

**Design und Entwicklung eines smarten
Bewässerungssystems für Pflanzen**

Studienarbeit I

Des Studiengangs Elektrotechnik – Automatisierung
an der Dualen Hochschule Baden-Württemberg Mannheim

Hannah Grüne

02.01.2025

Bearbeitungszeitraum:	30.09.2024 – 02.01.2025
Matrikelnummer, Kurs:	1529271, TEL22AT1
Betreuer:	Prof. Dr.-Ing. Bozena Lamek-Creutz

Eigenständigkeitserklärung

Ich, Hannah Grüne, versichere hiermit, dass ich meine Projektarbeit mit dem Thema: „Design und Entwicklung eines smarten Bewässerungssystems für Pflanzen“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Mannheim, der

Ort, Datum

Unterschrift (Hannah Grüne)

Zusammenfassung

Die Studienarbeit mit dem Titel „Design und Entwicklung eines smarten Bewässerungssystems für Pflanzen“ untersucht die Entwicklung eines automatisierten Systems zur Pflanzenbewässerung. Ziel ist die Umsetzung eines modularen, auf einem Arduino basierenden Systems, das die Feuchtigkeit der Erde und den Wasserstand überwacht, um bei Bedarf automatisiert zu gießen. Der Bericht geht auf die Auswahl und Kalibrierung der Hardwarekomponenten, darunter Bodenfeuchtesensoren, Füllstandsensoren und Pumpen, ein. Zudem wird ein Algorithmus implementiert, der Gießzyklen basierend auf Sensordaten steuert und die Ergebnisse in einer JSON-Datei speichert, die als Datenschnittstelle für eine geplante App dient. Die Arbeit hebt Herausforderungen wie Stromversorgung und Datenverarbeitung hervor und bietet Ansätze zur Integration von Optimierungsmöglichkeiten wie 3D-gedruckten Gehäusen und zusätzlichen Sensoren. Sie schließt mit einem funktionsfähigen System, das zukünftige Erweiterungen ermöglicht.

Abstract

The student research project entitled “Design and development of a smart irrigation system for plants” investigates the development of an automated system for watering plants. The aim is to implement a modular, Arduino-based system that monitors soil moisture and water levels to automate watering when needed. The report covers the selection and calibration of the hardware components, including soil moisture sensors, level sensors and pumps. It also implements an algorithm that controls watering cycles based on sensor data and stores the results in a JSON file that serves as a data interface for a planned app. The work highlights challenges such as power supply and data processing and offers approaches for integrating optimisation options such as 3D-printed housings and additional sensors. It concludes with a functional system that enables future extensions.

Inhaltsverzeichnis

Eigenständigkeitserklärung.....	II
Zusammenfassung	III
Abstract	III
Inhaltsverzeichnis	IV
Abbildungsverzeichnis	VI
Tabellenverzeichnis.....	VI
Listingverzeichnis	VI
Abkürzungsverzeichnis	VII
Vorwort	VIII
Einleitung	1
1 Stand der Technik	3
1.1 Aktuelle Technologien im Bereich Smart Farming	3
1.2 Smart Home Systeme zur Bewässerung von Pflanzen	4
2 „Hardware“-Aufbau.....	6
2.1 Konzepterstellung und Komponentenübersicht	6
2.2 Auswahl des Mikrocontrollers + Vergleich	7
2.3 Verwendete Sensoren/Aktoren	10
2.3.1 Bodenfeuchtesensor	10
2.3.2 Füllstandsensor	11
2.4 Schaltungsaufbau	11
2.5 Kalibrierung der Sensoren und Aktoren	16
3 Programmablauf.....	21
3.1 Abgrenzung zu voriger Studienarbeit.....	21
3.2 Gießzyklus	22
3.3 Schreiben und Speichern der JSON-Datei	25
3.4 Funktionstest	27
4 Datenaustausch zwischen App und Arduino.....	29
4.1 Technische Optionen für die Umsetzung	29
4.2 Geplante Umsetzung der Kommunikation mit REST-API	31

5	Fazit.....	32
	Anhangsverzeichnis.....	IX
	Anhang	X
	Literaturverzeichnis.....	XVIII

Abbildungsverzeichnis

Abbildung 1: Übersicht Pflanzenbewässerungssystem	6
Abbildung 2: Schaltungsaufbau des finalen Pflanzenbewässerungssystems	13
Abbildung 3: Realer finaler Schaltungsaufbau	15
Abbildung 4: Grobe Skizzen Wasserbehälter für Pflanzenbewässerungssystem	16
Abbildung 6: Messwerte zur Grenzwertfassung des Bodenfeuchtesensors (A0) - Zustand trocken	18
Abbildung 7: Links: Gesamtaufbau des Pflanzenbewässerungssystems; Rechts: Installation der Pumpe im Wasserbehälter	20
Abbildung 8: Ablaufdiagramm des Gießzyklus innerhalb der loop-Funktion	22
Abbildung 12: Übersicht der verwendeten Klassen im Arduino Code	25

Tabellenverzeichnis

Tabelle 1: Marktanalyse Mikrocontroller – die relevanten Merkmale im Vergleich	8
Tabelle 2: Auflistung der verwendeten Komponenten	12
Tabelle 3: Pinanschlüsse für finalen Schaltungsaufbau	14
Tabelle 4: Ermittelte Mittelwerte der Grenzwerte des Bodenfeuchte- und Füllstandsensors	19
Tabelle 5: Durchflussbestimmung Pumpe	19
Tabelle 6: Übersicht der Optionen für den Datenaustausch zwischen dem Arduino und der App	29

Listingverzeichnis

Listing 1: Testprogramm zur Messdatenerfassung zur Definition der Grenzwerte	17
Listing 2: loop-Funktion des Arduino-Programms	23
Listing 3: Klasse Pump	24
Listing 4: Funktion waterPlant innerhalb der Klasse Plant	24
Listing 5: Ausschnitt aus saveToJson-Funktion	26
Listing 6: JSON-Datei des Funktionstests	28

Abkürzungsverzeichnis

HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
MISO	Master Input Slave Output
REST-API	Representational State Transfer - Application Programming Interface
SCK	Serial Clock
SDA	Serial Data

Vorwort

Die Studienarbeit befasst sich mit der Ausarbeitung und Umstrukturierung eines Pflanzenbewässerungssystems mit einem App-Interface, welche in der fünften Theoriephase im Zeitraum vom 30.09.2024 bis zum 02.01.2025 an der Dualen Hochschule Baden-Württemberg absolviert wurde. Da es sich bei dieser um ein zweigeteiltes Projekt handelt, wird im Verlaufe der Arbeit gegebenenfalls auf den anderen Bericht verwiesen. Dabei kann es zu leichten Übereinstimmungen kommen.

Der Teil der Arbeit mit dem Titel „Entwicklung einer Android-App zur automatisierten Bewässerung von Pflanzen“ befasst sich primär mit dem App-Interface und der Schnittstelle zu dem verwendeten Mikrocontroller, während sich dieser Teil auf die Hardwarekomponenten, durchgeführte Messungen sowie die Umsetzung fokussiert.

Hiermit möchte ich mich auch herzlich bei meiner Kommilitonin Celine Schneider und meiner Betreuerin Prof. Dr.-Ing. Bozena Lamek-Creutz für die Unterstützung bedanken.

