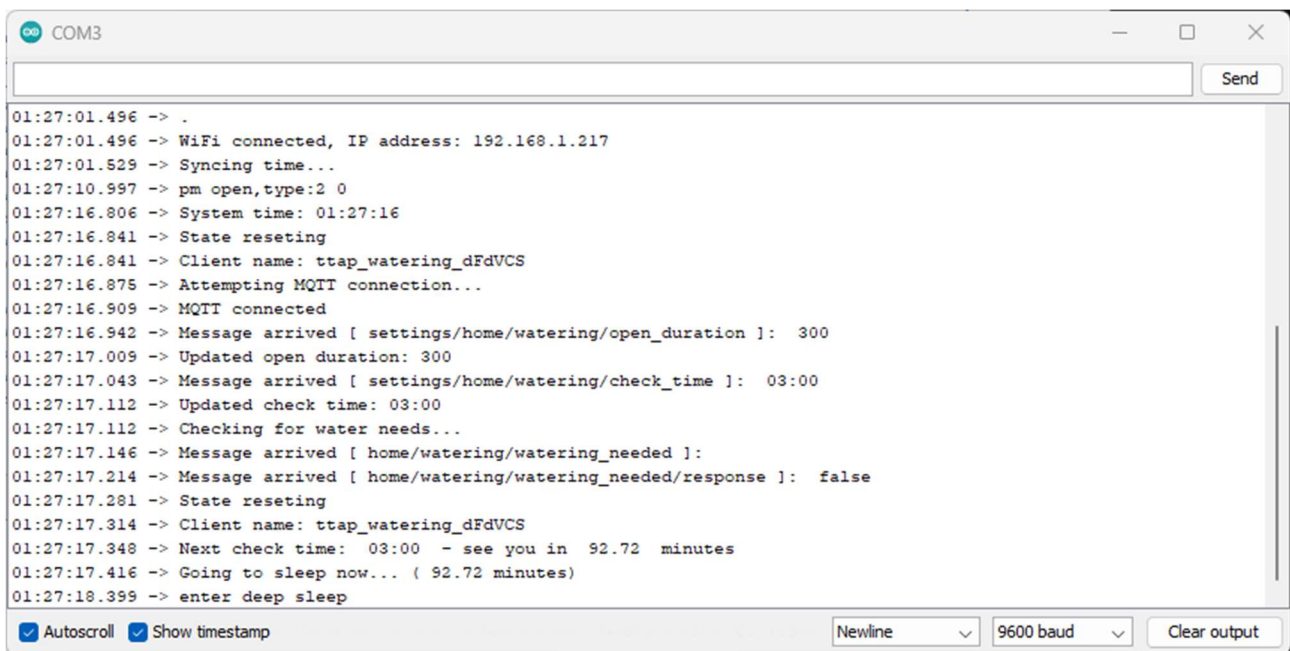


8 - Detailversion 3D Druck Modell


```
DEBUG hub::core::manager: ModuleManager initialized
DEBUG hub::core::manager: Registered handler for 'watering_needed'
DEBUG hub::core::manager: Registered module 'sensor'
DEBUG hub::core::manager: Registered handler for 'needs_water'
DEBUG hub::core::manager: Registered module 'watering'
INFO hub: TerraTap running... Press Ctrl+C to exit.
INFO hub: Starting MQTT broker... This may take a few seconds.
INFO hub::mqttc: Client created and stored in global static variable
DEBUG hub::core::manager: Published settings for 'watering' retain = true
DEBUG hub::core::manager: Published settings for 'sensor' retain = true
DEBUG hub::core::manager: Subscribed to 'home/watering/needs_water'
DEBUG hub::core::manager: Subscribed to 'home/sensor/watering_needed'
[unoptimized + debuginfo] target(s) in 0.39s
mqtttd.exe
INFO rumqtttd::server::broker: Listening for remote connections config="v5-1" listen_addr="0.0.0.0:1884"
INFO rumqtttd::server::broker: Listening for remote connections config="v4-1" listen_addr="0.0.0.0:1883"
INFO rumqtttd::server::broker: accept name="v4-1" addr=127.0.0.1:57680 count=0 tenant=None
INFO run:[>] incoming{connection_id=0}:incoming_connect{client_id="hub"}: rumqtttd::router::routing: Clie

DEBUG rumqttc::state: Publish. Topic = settings/watering/check_time, Pkid = 1, Payload Size = 7
DEBUG rumqttc::state: Publish. Topic = settings/sensor/check_time, Pkid = 2, Payload Size = 7
DEBUG rumqttc::state: Pingreq,
```