Einleitung

Das Thema Smart Home ist in Zeiten des Internet of Things (IoT) nicht wegzudenken. Während 2019 die Anzahl der Nutzer weltweit von Smart Home Installationen bei 191,38 Millionen lag, so hat sich dieser im September 2024 bereits mit 422,19 Millionen Nutzern mehr als verdoppelt. Prognosen zeigen für 2028 eine weitere Steigerung der Smart Home Nutzer von 180%. Vgl. [1] Diese Zahlen belegen, dass gerade im Bereich der Digitalisierung mit weiteren innovativen Lösungen zu rechnen ist.

Aber was genau versteckt sich hinter dem Begriff Smart Home? „Der Begriff "Smart Home" zielt auf das informations- und sensortechnisch aufgerüstete, in sich selbst und nach außen vernetzte Zuhause“ [2] Dabei liegt vor allem eine Automatisierung von Aufgaben und damit eine Erleichterung des Alltags für den Anwender im Vordergrund. Vgl. [2] Mittels dieser automatisierten Prozesse bieten sich neue Möglichkeiten, Geräte ortsunabhängig steuern zu können. So lässt sich beispielsweise komfortabel durch ein ferngesteuertes Einschalten von Klimaanlage oder Heizung vor der Rückkehr in die eigenen vier Wände die perfekte Umgebungstemperatur sicherstellen. Ein weiterer Vorteil der Nutzung von Smart Home Installationen die Ermittlung und Nutzung von gerätespezifischen Daten. Somit lassen sich Aspekte wie energieeffizienteres Wohnen oder nutzerangepasstes Verhalten realisieren. Mittels der Datenauswertung ergibt sich zusätzlich die Möglichkeit, einfache Abläufe vollständig automatisiert ablaufen zu lassen. Viele dieser Abläufe lassen sich hierbei bereits mittels eines gängigen Mikrocontrollers realisieren.

Während bei den meisten Produkten viele unterschiedliche Lösungen unterschiedlicher Anbieter auf dem Markt vertreten sind, gibt es im Bereich der Pflanzenbewässerungssysteme zum heutigen Stand der Technik kaum fertige Lösungen zu erwerben. Unter Betrachtung einer Statistik über die Beliebtheit von Pflanzen stellt sich heraus, dass etwa 74% der Befragten mindestens eine Zimmerpflanze besitzen. Vgl. [3] Der Zusammenhang zwischen Smart Home und der Beliebtheit von Pflanzen lässt sich mittels einer weiteren Umfrage ziehen. So gibt etwa die Hälfte aller Befragten von sich an, kaum bis kein Talent bei der Lebenserhaltung von Pflanzen vorzuweisen. Vgl. [4] Dies lässt sich damit in Verbindung bringen, dass jede Pflanze spezifischen Anforderungen wie beispielsweise fest definierte Gießzyklen benötigt. Somit lässt sich auf Basis dessen ein Interesse für ein automatisiertes System herstellen.

Ziel dieser Arbeit soll daher die Entwicklung eines automatisierten Pflanzenbewässerungssystems auf Basis eines Arduino in Form einer Smartphone-App sein. Der Fokus soll dabei auf einem automatisierten Gießzyklus sowie einer Datenübersicht über die erfolgten Gießvorgänge liegen. Über die verbauten Sensoren soll die Feuchtigkeit der Erde sowie der Füllstand des Behälters ermittelt werden. Außerdem findet mittels der Pumpe bei Bedarf und vorhandenem Füllstand die Pflanze eine automatisierte Bewässerung statt. Zusätzlich erfolgt die Übermittlung der Messdaten und Fehlermeldungen mittels WLAN und wodurch diese auf dem Endgerät angezeigt werden können.

Im Verlauf der Bearbeitung wird im ersten Teil der derzeitige Stand der Technik näher betrachtet und anschließend die Herangehensweise der Realisierung klar beschrieben. Anschließend wird die Inbetriebnahme des Systems sowie die Funktionsweise geprüft. Darüber hinaus wird die Grundlage für den Datenaustausch zwischen App und Arduino geschaffen.

1 Stand der Technik

Wie bereits in der Einleitung hingeführt, konnte der Bereich Smart Home in den letzten Jahren schnell sowohl aufgrund der hohen Anfrage aber auch aufgrund des schnellen Wandels der Technologie expandieren. Vorteile wie Datenüberwachung und der Automatisierung von Prozessen, durch die häufig eine Effizienzsteigerung resultiert, wurde das Prinzip von Smart Home auch in den Industriellen Sektoren angepasst. „Smart“ steht somit stellvertretend für die Eigenschaften der neuen Möglichkeiten. Zusätzlich wird dem Begriff jeweils ein Sektor zugewiesen. So werden beispielsweise Technologien in der Industrie unter dem Begriff „Smart Factory“ oder in der Landwirtschaft unter dem Begriff „Smart Farming“ zugeordnet. In diesem Absatz werden neue Innovationen des „Smart Farmings“ sowie des „Smart Homes“ unter Betrachtung des Aspekts der Pflanzenbewässerung vorgestellt. Abschließend wird eine Abgrenzung der Lösung der Studienarbeit zu bereits bestehenden Projekten hergestellt.

1.1 Aktuelle Technologien im Bereich Smart Farming

In der Agrar-Industrie stellt sich „Smart Farming“ oder auch „Precision Farming“ als das Werkzeug zur örtlichen und zeitlichen Optimierung heraus. Mit der Digitalisierung der Landwirtschaft mittels Bewässerungssystemen oder innovativen Erntemaschinen rückt die Überwachung der Pflanzen immer mehr in den Fokus. [5] Mittels Sensoren wie beispielsweise zur Erfassung der Bodenfeuchte oder Beschaffenheit kann die gesamte Ernte dauerhaft überwacht und dadurch die optimalen Bedingungen für unterschiedliche Pflanzen sichergestellt werden. Der Einsatz von neuen Technologien ermöglicht eine Erstellung von 3D-Bodenkarten zur Ermittlung der Bearbeitungstiefe. Zusätzlich können die Daten in Echtzeit über Cloud-Dienste für die Landwirte jederzeit einsehbar sein. Gerade der Aspekt der Datenerfassung über die Cloud löst Probleme wie Harmonisierungsprobleme der Datenformate und ermöglicht ein Outsourcen der Datenverarbeitung über Dienstleister. Vgl. [6, S. 33f] Auf weitere Aspekte wie KI-Unterstützung und Datensicherheit wird Verlauf im Bericht nicht tiefer eingegangen.

1.2 Smart Home Systeme zur Bewässerung von Pflanzen

Auch im Bereich Smart Home gibt es bereits Lösungen, um die eigenen Zimmerpflanzen ohne großen Aufwand am Leben zu erhalten. Durch Automatisierung entfällt beispielsweise das Pflegen von Gießplänen sowie die direkte Bewässerung. Durch eine Gegenüberstellung vorhandener Produkte in diesem Gebiet sollen die Unterschiede der Systeme herausgearbeitet werden.

In Anhang 1 ist tabellarisch eine Übersicht der verschiedenen Bewässerungssysteme dargestellt. Hierfür wird von jeder Produktkategorie ein repräsentatives Produkt auf dem Markt ausgewählt und entsprechend der vom Hersteller angegebenen Daten ausgearbeitet.