12 - Hub Boot



```
COM3
01:27:01.496 -> .
01:27:01.496 -> WiFi connected, IP address: 192.168.1.217
01:27:01.529 -> Syncing time...
01:27:10.997 -> pm open,type:2 0
01:27:16.806 -> System time: 01:27:16
01:27:16.841 -> State resetting
01:27:16.841 -> Client name: ttap_watering_dFdVCS
01:27:16.875 -> Attempting MQTT connection...
01:27:16.909 -> MQTT connected
01:27:16.942 -> Message arrived [ settings/home/watering/open_duration ]: 300
01:27:17.009 -> Updated open duration: 300
01:27:17.043 -> Message arrived [ settings/home/watering/check_time ]: 03:00
01:27:17.112 -> Updated check time: 03:00
01:27:17.112 -> Checking for water needs...
01:27:17.146 -> Message arrived [ home/watering/watering_needed ]:
01:27:17.214 -> Message arrived [ home/watering/watering_needed/response ]: false
01:27:17.281 -> State resetting
01:27:17.314 -> Client name: ttap_watering_dFdVCS
01:27:17.348 -> Next check time: 03:00 - see you in 92.72 minutes
01:27:17.416 -> Going to sleep now... ( 92.72 minutes)
01:27:18.399 -> enter deep sleep

Autoscroll Show timestamp Newline 9600 baud Clear output
```

13 - Watering Boot

12 Testkonzept

12.1 Teststrategie

Die Teststrategie ist in drei Phasen unterteilt: Unit-Tests für den HUB, manuelle Tests der Mikrocontroller während der Entwicklung und abschliessend End-to-End Tests. Mithilfe dieser Strategie soll die Fehlerfreie Funktionsweise des Systems gewährleistet werden.

12.1.1 Testclients

Für das Manuelle Testen wird MQTTX verwendet. MQTTX ist ein Desktop Client welcher das Verbinden zu einem MQTT Broker zulässt und somit das Debugging vereinfacht. Ausserdem wird mithilfe der Arduino IDE die Funktionsweise der einzelnen Komponenten überprüft sowie der Code welcher für diese geschrieben wird.

Für alle Rust basierten Tests muss eine Rust Version auf dem System sein.



15 - MQTTX Logo

<https://mqttx.app>



16 - Arduino Logo

<https://www.arduino.cc>



14 - Rust Logo

<https://rustup.rs>

12.2 Test-Cases

Testfall-Nr	1
Anforderungen	Modul-Manager sollte korrekt initialisiert werden.
Beschreibung	Überprüfung, ob der Modul-Manager korrekt erstellt wird.
Voraussetzung	Keine
Eingabe	Erstellen eines neuen Modul-Managers.
Erwartetes Resultat	Modul-Manager wird erfolgreich initialisiert.

15 - Testfall 1: Initialisierung des Modul-Managers

Testfall-Nr	2
Anforderungen	Naive Time sollte korrekt serialisiert werden.
Beschreibung	Überprüfung der Serialisierung von Naive Time.
Voraussetzung	Keine
Eingabe	Serialisieren eines Naive Time Objekts.
Erwartetes Resultat	Naive Time wird erfolgreich serialisiert.

16 - Testfall 2: Serialisierung von Naive Time

Testfall-Nr	3
Anforderungen	Modul sollte korrekt registriert werden.
Beschreibung	Überprüfung der Registrierung eines neuen Moduls.
Voraussetzung	Initialisierter Modul-Manager
Eingabe	Registrieren eines neuen Moduls.
Erwartetes Resultat	Modul wird erfolgreich registriert.

17 - Testfall 3: Registrierung eines Moduls

Testfall-Nr	4
Anforderungen	Präfix sollte korrekt im Modul-Manager gesetzt werden
Beschreibung	Überprüfung der Präfix-Einstellung.
Voraussetzung	Initialisierter Modul-Manager.
Eingabe	Setzen eines Präfixes.
Erwartetes Resultat	Präfix wird erfolgreich gesetzt.

18 - Testfall 4: Präfix-Einstellung im Modul-Manager

Testfall-Nr	5
Anforderungen	Naive Time sollte korrekt deserialisiert werden.
Beschreibung	Überprüfung der Deserialisierung von Naive Time.
Voraussetzung	Keine
Eingabe	Deserialisieren eines Naive Time Objekts.
Erwartetes Resultat	Naive Time wird erfolgreich deserialisiert.

19 - Testfall 5: Deserialisierung von Naive Time

Testfall-Nr	6
Anforderungen	Konfigurationsdatei sollte korrekt gespeichert werden.
Beschreibung	Überprüfung des Speichervorgangs der Konfigurationsdatei.
Voraussetzung	Initialisierte Konfiguration
Eingabe	Speichern der Konfigurationsdatei.
Erwartetes Resultat	Konfigurationsdatei wird erfolgreich gespeichert.

20 - Testfall 6: Speichern der Konfigurationsdatei

Testfall-Nr	7
Anforderungen	Konfigurationsdatei sollte korrekt geladen werden.
Beschreibung	Überprüfung des Ladevorgangs der Konfigurationsdatei.
Voraussetzung	Vorhandene Konfigurationsdatei
Eingabe	Laden der Konfigurationsdatei.
Erwartetes Resultat	Konfigurationsdatei wird erfolgreich geladen.