Bei dem System von Bluma handelt es sich um einen Tonkegel. Dieser gibt aufgrund der Materialbeschaffenheit nur Wasser an die Erde ab, wenn diese zu trocken wird und ist somit eine natürliche und energiefreie Lösung für eine automatische Bewässerung. Durch die Variante mit dem integrierten Schlauch befeuchtet sich das Material selbst über den angrenzenden Behälter. Es ist jedoch zu beachten, dass der Füllstand des Behälters von dem Anwender regelmäßig selbst zu überprüfen ist, um einen reibungslosen Gießzyklus zu gewährleisten.

Produkte wie der selbstgießende Blumentopf von Lazy Leaf fungieren als selbstgesteuertes direktes Bewässerungssystem. Dabei verspricht der Hersteller eine automatisierte, im Blumentopf integrierte Pumpenlösung inklusive Wasserbehälter, bei dem die Pflanze entsprechend der eingestellten Timerfunktion und Gießmenge bewässert werden kann. Dieses Produkt besitzt 10 unterschiedliche Modi, mit denen alle gängigen Pflanzenbedürfnisse abgedeckt werden sollen. Dazu muss der Anwender die Grundbedürfnisse der zu bewässernden Pflanze einschätzen können, um eine richtige Auswahl des Modus zu treffen. Zudem ist auch hier eine automatische Meldung für das Nachfüllen des Wasserbehälters nicht gegeben, wodurch auch hier eine manuelle Überprüfung dessen notwendig ist.

Hinsichtlich der Sensortechnik gibt es in Bezug auf die Pflanzenbewässerungssysteme viele neue Technologien. Bei dem KI-angebundenen, smarten Pflanzensensor von FYTA können die Daten wie beispielsweise Bodenfeuchte der zu überwachenden Pflanze mittels Bluetooth oder auch WLAN an eine App übermittelt werden um jederzeit Echtzeitdaten oder auch Langzeitdaten abrufen zu können. Mittels einer am Sensor angebrachten Kamera und einer KI-auswertbaren Datenbank, können so

Nährstoffmängel der Pflanze diagnostiziert werden. Der Gießvorgang ist bei diesem System nicht vorgesehen und ist entweder manuell oder durch eine andere Produktlösung durchzuführen.

Ein weiterer Trend der Pflanzenbewässerungssysteme sind die sogenannten DIY-Kits. Dabei handelt es sich um eine vorgefertigte Auswahl an Hardware Komponenten für den Anschluss und die Steuerung über einen Mikrocontroller. Diese Art der Lösung entspricht zum größten Teil der im Bericht angewendeten Lösung. Dabei ist anzumerken, dass für die Installation ein technisches Verständnis für die Verdrahtung der unterschiedlichen Komponenten, für die Programmierung des Mikrocontrollers sowie für die Kalibrierung der Sensoren und Aktoren von Nöten ist.

Bei der Untersuchung der unterschiedlichen Produkte ergibt sich die Fragestellung, warum keine bestehende Lösung zur Steuerung über den Mikrocontroller gibt. Dies lässt sich mittels einer Betrachtung des Objekts Zimmerpflanze feststellen. Die Individualität jeder einzelnen Pflanze und ihrer Umgebung kann nur schwer über ein bestehendes System ausgewertet werden. Dazu zählen zum einen Grundbedürfnisse der Pflanze, wie beispielsweise der Wasser- oder Nährstoffbedarf, die je nach Art und Größe der Pflanze andere Werte erwarten. Andere Faktoren sind die Größe des Blumentopfs oder die Dichte und Art der verwendeten Bodenerde. Aufgrund der hohen Abweichungen der einzelnen Faktoren kann ohne KI-Unterstützung somit nur schwer eine allgemeine Lösung für alle Pflanzenarten geschaffen werden.

2 „Hardware“-Aufbau

Im folgenden Absatz soll der Hardware-Aufbau für das Pflanzenbewässerungssystem beschrieben werden. Dazu werden die unterschiedlichen Hauptkomponenten analysiert, die Auswahlkriterien festgelegt und anschließend eine Gegenüberstellung unterschiedlicher Produkte durchgeführt. Im nächsten Schritt werden die Verdrahtung sowie die Realisierung des Gehäuses ausgearbeitet.

2.1 Konzepterstellung und Komponentenübersicht

Der erste wichtige Meilenstein des Projekts ist die sogenannte Konzepterstellung. Hierzu wird im ersten Schritt die genaue Funktionsweise des Pflanzenbewässerungssystems definiert und anschließend alle benötigten Komponenten für die Realisierung ausgelegt. Der Umfang des Projekts soll dabei durch ein modulares System erweiterbar definiert werden, um die Möglichkeit zu bieten, das Projekt auch in weiteren Studienarbeiten weiter auszuarbeiten.

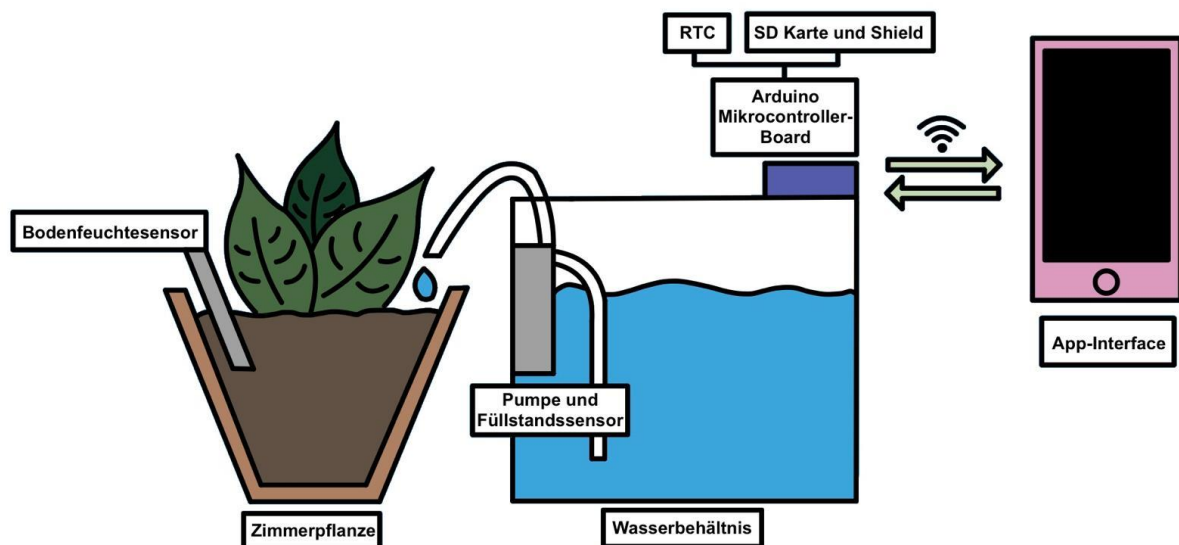


Abbildung 1: Übersicht Pflanzenbewässerungssystem

Quelle: Eigene Darstellung

In Abbildung 1 wird der erste grobe Entwurf des Pflanzenbewässerungssystems dargestellt. Der Fokus im Bereich Sensorik liegt dabei in der Erfassung der Bodenfeuchte und des Füllstands. Diese Daten sollen über die am Arduino angeschlossene Real-Time-Clock (RTC) stündlich erfasst und entsprechend verarbeitet werden. So wird bei zu geringer Bodenfeuchte über den Arduino die Pumpe

eingeschaltet und leitet damit den Gießzyklus ein. Bei zu geringem Füllstand des Wasserbehälters bleibt zum Schutz vor einem Defekt der Pumpe der Gießzyklus aus. Die erfassten Daten sowie gegebenenfalls erfolgte Gießzyklen und Fehlermeldungen sollen zusätzlich in eine JSON-Datei geschrieben und auf der MicroSD-Karte gespeichert werden. Diese dient als Datenbank für das App-Interface, in dem eine grafische Übersicht über den Zustand der Pflanze, des Systems sowie die Fehlermeldungen an den Anwender weitergegeben werden.