21 - Testfall 7: Laden der Konfigurationsdatei

Testfall-Nr	8
Anforderungen	Simple Client Modul sollte korrekt arbeiten.
Beschreibung	Überprüfung der Funktionalität des Simple Client Moduls.
Voraussetzung	Initialisiertes Modul
Eingabe	Ausführen eines einfachen Modultests.
Erwartetes Resultat	Simple Client Modul funktioniert wie erwartet.

22 - Testfall 8: Simple Client Modul Test

Testfall-Nr	9
Anforderungen	MQTT Client sollte korrekt mit dem Hub kommunizieren und Nachrichten austauschen.
Beschreibung	Überprüfung der Verbindung und Nachrichtenübermittlung zwischen MQTT Client und Hub.
Voraussetzung	Laufender MQTT Broker, gestarteter Hub
Eingabe	Senden und Empfangen von Nachrichten über MQTT.
Erwartetes Resultat	Nachrichten werden erfolgreich gesendet und empfangen.

23 - Testfall 9: MQTT Verbindung und Nachrichtenübermittlung

Testfall-Nr	10
Anforderungen	Hub sollte korrekt auf Anfragen zum Bewässerungsbedarf reagieren.
Beschreibung	Überprüfung, ob der Hub den Bewässerungsbedarf korrekt meldet.
Voraussetzung	Laufender Hub, angeschlossene Sensoren
Eingabe	Anfrage an den Hub, ob Bewässerung benötigt wird.
Erwartetes Resultat	Hub meldet korrekt, ob Bewässerung benötigt wird (false).

24 - Testfall 10: Überprüfung des Bewässerungsbedarfs

Testfall-Nr	11
Anforderungen	Hub sollte korrekt bestätigen, dass Bewässerung erforderlich ist.
Beschreibung	Überprüfung, ob der Hub eine Bestätigung sendet, dass Bewässerung erforderlich ist.
Voraussetzung	Laufender Hub, angeschlossene Sensoren
Eingabe	Anfrage an den Hub zur Bestätigung, dass Bewässerung erforderlich ist.
Erwartetes Resultat	Hub bestätigt korrekt, dass Bewässerung erforderlich ist (true).

25 - Testfall 11: Bestätigung der Bewässerung

12.3 Unit-Tests für den Hub

Da das HUB in Rust entwickelt wird, werden Tests mit dem Integrierten Test-Framework geschrieben. Unit-Tests wurden für die Core-Komponenten des HUB geschrieben um diese auf ihre Fehlerfrei Funktion zu Testen. Die Unit-Tests wurden basierend auf den zuvor erstellten Test-Cases entwickelt.

Modul	Test	Beschreibung
core::manager	test_new	Stellt die Ordentliche Erstellung und Instanziierung des Managers sicher.
core::manager	test_register_module	Stellt sicher das alle Inhalte des ClientModuls richtig im Manager registriert werden.
core::manager	test_setting_prefix_correctly	Überprüft ob Einstellungen korrekt mit dem Topic des Moduls als Präfix versehen werden.
core::serde	test_serialize_naive_time	Stellt sicher das die Zeit Konfiguration richtig konvertiert werden kann.
core::serde	test_deserialize_naive_time	Stellt sicher das die Zeit Konfiguration richtig konvertiert werden kann.
core::traits::config	test_config_file_load	Überprüft ob die Konfigurationsdatei richtig geladen.
core::traits::config	test_config_file_save	Überprüft ob die Konfigurationsdatei richtig gesichert wird. Überprüft auch das Laden erneut.
core::traits::module	test_client_module	Überprüft ob ein Implementiertes Client Modul Ordnungsgemäss Funktioniert. Mit Einstellungen.
core::traits::module	test_simple_client_module	Überprüft ob ein minimal Implementiertes Client Modul Ordnungsgemäss Funktioniert.

26 - Unit-Tests für den Hub

12.4 E2E Testing

Die End-to-End-Tests sind dafür konzipiert, die korrekte Kommunikation zwischen den verschiedenen Komponenten des Systems zu gewährleisten. Dazu wird das HUB gestartet und die Nachrichten, die zwischen den Geräten ausgetauscht werden, werden simuliert.

Gerät	Beschreibung
Sensor	Sendet eine Nachricht an das Hub über home/sensor/watering_needed , prüft ob der Status verändert wurde.
Watering	Sendet eine Anfrage über home/watering/watering_needed , prüft ob eine Rückgabe erhalten wird.

27 - E2E Testing

13 Tests

13.1 Unit-Test Resultate

Tester: Fokko Vos

Datum: 10.06.2024

Resultat: Erfolgreich; Fehlerfrei

Modul	Test	Resultat
core::manager	test_new	OK
core::manager	test_register_module	OK
core::manager	test_setting_prefix_correctly	OK
core::serde	test_serialize_naive_time	OK
core::serde	test_deserialize_naive_time	OK
core::traits::config	test_config_file_load	OK
core::traits::config	test_config_file_save	OK
core::traits::module	test_client_module	OK
core::traits::module	test_simple_client_module	OK

28 - Unit-Test Resultate

Screenshot der Ausführung:

```
PS E:\040 Rust\ttap-sol> cargo test -p hub
Compiling hub v0.1.0 (E:\040 Rust\ttap-sol\hub)
Finished `test` profile [unoptimized + debuginfo] target(s) in 1.28s
Running unittests src\main.rs (target\debug\deps\hub-10413d951fdaf794.exe)

running 9 tests
test core::manager::tests::test_module_manager_new ... ok
test core::serde::tests::test_serialize_naive_time ... ok
test core::manager::tests::test_register_module ... ok
test core::manager::tests::test_setting_prefix_correctly ... ok
test core::serde::tests::test_deserialize_naive_time ... ok
test core::traits::config::tests::test_config_file_save ... ok
test core::traits::module::simple_module_tests::test_simple_client_module ... ok
test core::traits::config::tests::test_config_file_load ... ok
test core::traits::module::module_tests::test_client_module ... ok

test result: ok. 9 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

17 - Unit-Tests Resultate

13.2 E2E Resultate

Tester: Fokko Vos

Datum: 10.06.2024

Resultat: Erfolgreich; Fehlerfrei