Auf Basis dieses Konzepts ergeben sich folgende Komponenten:

- Mikrocontroller inklusive zugehörigem Netzteil
- RTC
- MicroSD Karte und SD-Shield
- Pumpe inklusive Schläuche
- Bodenfeuchtesensor
- Füllstandsensor
- Wasserbehälter

Grundlegend wird sich hier für die Nutzung der RTC entschieden, um die Datenpakete einer festen Uhrzeit und einem festen Datum zuordnen zu können. Dies bietet den Vorteil, dass diese für die Anzeige im App-Interface genutzt werden können. Zusätzlich kann die Funktion ebenso für die stündliche Abfrage des potentiellen Gießzyklus verwendet werden. Die MicroSD-Karte wird verwendet, um stündlich ein aktualisiertes Skript an die App weiterzuleiten und somit eine externe Datenspeicherung wie beispielsweise einen Cloudservice zu vermeiden.

2.2 Auswahl des Mikrocontrollers + Vergleich

Das "Herzstück" des Schaltungsaufbaus ist der Mikrocontroller. Über diesen findet das Auslesen und Speichern der sensorspezifischen Daten statt. Zusätzlich fungiert er als Kommunikationsschnittstelle zwischen Bewässerungssystem und dem App-Interface. Dementsprechend soll zur Auswahl des am besten geeigneten Mikrocontrollers zunächst über eine Gegenüberstellung der für das Projekt geeigneten Controller stattfinden. Dabei wird jeweils das aktuelle Modell der 3 gängigsten Mikrocontrollerhersteller untersucht. Zusätzlich wird der Arduino Nano 33 BLE Sense

mit in die Analyse aufgenommen, da dieser bereits integrierte Sensorik enthält, die auf die Funktionalität bei dem Projekt geprüft werden soll. Hierzu ist es von hoher Bedeutung alle für das Projekt relevanten Merkmale auszumachen.

Tabelle 1: Marktanalyse Mikrocontroller – die relevanten Merkmale im Vergleich

Quellen: Vgl. [7], [8], [9], [10]

Kriterium	Arduino Uno R4 WiFi	ESP32	Arduino Nano 33 BLE Sense	Raspberry Pi Pico W
Preis	Günstig	Sehr günstig	Mittel	Günstig
Einsteigerfreundlichkeit	Sehr hoch	Mittel	Hoch	Mittel
GPIO-Pins	14	34	14	26
Analog-Eingänge	6	18 (mit Einschränkungen)	8	3
Stromverbrauch	Niedrig	Mittel	Niedrig	Niedrig
WLAN-Fähigkeit	Ja (integriert)	Ja	Ja	Ja
Betriebsspannung	5 V (ESP32-S3: 3,3 V)	3,3 V	3,3 V	3,3 V
Programmiersprache	C/C++ (Arduino IDE)	C/C++, MicroPython	C/C++ (Arduino IDE)	C/C++, MicroPython, CircuitPython
Betriebssystem	Kein OS (Bare-metal)	FreeRTOS	Kein OS (Bare-metal)	Kein OS (Bare-metal)
Community-Support	Sehr groß	Groß	Groß	Wachsend

Die in Tabelle 1 aufgelisteten Aspekte lassen sich mit den Anforderungen des Systems gegenüberstellen. Im ersten Schritt lässt sich feststellen, dass sich theoretisch alle Optionen für die Realisierung eignen, da keine großen Abweichungen in der allgemeinen Struktur bestehen. Alle Module sind WLAN-fähig und bieten grundsätzlich die Anforderungen an das System.

Die Auswahl des Mikrocontrollers wird so gewählt, dass die Anzahl der zusätzlich benötigten Komponenten sich auf das Minimum bezieht. Auswahlkriterium war in diesem Fall die höhere Betriebsspannung des Arduino Uno r4 im Gegensatz zu den anderen Mikrocontrollern. Da die Pumpe sowie die Sensorik mit einer Versorgungsspannung von 5V erwarten, ist keine zusätzliche Spannungsquelle zu integrieren. Zusätzlich bietet die Auswahl keine relevanten Einschränkungen. Die im Arduino Nano 33 BLE Sense verbaute Sensorik bietet beispielsweise in diesem Anwendungsfall keinen Mehrwert. Ebenso sprechen Kriterien wie die hohe Einsteigerfreundlichkeit und der Community Support für die Nutzung eines Mikrocontrollers der Arduino Familie.

2.3 Verwendete Sensoren/Aktoren

Wie bereits in der Konzepterstellung in 2.1 aufgezählt, wird für die Realisierung ein Bodenfeuchtesensor, ein Füllstandsensor sowie eine Pumpe benötigt. Dazu werden die für den jeweiligen Anwendungsfälle betrachtet und auf Basis die relevanten Kriterien ausgearbeitet und mit den möglichen Sensoren und Aktoren abgeglichen.

2.3.1 Bodenfeuchtesensor

Für die Auswahl des Bodenfeuchtesensors ist die Bodenbeschaffenheit von hoher Relevanz. Die Messung sollte dabei möglichst im unteren Drittel des Blumentopfes erfolgen, um den Wert der Feuchtigkeit nahe den Wurzeln erfassen zu können. Grundsätzlich ist zu beachten, dass je nach Messposition mit hohen Wertabweichungen zu rechnen ist. Auch die Zusammensetzung der Erde kann dabei zu unterschiedlichen Messabweichungen führen. Unabhängig von der Wahl des Sensors ist es von hoher Bedeutung, den Schwellenwert für die Bewässerung sorgfältig einzustellen. Vgl. [11], [12], [13]

Für Messungen der Feuchtigkeit in Bezug auf Arduino-Projekte finden primär kapazitive und resistive Sensoren ihre Anwendung. Aufgrund ihrer geringen Kosten fällt häufig der resistive Widerstand in den Fokus. Dieser misst den elektrischen Widerstand im Boden, der abhängig vom Feuchtigkeitsgehalt der Erde ist. Als Gegenargument für die Verwendung lässt sich der Aspekt Langlebigkeit aufzählen. Diese ist beim resistiven Widerstand häufig aufgrund von Korrosion stark eingeschränkt. Diese wirkt sich ebenfalls auf die Messwertgenauigkeit aus. Der kapazitive Widerstand gilt dabei als deutlich langlebiger und eignet sich daher besser für die Anwendung. Vergleichsweise ist dieser etwas teurer als der resistive Widerstand, bietet jedoch zusätzlich eine höhere Präzision und Korrosionsbeständigkeit, da das Material nicht direkt mit dem Substrat in Berührung kommt. Die Messung erfolgt beim kapazitiven Widerstand über eine Erfassung der dielektrischen Eigenschaften des Bodens, die ebenfalls je nach Feuchtigkeitsgrad variieren. Zusammenfassend lässt sich sagen, dass der kapazitive Bodenfeuchtesensor die beste Wahl für ein zuverlässiges Pflanzenbewässerungssystem mit Arduino ist. Vgl. [14], [15], [13]

2.3.2 Füllstandsensor

Der Füllstandsensor wird benötigt, um sicherzustellen, dass der Behälter ausreichend gefüllt ist. Damit wird zum einen die Pumpe vor einem Defekt durch das Ansaugen von Luft geschützt, andererseits können die Daten auch zur Verbesserung des Prozesses beitragen. Mittels einer Meldung kann der Anwender dazu aufgefordert werden, den Wassertank zu befüllen, falls der Füllstand zu niedrig wird. Damit wird sichergestellt, dass der Gießzyklus erfolgreich abläuft und die Pflanze bewässert wird.

Es gibt unterschiedliche Möglichkeiten den Füllstand eines Behälters zu erfassen. Der gängigste dafür ist der sogenannte Schwimmer-Reedschalter, der aus einem Schwimmer besteht, der einen elektrischen Kontakt auslöst, sobald eine Wasserstandsgrenze erreicht wird. Auch wenn das System als besonders zuverlässig gilt, handelt es sich bei dieser Variante um eine Zustandserkennung und nicht um eine Messung über einen Zeitraum. Eine weitere Möglichkeit zur Erfassung sind die photoelektrischen Sensoren. Diese senden ein Lichtsignal aus und erfassen über die Zeit den zurückgelegten Weg. Auch diese Möglichkeit gilt als besonders zuverlässig. Für Arduino-Projekte kommen häufig Bodenfeuchtesensoren als Füllstandsensoren zum Einsatz. Besonders kapazitive Sensoren werden häufig verwendet und bieten präzise Messergebnisse. Vgl. [13], [16]

Auf Basis dieser Erkenntnisse und Recherche nach geeigneten Modellen wird sich für eine Lösung entschieden, bei der es sich um eine Pumpe mit integriertem kapazitivem Sensor handelt. Diese Lösung überzeugt vor allem aufgrund des geringen Platzes und geringem Anschlussaufwand.

2.4 Schaltungsaufbau

Auf Basis der in Kapitel 2.1 aufgezählten Komponenten wird im folgenden Abschnitt die finale Realisierung beschrieben. Dabei wird die Herangehensweise sowie aufkommende Problemstellungen sowie die Lösung dieser beschrieben.

Tabelle 2: Auflistung der verwendeten Komponenten

Quelle: Eigene Darstellung

	Funktion	Bauteilbezeichnung	Eigenschaften
1	Mikrocontroller	Arduino Uno r4	Datenblatt [7]
2	RTC	RTC DS1307	Datenblatt [17]
3	MicroSD-Karte und SD-Shield	DEBO MICROSD 2 INTENSO MSDHC4G MicroSDHC	4GB
4	Bodenfeuchtigkeitssensor	Capacitive Soil Moisture Sensor V2.0	5V Versorgungs- spannung, analoger Ausgang 0 – 3 V
5	Pumpe inklusive Füllstandsensoren	M5 Stack U101	5V Versorgungs- spannung, 5 W Pumpenleistung
6	Spannungswandler	Joy-it B-POW Entwicklerboard	Spannungsversorgung von 5V/3,3V
7	Breadboard	-	Half-Board
8	Anschlussleitungen	-	Stecker-Stecker, Buchse Buchse, Adapter Buchse auf Stecker
9	Netzteil	AC to DC Power Adapter X0014365KT	9V, 500mA 1000mA
10	Wasserbehälter	Eimer mit Deckel Kunststoffeimer	5l Fassungsvermögen, 22,9L x 19,5B x 18,2 H cm

Um den Schaltungsaufbau realisieren zu können, werden in Tabelle 2 alle verbauten Komponenten aufgelistet. Dabei handelt es sich um die finale Auswahl. Auf Basis der in Kapitel 2.2 und Kapitel 2.3 ausgearbeiteten Erkenntnisse fällt die Wahl auf den Arduino Uno r4 (1) und die beiden kapazitiven Sensoren (4) (5). Obwohl im Arduino bereits eine RTC verbaut ist, wird eine externe RTC (2) angeschlossen, da diese zusätzlich mit einer Batterie gespeist werden kann. Dies dient dazu, im Falle eines Stromausfalls die Zeit nicht neu kalibrieren zu müssen. Wie auch bereits in der

Konzeptausarbeitung dargestellt, sollen die Daten auf der am Arduino angeschlossenen MicroSD-Karte (3) abgespeichert werden. Dafür wurde sich für die MicroSD-Karte 4 GB entschieden.

Beim Anschluss der RTC und SD-Karte und einem Testprogramm, dass die aktuelle Zeit auf der MicroSD-Karte hinterlegt, ergab sich eine Problematik mit der Spannungsversorgung des Arduinos. Das Testprogramm gab falsche Werte der RTC zurück. Beim weiteren Anschließen eines Sensors konnte das Programm die MicroSD-Karte nicht mehr identifizieren und gab eine Fehlermeldung zurück. Als Lösung ergibt sich eine externe Spannungsversorgung. Das ursprünglich für den Arduino verwendete Netzteil (9) wird mittels eines Spannungswandlers (8) auf die von den Komponenten geforderte Versorgungsspannung von 5V geregelt.

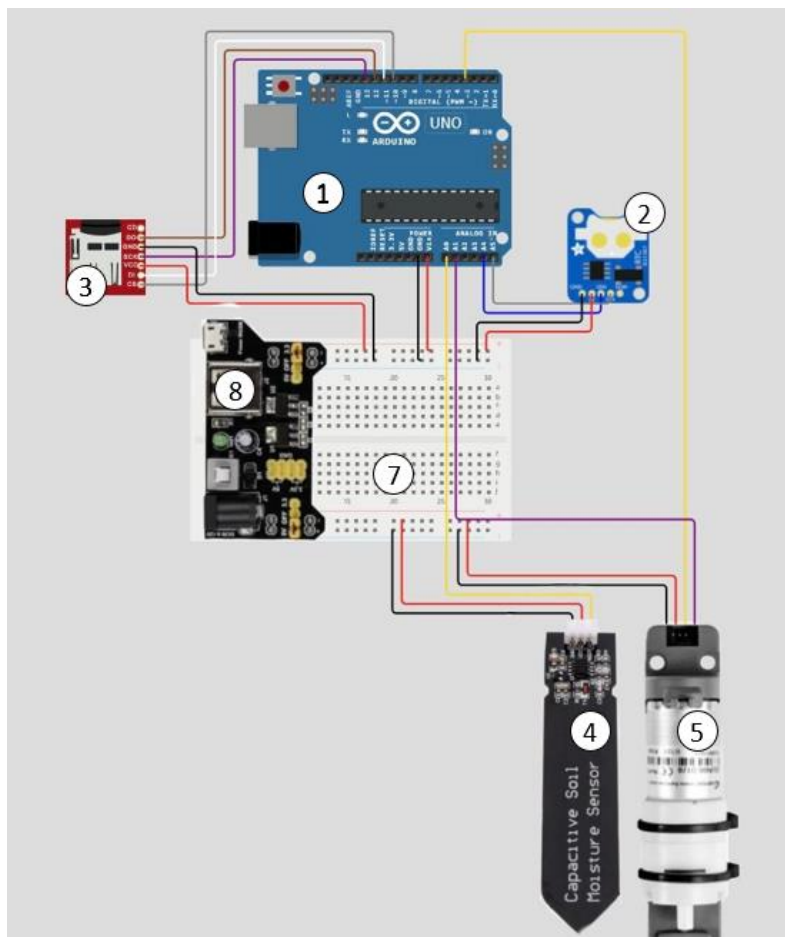


Abbildung 2: Schaltungsaufbau des finalen Pflanzenbewässerungssystems

Quelle: Eigene Darstellung realisiert über WOKWI [18]