```
PS E:\040 Rust\ttap-sol> cargo run -p e2e
Compiling e2e v0.1.0 (E:\040 Rust\ttap-sol\e2e)
Finished `dev` profile [unoptimized + debuginfo] target(s) in 2.85s
Running target\debug\e2e.exe
2024-06-10T17:39:45.453353Z INFO e2e: Starting integration test using Simulated versions of the Clients
2024-06-10T17:39:45.453910Z INFO e2e: Starting the Hub...
2024-06-10T17:39:45.460246Z INFO e2e: Hub started successfully
2024-06-10T17:39:45.460369Z INFO e2e: Setting up the simulated clients...
2024-06-10T17:39:45.460511Z INFO e2e: Configuring a MQTT Client to simulate clients connecting to the Hub
2024-06-10T17:39:45.460647Z INFO e2e: Creating the MQTT Client...
2024-06-10T17:39:45.461159Z INFO e2e: MQTT Client created successfully
2024-06-10T17:39:45.461330Z INFO e2e: Spawning handler for the MQTT Client...
2024-06-10T17:39:45.461487Z INFO e2e: Task spawned successfully
2024-06-10T17:39:45.461597Z INFO e2e: Waiting for the MQTT Client to connect to the Hub...
2024-06-10T17:39:45.461709Z INFO e2e: Starting the tests...
2024-06-10T17:39:45.461847Z INFO e2e: Waiting for all settings to be received...
2024-06-10T17:39:49.467608Z INFO e2e: Connected to MQTT broker
2024-06-10T17:39:49.468002Z INFO e2e: Subscribed to all topics
2024-06-10T17:39:49.469046Z INFO e2e: Received setting on topic 'home/sensor/check_time': 19:45
2024-06-10T17:39:49.469237Z INFO e2e: Received setting on topic 'home/watering/check_time': 19:50
2024-06-10T17:39:49.469508Z INFO e2e: Received setting on topic 'home/sensor/check_duration': 30
2024-06-10T17:39:49.469791Z INFO e2e: Received setting on topic 'home/watering/open_duration': 300
2024-06-10T17:39:50.468238Z INFO e2e: All settings received
2024-06-10T17:39:50.468637Z INFO e2e: ----- Watering Tests -----
2024-06-10T17:39:50.469052Z INFO e2e: Sending a message to the HUB to request the state
2024-06-10T17:39:50.469393Z INFO e2e: Waiting for the response...
2024-06-10T17:39:50.471546Z INFO e2e: Received response on topic 'home/watering/watering_needed': false
2024-06-10T17:39:51.471500Z INFO e2e: Watering tests completed
2024-06-10T17:39:51.47225Z INFO e2e: Watering tests passed: 1/1
2024-06-10T17:39:51.472844Z INFO e2e: ----- Sensor Tests -----
2024-06-10T17:39:51.473513Z INFO e2e: Sending a message to the hub to confirm watering is needed
2024-06-10T17:39:51.474153Z INFO e2e: Test if message was sent successfully, by requesting the watering state
2024-06-10T17:39:51.474689Z INFO e2e: Waiting for the response...
2024-06-10T17:39:51.478760Z INFO e2e: Received response on topic 'home/watering/watering_needed': true
2024-06-10T17:39:52.476497Z INFO e2e: Sensor tests completed
2024-06-10T17:39:52.476988Z INFO e2e: Sensor tests passed: 2/2
2024-06-10T17:39:52.477421Z INFO e2e: ----- Cleanup -----
2024-06-10T17:39:52.477846Z INFO e2e: Shutting down the MQTT client...
2024-06-10T17:39:52.478383Z INFO e2e: Shutting down MQTT client...
2024-06-10T17:39:52.479000Z INFO e2e: MQTT client shut down successfully
2024-06-10T17:39:52.479356Z INFO e2e: Killing the Hub...
2024-06-10T17:39:52.479737Z INFO e2e: Hub killed successfully
2024-06-10T17:39:52.479995Z INFO e2e: E2E test completed
2024-06-10T17:39:52.480227Z INFO e2e: Tests passed: 3/3
```

18 - E2E Resultate

14 Reflektion

Das TerraTap-Projekt war eine intensive und lehrreiche Erfahrung, die über fünf Wochen verschiedene Phasen von der Informationsbeschaffung bis zur Realisierung und Auswertung durchlief. Der zentrale Fokus lag auf der Entwicklung eines automatisierten Wassersteuerungssystems. Rückblickend war das Projekt geprägt von technischen Herausforderungen, kreativen Lösungen und wertvollen Erkenntnissen.

Eine der grössten Herausforderungen bestand in der initialen Informationsbeschaffung. Die fundierte Recherche zu Strom, Relays und anderen Komponenten bildete die Grundlage für das gesamte Projekt. Diese Phase verdeutlichte die Bedeutung einer gründlichen Vorbereitung, um spätere Fehler zu minimieren und die Effizienz der nachfolgenden Arbeitsschritte zu maximieren.

Die Auswahl geeigneter Tools war ein weiterer kritischer Punkt. Hier lernte ich, dass Benutzerfreundlichkeit und die Eignung für spezifische Projektanforderungen entscheidend sind. Tools wie Tinkercad und die Arduino-IDE erwiesen sich als besonders nützlich und unterstrichen die Notwendigkeit, flexible und anpassungsfähige Werkzeuge zu wählen.