In Abbildung 2 wird der Schaltungsaufbau grafisch dargestellt. Die Bauteile sind zur Zuordnung entsprechend der in

Tabelle 2 verwendeten Nummerierungen beschriftet. Die farblichen Anschlussleitungen sind so gewählt, dass sie den realen Schaltungsaufbau widerspiegeln. Die Sensoren und Aktoren sowie der Spannungswandler werden der Grafik hinzugefügt, da diese nicht im Tool auswählbar sind.

Tabelle 3: Pinanschlüsse für finalen Schaltungsaufbau

Quelle: Eigene Darstellung

Bauteil	Pin	Anschluss
(1)	VIN	5V
	GND	GND
(2)	VCC	5V
	GND	GND
	SDA	A4
	SCK	A5
(3)	VCC	5V
	GND	GND
	SCK	13
	MISO	12
	MOSI	11
	CS	10
(4)	VCC	5V
	GND	GND
	AOUT	A0
(5)	VCC	5V
	GND	GND
	IN	3
	OUT	A1

Zusätzlich werden in Tabelle 3 die Anschlüsse festgehalten. Dabei wurden die Pins jeweils der Komponente zugeordnet, um eine klare Strukturierung zu schaffen. Der Arduino (1) wird über das Breadboard an die Versorgungsspannung und Masse angeschlossen. Die RTC (2) wird ebenfalls über das Board bestromt, wird jedoch zusätzlich über die beiden Analogpins A4 und A5 am Arduino angeschlossen. Dabei wird eine I2C-Verbindung hergestellt. SCK ist dabei die Serial Clock und stellt die Taktzeit ein, während SDA für den Ausdruck Serial Data steht und für die

Datenübermittlung zuständig ist. Die SD-Karte wird in das SD-Shield (3) gesteckt und dieses ebenfalls an die Versorgungsspannung angeschlossen. Zusätzlich werden die restlichen Pins entsprechend der Tabelle angeschlossen. Die Bezeichnung MISO (Master Input Slave Output) ist dabei mit dem Arduino Datenblatt hinterlegten Begriff CIPO (Controller In Peripheral Out) gleichgesetzt. Für die Sensoren wird jeweils der Analogausgang (AOUT/ OUT) auf einen der freien Analogpins gesetzt und entsprechend die Bestromung angepasst. Der zusätzliche Pin an der Pumpen-Sensor-Komponente mit der Bezeichnung IN ist für die Ansteuerung der Pumpe und wird über einen freien Digitalpin (Pin 3) angeschlossen.

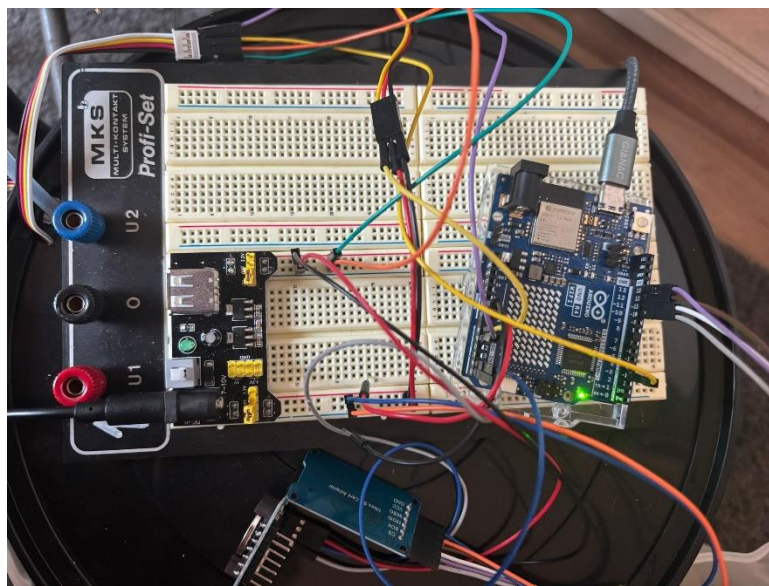


Abbildung 3: Realer finaler Schaltungsaufbau

Quelle: Eigene Darstellung

Der reale Schaltungsaufbau wird in Abbildung 3 abgebildet. Durch die Verwendung von Steckverbindung stellt sich das System als nicht portabel heraus. Zudem baut die Lösung zu diesem Zeitpunkt auf Funktionalität und bietet in Bezug auf das Design klaren Verbesserungsbedarf. Eine mögliche Lösung wäre beispielsweise ein 3D-gedruckte Version eines Deckels, in dem alle modularen Komponenten geschützt und befestigt sind.

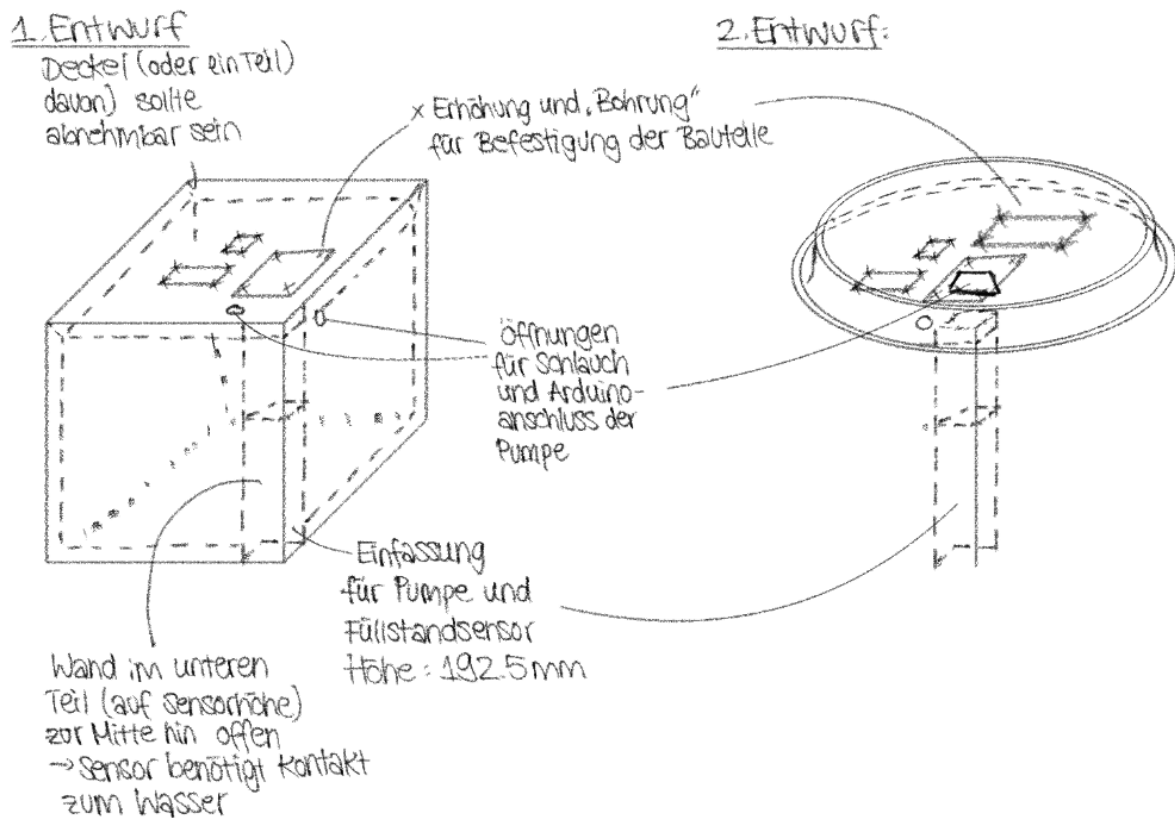


Abbildung 4: Grobe Skizzen Wasserbehälter für Pflanzenbewässerungssystem

Quelle: Eigene Darstellung

Aufgrund des zeitlichen Rahmens des Projekts finden in diesem Bereich erste Entwürfe sowie erste Expertengespräche statt. Der erste Entwurf (Abbildung 4 links) stellt dabei einen gesamten Behälter dar, der einen abnehmbaren Deckel beinhaltet. Die wichtigste Komponente des Konstrukts ist dabei die Einfassung zur Befestigung der Pumpe. Diese sorgt für eine stabile Position und damit eine verbesserte Messumgebung für die Füllstandsmessung. Nach einem Austausch mit dem 3D-Druck-Experten der DHBW Mannheim, stellt sich heraus, dass ein gesamter Behälter sehr Kosten und Zeitintensiv in der Konstruktion ist und deshalb der Deckel alleine als Konstruktion geeigneter sei. Die daraus resultierende Skizze ist in Abbildung 4 rechts dargestellt.

2.5 Kalibrierung der Sensoren und Aktoren

Um im Programm die Sensorwerte verarbeiten zu können, werden die Extrembedingungen der jeweiligen Sensoren bestimmt. Damit ist beim Bodenfeuchtesensor der Zustand der ausgetrockneten Erde, das heißt 0% Feuchtigkeit, sowie durchnässter Erde als 100% Feuchtigkeit zu definieren. Beim

Füllstandsensor werden die Grenzwerte 0% als „Behälter leer“ und 100% als „Behälter gefüllt“ festgelegt.

```
// Pin-Definition
const int sensorPin = A0; // Analog Pin für den Sensor
// A0 für Bodenfeuchtesensor; A1 für Füllstandsensor

// Variablen für Zeitsteuerung
unsigned long previousMillis = 0; // vorherige Zeit
const unsigned long interval = 30000; // Intervall: 30 Sekunden (in
Millisekunden)

void setup() {
  // Serielle Kommunikation starten
  Serial.begin(9600);
  Serial.println("Sensor-Testprogramm gestartet");

  // Sensor-Pin konfigurieren
  pinMode(sensorPin, INPUT);
}

void loop() {
  // Aktuelle Zeit erfassen
  unsigned long currentMillis = millis();

  // Überprüfen, ob 30 Sekunden vergangen sind
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis; // Zeit zurücksetzen

    // Sensorwert lesen
    int sensorValue = analogRead(sensorPin);

    // Sensorwert auf der seriellen Konsole ausgeben
    Serial.println(sensorValue);
    // Die Ausgabe wird anschließend in eine .txt-Datei kopiert
  }
}
```

Listing 1: Testprogramm zur Messdatenerfassung zur Definition der Grenzwerte

Quelle: Eigene Darstellung

Um einen möglichst genauen Wert zu ermitteln, wird über das in Listing 1 dargestellte Testprogramm mittels dem Arduino und den angeschlossenen Sensoren in Abständen von 30 Sekunden mehrere Messwerte erfasst. Diese werden anschließend in einer Textdatei für jeden Zustand gespeichert und abgelegt. Mittels einem python-Skript werden anschließend die Messergebnisse aus der Textdatei gelesen, der Mittelwert errechnet und die Messreihe sowie der Mittelwert grafisch ausgegeben. Der Code sowie alle erfolgten Messungen sind in Anhang 2 bis Anhang 5 angehängt.

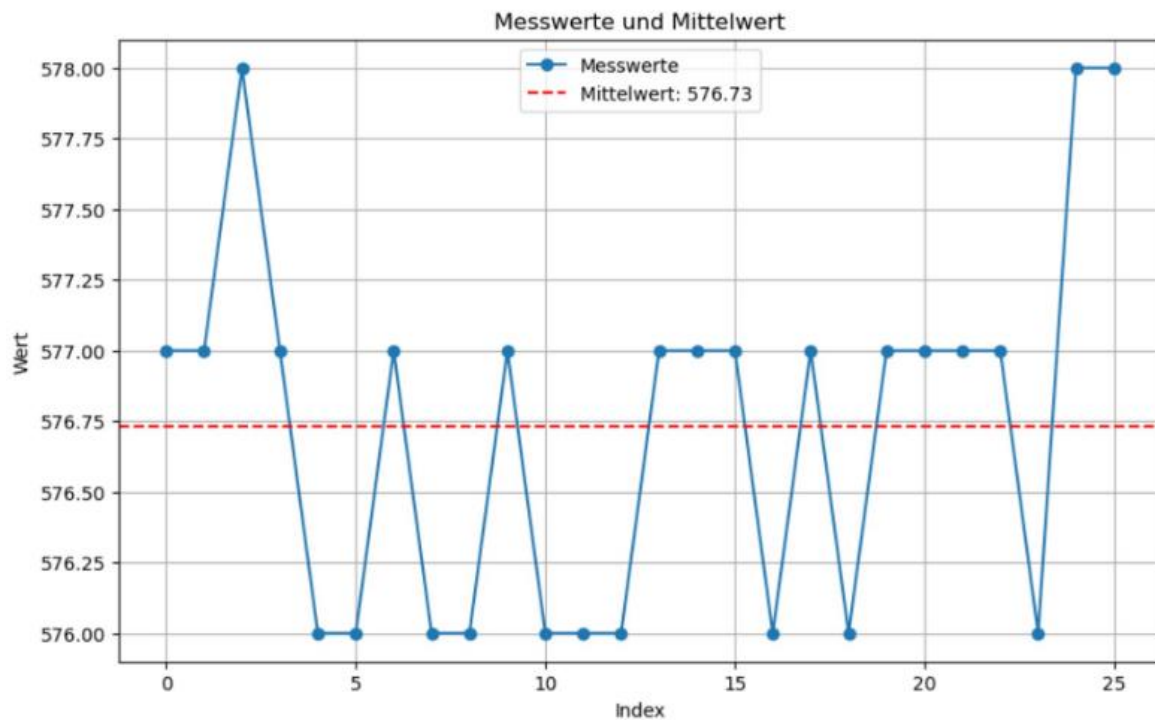


Abbildung 5: Messwerte zur Grenzwertfassung des Bodenfeuchtesensors (A0) - Zustand trocken