Die Realisierungsphase bot wertvolle Lernerfahrungen in der praktischen Anwendung theoretischer Kenntnisse. Die Programmierung der Steuerungssoftware und des MQTT-Servers war anspruchsvoll und erforderte eine tiefgehende Einarbeitung in die entsprechenden Technologien. Iterative Tests und das gründliche Studium der Dokumentation waren hierbei essenziell. Ein kreativer Moment war die Nutzung eines alten Tupperware-Behälters für die Wassersteuerung, was die Bedeutung praktischer und kosteneffizienter Lösungen unterstrich.

Die kontinuierliche Dokumentation und Reflexion über den Projektverlauf waren entscheidend für den Erfolg des Projekts. Diese Praxis half, den Überblick zu behalten, Entscheidungen zu validieren und die Transparenz zu wahren. Die Reflexionen ermöglichten es, aus Fehlern zu lernen und zukünftige Projekte effizienter zu gestalten.

Insgesamt hat das TerraTap-Projekt meine Fähigkeiten in der technischen Recherche, der Auswahl und Anwendung von Tools sowie in der praktischen Umsetzung technischer Lösungen erheblich erweitert. Die iterative Herangehensweise und die kontinuierliche Reflexion erwiesen sich als Schlüsselkomponenten für den Projekterfolg. Diese Erfahrungen und Erkenntnisse werden in zukünftigen Projekten von grossem Nutzen sein und haben mein Verständnis für die Komplexität und die Anforderungen technischer Entwicklungsprojekte vertieft. Ich bin stolz auf die erzielten Ergebnisse und freue mich darauf, die gewonnenen Kenntnisse in neuen Herausforderungen anzuwenden.

15 Abschluss und Ausblick

Das TerraTap-Projekt war ein anspruchsvolles und lehrreiches Unterfangen, bei dem ich zahlreiche technische Herausforderungen meistern konnte. Die Auswahl und Nutzung von Tools wie Tinkercad und der Arduino-IDE sowie die Programmierung des ESP8266-Chips und des MQTT-Servers waren zentrale Aspekte des Projekts. Der kreative Einsatz von Alltagsgegenständen, wie einem Tupperware-Behälter für die Wassersteuerung, zeigte die Bedeutung praktischer und kosteneffizienter Lösungen.

Insgesamt konnte ich durch dieses Projekt meine technischen Fähigkeiten erheblich erweitern und wertvolle Erfahrungen sammeln, die mir in zukünftigen Projekten von grossem Nutzen sein werden. Besonders die Bereiche der Mikrocontroller-Programmierung und der Entwicklung von IoT-Lösungen haben mein Interesse geweckt und bieten spannende Perspektiven für zukünftige Arbeiten.

In Zukunft plane ich, die im TerraTap-Projekt erlernten Techniken und Methoden weiter zu vertiefen und in neuen Projekten anzuwenden. Ich freue mich darauf, diese Herausforderungen anzunehmen und mein Wissen kontinuierlich zu erweitern, um innovative und effiziente Lösungen zu entwickeln.

16 Verweise

16.1 Tabellenverzeichnis

1 - VERSIONIERUNG	5
2 – ZEITPLANUNG	8
3 - ARBEITSRAPPORT 13.05 – 19.05	10
4 - ARBEITSRAPPORT 20.05 – 26.05	12
5 - ARBEITSRAPPORT 27.05 – 02.06	14
6 - ARBEITSRAPPORT 03.05 – 09.06	15
7 - ARBEITSRAPPORT 10.06 – 14.06	16
8 - ERWEITERUNGEN VISUAL STUDIO CODE.....	23
9 – STEUERUNGSSYSTEM BIBLIOTHEKEN	24
10 - HUB BIBLIOTHEKEN	25
11- MQTTD BIBLIOTHEKEN.....	25
12 - HARDWARE KOMPONENTEN: SENSOR	28
13 - HARDWARE KOMPONENTEN: WASSERSTEUERUNG	28
14 - HARDWARE KOMPONENTEN: HUB	28
15 - TESTFALL 1: INITIALISIERUNG DES MODUL-MANAGERS	74
16 - TESTFALL 2: SERIALISIERUNG VON NAIVE TIME	74
17 - TESTFALL 3: REGISTRIERUNG EINES MODULS	74
18 - TESTFALL 4: PRÄFIX-EINSTELLUNG IM MODUL-MANAGER	74
19 - TESTFALL 5: DESERIALISIERUNG VON NAIVE TIME	74
20 - TESTFALL 6: SPEICHERN DER KONFIGURATIONSDATEI.....	75
21 - TESTFALL 7: LADEN DER KONFIGURATIONSDATEI	75
22 - TESTFALL 8: SIMPLE CLIENT MODUL TEST.....	75
23 - TESTFALL 9: MQTT VERBINDUNG UND NACHRICHTENÜBERMITTLUNG.....	75
24 - TESTFALL 10: ÜBERPRÜFUNG DES BEWÄSSERUNGSBEDARFS	75
25 - TESTFALL 11: BESTÄTIGUNG DER BEWÄSSERUNG	76
26 - UNIT-TESTS FÜR DEN HUB	77
27 - E2E TESTING.....	77
28 - UNIT-TEST RESULTATE.....	78
29 - GLOSSAR	86
30 - ARDUINO BIBLIOTHEKVERSIONEN	87
31 - RUST PAKETVERSIONEN	87
32 - RUST EXTENSION VERSIONEN.....	87
33 - KOSTENAUFSTELLUNG.....	88

16.2 Abbildungsverzeichnis

1 - SYSTEMARCHITEKTUR.....	20
2 - SENSOR SCHALTKEIS (VISUELL LINKS, TECHNISCH RECHTS)	26
3 - WASSERSTEUERUNG SCHALTKEIS (VISUELL LINKS, TECHNISCH RECHTS)	26
4 - KOMMUNIKATIONS DIAGRAMM	27
5 - SENSOR SCHALTKEIS	29
6 - WASSERSTEUERUNG SCHALTKEIS.....	30
7 - INITIAL TESTING: SENSOR	71
8 - DETAILVERSION 3D DRUCK MODELL	71
9 - VERBINDUNG ZWISCHEN HUB & CLIENT	71
10 - INITIALVERSION 3D DRUCK ENTWURF	71
11 - BEHÄLTER WASSERSTEUERUNG	71
12 - HUB BOOT	72
13 - WATERING BOOT	72
14 - RUST LOGO	73
15 - MQTTX LOGO	73
16 - ARDUINO LOGO	73
17 - UNIT-TESTS RESULTATE	78
18 - E2E RESULTATE	79

17 Quellenverzeichnis

Arduino IDE. (14. 06 2024). Von Arduino: <https://www.arduino.cc/> abgerufen

CratesIO. (14. 06 2024). Von Rust Community: <https://crates.io/> abgerufen

DrawIO. (14. 06 2024). Von diagrams.net: <https://www.diagrams.net/> abgerufen

FreeCAD. (14. 06 2024). Von FreeCAD Community: <https://www.freecadweb.org/> abgerufen

Fritzing. (14. 06 2024). Von Fritzing Project: <https://fritzing.org/> abgerufen

Fusion 360. (14. 06 2024). Von Autodesk: <https://www.autodesk.com/products/fusion-360/overview> abgerufen

MQTT Documentation. (14. 06 2024). Von MQTT Rust Library: <https://crates.io/crates/rumqttc> abgerufen

OpenAI. (14. 06 2024). *GPT (Generative Pre-trained Transformer).* Von OpenAI: <https://www.openai.com/> abgerufen

PlatformIO. (14. 06 2024). Von PlatformIO Labs: <https://platformio.org/> abgerufen

Proteus. (14. 06 2024). Von Labcenter Electronics: <https://www.labcenter.com/> abgerufen

Tinkercad. (14. 06 2024). Von Autodesk: <https://www.tinkercad.com/> abgerufen

Visual Studio Code. (14. 06 2024). Von Microsoft: <https://code.visualstudio.com/> abgerufen

18 Glossar

Begriff/Abkürzung	Beschreibung
3D-Druck	Die Konstruktion eines dreidimensionalen Objekts aus einem digitalen Modell.
3D-Druck Modellierung	Der Prozess der Erstellung einer dreidimensionalen digitalen Darstellung eines Objekts.
Abonnement	In Nachrichtensystemen eine Anfrage, Benachrichtigungen oder Daten zu erhalten, wenn diese verfügbar werden.
AOUT-Pin	Ein Pin an einem Sensor oder Mikrocontroller, der ein analoges Signal ausgibt.
API	Ein Satz von Werkzeugen, Definitionen und Protokollen für den Bau und die Integration von Anwendungssoftware.
Arduino IDE	Eine integrierte Entwicklungsumgebung für den Bau und das Hochladen von Programmen auf Arduino-kompatible Boards.
Async-trait	Eine Rust-Bibliothek, die die Verwendung von asynchronen Funktionen in Trait-Definitionen ermöglicht.
Auth-Token	Ein Sicherheitstoken, das zur Authentifizierung dient.
ClientModule Trait	Ein Set von Methoden in Rust, das spezifische Funktionalitäten für Client-Module bereitstellt.
Cloud Service	Online-Dienste, die verschiedene Computerressourcen über das Internet bereitstellen.
CP2102	Ein USB-zu-Seriell-Konverter-IC, der die Kommunikation zwischen einem Computer und einem Mikrocontroller über USB ermöglicht.
Crates.io	Das Paket-Registry der Rust-Community, das es Entwicklern ermöglicht, ihre Bibliotheken und Projekte zu teilen.
DC-DC Buck Converter	Ein Netzteil, das die Spannung von seinem Eingang auf seinen Ausgang reduziert.
Debugging	Der Prozess des Findens und Behebens von Fehlern in einem Software- oder Hardwaresystem.
Downstepper-Module	Ein anderes Wort für DC-DC Buck Converter, der die Spannung von einem höheren Niveau auf ein niedrigeres Niveau reduziert.
E2E Testing	End-to-End-Tests, die die Funktionalität und Leistung eines Systems von Anfang bis Ende validieren.
Eingangsverarbeitung	Die Methoden und Funktionen, die verwendet werden, um Eingabedaten von Sensoren oder Benutzeraktionen zu verarbeiten.
Eingebettete Systeme	Computersysteme, die in elektronische Geräte eingebettet sind.
Ereignisgesteuertes Debugging	Eine Debugging-Methode, die sich auf Ereignisse konzentriert, die Aktionen im System auslösen.

ESP32

Eine Serie von Mikrocontrollern mit integriertem Wi-Fi und dualen Bluetooth, verwendet für verschiedene IoT-Anwendungen.

ESP8266	Ein kostengünstiger Wi-Fi-Mikrochip mit vollständigem TCP/IP-Stack und Mikrocontroller-Funktionalität.
Firmware	Eine spezielle Klasse von Computersoftware, die eine niedrige Kontrolle über die Hardware eines Geräts bietet.
Fritzing	Eine Open-Source-Software für das Design elektronischer Schaltungen.
GPIO	Allgemeine Eingabe-/Ausgabe-Pins auf einem integrierten Schaltkreis oder Mikrocontroller, die von Software gesteuert werden können.
Hochfrequenz	Bezieht sich auf die hohe Oszillationsrate elektromagnetischer Wellen, relevant für drahtlose Kommunikationstechnologien.
IoT	Internet der Dinge, ein Netzwerk von physischen Geräten mit Elektronik, Software und Sensoren, die Daten austauschen.
Kommunikationsprotokoll	Ein System von Regeln, das es zwei oder mehr Entitäten eines Kommunikationssystems ermöglicht, Informationen auszutauschen.
Kompilieren	Der Prozess der Umwandlung von Quellcode in ausführbaren Maschinencode.
Komponentenregistrierung	Der Prozess, eine Komponente in ein System aufzunehmen, damit sie verwaltet und genutzt werden kann.
Microcontroller	Ein kleiner Computer auf einem einzigen integrierten Schaltkreis, der einen Prozessorkern, Speicher und programmierbare Ein-/Ausgabe-Peripheriegeräte enthält.
Module	Eine eigenständige Einheit von Code, die eine spezifische Funktion innerhalb eines grösseren Systems ausführt.
ModuleManager	Ein Komponent im System, das verschiedene Module und deren Konfigurationen verwaltet.
MQTT	Message Queuing Telemetry Transport, ein leichtgewichtiges Nachrichtenprotokoll für kleine Sensoren und mobile Geräte.
NodeMCU	Ein Entwicklungsboard basierend auf dem ESP8266, das zur Prototypenentwicklung für IoT-Produkte verwendet wird.
Pneumatisches Ventil	Ein durch Luftdruck betriebenes Ventil, das den Wasserfluss steuert.
Proteus	Eine Software-Suite für elektronische Designautomatisierung, einschliesslich Schaltungssimulation und PCB-Design.
PubSubClient	Eine Arduino-Bibliothek, die das MQTT-Protokoll implementiert.
Relay Modul	Ein elektrisch betriebenes Schaltmodul, das es ermöglicht, mit Niederspannung Hochleistungsgeräte zu steuern.
Rust	Eine Programmiersprache, die Speicherfehler vermeiden soll und gleichzeitig hohe Leistung bietet.

Sensor	Ein Gerät, das Feuchtigkeitswerte im Boden misst, um die Bewässerung zu optimieren.
Serialization/Deserialization	Der Prozess der Umwandlung einer Datenstruktur in ein Format, das einfach gespeichert oder übertragen werden kann und wieder zurück.
SimpleConnect	Eine Bibliothek oder Funktion, die grundlegende Verbindungsbehandlungen erleichtert.
Systemintegration	Der Prozess, verschiedene Teilsysteme zu einem System zusammenzuführen und sicherzustellen, dass sie zusammenarbeiten.
Thread-Sichere Initialisierung	Die sichere Initialisierung einer Variablen oder Ressource in einer Multi-Thread-Umgebung.
Tinkercad	Eine benutzerfreundliche Webanwendung für 3D-Design, Elektronik und Kodierung.
Trait	In Rust, eine Sammlung von Methoden, die für einen unbekannten Typ definiert sind.
Unit-Test Framework	Ein Software-Framework, das das Schreiben und Ausführen von Unit-Tests erleichtert.
Unit-Tests	Tests, die von Softwareentwicklern geschrieben und ausgeführt werden, um sicherzustellen, dass ein spezifischer Abschnitt einer Anwendung wie erwartet funktioniert.
Visual Studio Code	Ein Quelltext-Editor, der von Microsoft für Windows, Linux und macOS entwickelt wurde.
Zentraler Knotenpunkt	Ein zentraler Punkt, an dem alle Datenverbindungen konvergieren und der die Kommunikation und den Datenaustausch verwaltet.

19 Anhang

19.1 Arduino Bibliothekversionen

Name	Version
PubSubClient	2.8.0
ESP8266WiFi	3.1.2 - Esp8266 Board by ESP8266 Community

30 - Arduino Bibliothekversionen

19.2 Rust Paketversionen

Name	Version
serde	1.0.203
serde_json	1.0.117
tokio	1.38.0
tracing	0.1.40
tracing-subscriber	0.3.18
rumqttc	0.24.0
rumqttd	0.19.0
once_cell	1.19.0
chrono	0.4.38
async-trait	0.1.80
config	0.14.0

31 - Rust Paketversionen

19.3 Rust Extension Versionen

Erweiterung	Version
rust-analyzer	v0.4.1994 (pre-release)
crates	v0.6.6
Dracula For Rust Theme	v0.0.4
Even Better TOML	v0.19.2 (Preview)
Prettier - Code formatter (Rust)	v0.1.9
Rust Test Explorer	v0.11.0 (Preview)
Error Lens	v3.18.0
Todo Tree	v0.0.226
GitHub Copilot	v1.200.0

32 - Rust Extension Versionen

19.4 Kosten

Artikel	Preis
32 Stück doppelseitige Leiterplatte Prototyp-Kit	4,90 USD
1 Kanal 12V Relais Modul Platinen Abschirmung unterstützt High-und Low-Level-Trigger	1,37 USD
Kapazitive Boden Feuchtigkeit Sensor Korrosion Beständig	2,40 USD
Nodemcu v2 esp8266 ESP-12E esp 12e wifi entwicklungs karte cp2102	9,66 USD
Bt 2,4g wifi modul esp32 mit batterie halter ESP32-WROOM-32 cp2102	0,99 USD
Elektrische Magnetventil DN20 normal Geschlossen Pneumatisch	19,83 USD
560 Stück u-Form löt freies Breadboard-Überbrückungskabel-Kabels	6,27 USD
Dupont-Linie 10cm/20cm/30cm	10,30 USD
15m PVC wasserdichtes selbst klebendes Klebeband schwarz	5,05 USD
5m Nano Klebeband Doppelseitiges Klebeband	2,41 USD
5 zweite Starke Kleber Instant Klebstoff Metall Kunststoff	10,14 USD
DC 12V zu 5V 3A Step Down Modul	2,61 USD
Eine 1 kanal relais modul 12V high und low-level-trigger	2,68 USD
3D Druck	20.90 USD
Summe	99.51 USD
Summe CHF	90.46 CHF

33 - Kostenaufstellung