Quelle: Eigene Darstellung

In Abbildung 5 ist die Messreihe für den Bodenfeuchtesensor im absolut trockenen Zustand dargestellt. Auf der x-Achse sind dabei die Indizes der Messzeitpunkte aufgeführt, d.h. der Wert 1 entspricht 30 Sekunden. Auf der y-Achse sind die Analogwerte des Sensors aufgeführt. Dieses Schema wird auch auf die anderen Messungen angewendet. Für die Messung wird der Sensor in ausgetrockneter Erde angebracht. Die Messwerte sind stabil und bewegen sich in einer maximalen Abweichung von 1,25 Einheiten. Im Gegensatz dazu ist bei der Messung des durchnässten Zustands (Anhang 3) aufgrund der Unregelmäßigkeit von Flüssigkeiten deutlich schwankender. Um dennoch einen präzisen Mittelwert zu erfassen, werden in diesem Fall mehr Messzeitpunkte durchgeführt. Die in Anhang 4 und Anhang 5 abgebildeten Graphen zeigen die ermittelten Werte für den leeren und vollständig gefüllten Behälter an. Bei einer genaueren Untersuchung der letzten Messung fällt jedoch auf, dass die Messabweichung nicht realistisch ist, da eine Abweichung von fast 100 Einheiten vorliegt. Auch bei mehrfachen Wiederholungen der Messungen ergeben sich starke Abweichungen ohne eine Veränderung der Randbedingungen. Dementsprechend ist hier von einem Sensordefekt auszugehen und die Messung nach Austausch erneut durchzuführen.

Tabelle 4: Ermittelte Mittelwerte der Grenzwerte des Bodenfeuchte- und Füllstandsensors

Quelle: Eigene Darstellung

Sensor	Unterer Grenzwert (0%)	Oberer Grenzwert (100%)
Bodenfeuchte	576,73	243,37
Füllstand	387,23	324,25 (ungültig)

Zur Übersicht sind in Tabelle 4 alle errechneten Mittelwerte aufgelistet. Die Erkenntnis aus den Werten ist, dass es sich um eine umgekehrte Skalierung handelt. Das bedeutet, dass der untere Grenzwert den größeren Zahlenwert enthält als der obere Grenzwert. Dies muss bei der prozentualen Umrechnung des Messwerts beachtet werden. Arduino IDE bietet dazu die sogenannte map-Funktion. Damit lassen sich die erfassten Grenzwerte in Prozentwerte umformen und der neue gemessene Wert auf die Skala einfügen.

Tabelle 5: Durchflussbestimmung Pumpe

Quelle: [19, S. 18]

ID	Zeit [s]	Gewicht [g]	Durchfluss [ml/s]	Bemerkung
1	10	40,8	4,08	Messung mit Software
2	30	113,7	3,79	Messung von Hand
3	30	118,3	3,94	Messung von Hand
4	30	116,5	3,88	Messung von Hand
5	30	113,7	3,79	Messung von Hand
6	30	120,5	4,02	Messung mit Software
7	60	230,7	3,85	Messung von Hand
8	60	248,5	4,14	Messung mit Software
9	60	246,2	4,10	Messung mit Software
10	60	243,5	4,06	Messung mit Software

Für die Messung der Durchflussmenge werden die Werte aus der Studienarbeit des vorigen Jahrs übernommen, da ein identisches Pumpenmodell verwendet wird. Die Ergebnisse sind in Tabelle 5 zusammengefasst. Der ermittelte Durchschnittswert von 4,08 ml/s wird als Grundlage für die Dauer des Gießzyklus bestimmen.



Abbildung 6: Links: Gesamtaufbau des Pflanzenbewässerungssystems; Rechts: Installation der Pumpe im Wasserbehälter

Quelle: Eigene Darstellung

Abschließend wird in Abbildung 6 links der gesamte Hardwareaufbau abgebildet. Der Deckel des Wasserbehälters ist so präpariert, dass die elektronischen Anschlüsse sowie der Pumpschlauch, der in die Erde der Pflanze verlegt wird, nach außen gelangen können. Im rechten Bild ist die Installation der Pumpe und des Pumpschlauchs im Inneren des Behälters dargestellt.

3 Programmablauf

Durch den abgeschlossenen Hardware-Aufbau kann im nächsten Schritt das Programm auf seine Funktion getestet werden. Da diese Studienarbeit auf der Grundlage eines bereits existierenden Codes aufbaut, wird hierzu im ersten Schritt eine Abgrenzung zur vorigen Arbeit geschaffen. Anschließend wird auf die zwei Hauptfunktionen des Programms eingegangen. Die erste ist dabei der autonom ablaufende Gießzyklus. Die zweite Funktion bietet mit der Erstellung einer JSON-Datei auf der MicroSD-Karte die Grundlage für die App-Übermittlung. Im letzten Schritt wird mittels eines Funktionstests die Funktionsfähigkeit des Programms überprüft.

3.1 Abgrenzung zu voriger Studienarbeit

Um eine klare Abgrenzung zu schaffen, wird zunächst ein Überblick des bestehenden Systems erstellt, um die herausgearbeiteten Unterschiede anschließend deutlich zu machen.

Der Ersteller entschied sich für ein halbautomatisches System. Dies bedeutet, dass dem Benutzer mittels eines Joysticks die Steuerung zwischen einer automatisierten und einer manuellen Bewässerung ermöglicht wird. Die Navigation erfolgt dabei über mehrere Ebenen eines Displays, um zum einen die Auswahl des Bewässerungsmodus zu ermöglichen, aber auch um die aktuellen Sensordaten der Pflanze auslesen zu können. Vgl. [19, S. 23ff]

Im Gegensatz zu der bestehenden Arbeit handelt es sich bei dieser Ausarbeitung um eine vollautomatisierte Lösung. Aus diesem Grund ist für das System ein User-Interface wie zum Beispiel Joy-Stick sowie die programmtechnische Ebene nicht notwendig. Eine Verwendung des bestehenden Programmcodes ist daher nicht möglich und daher eine Erstellung eines modifizierten Codes unumgänglich. Einzelne Elemente des Codes, wie beispielsweise die in Kapitel 2.5 auf Seite 19 erwähnte map-Funktion, werden als Grundlage verwendet.

Weitere Unterschiede zeigen sich vor allem in der Realisierung des Gießzyklus. Im existierenden Code wird über das sogenannte case-Verfahren in unterschiedliche Zustände innerhalb des Gießzyklus differenziert. Dabei entsteht vor allem eine hohe Komplexität durch eine zusätzliche Wartedauer und der Möglichkeit eines wiederholten Pumpimpulses. Um einer Überwässerung entgegenzuwirken, wird das

System bei zu häufig erfolgtem Gießen über eine Fehlermeldung beendet. Vgl. [19, S. 22]

Bei einem vollautomatisierten System ist dahingegen ein System-Shutdown nicht wünschenswert, da dieses ohne menschliches Einwirken funktionieren soll. Der Aufbau des in dieser Ausarbeitung realisierten Gießzyklus wird daher im nachfolgenden Kapitel näher beschrieben.

3.2 Gießzyklus

Das Hauptmerkmal eines vollautomatisierten Bewässerungssystems ist ein funktionsfähiger und somit möglichst störungsfreier Gießzyklus. Dementsprechend wird bei der Realisierung auf einen einfachen, wenig verschachtelten Ablauf gesetzt um etwaige Störungsquellen zu vermeiden.

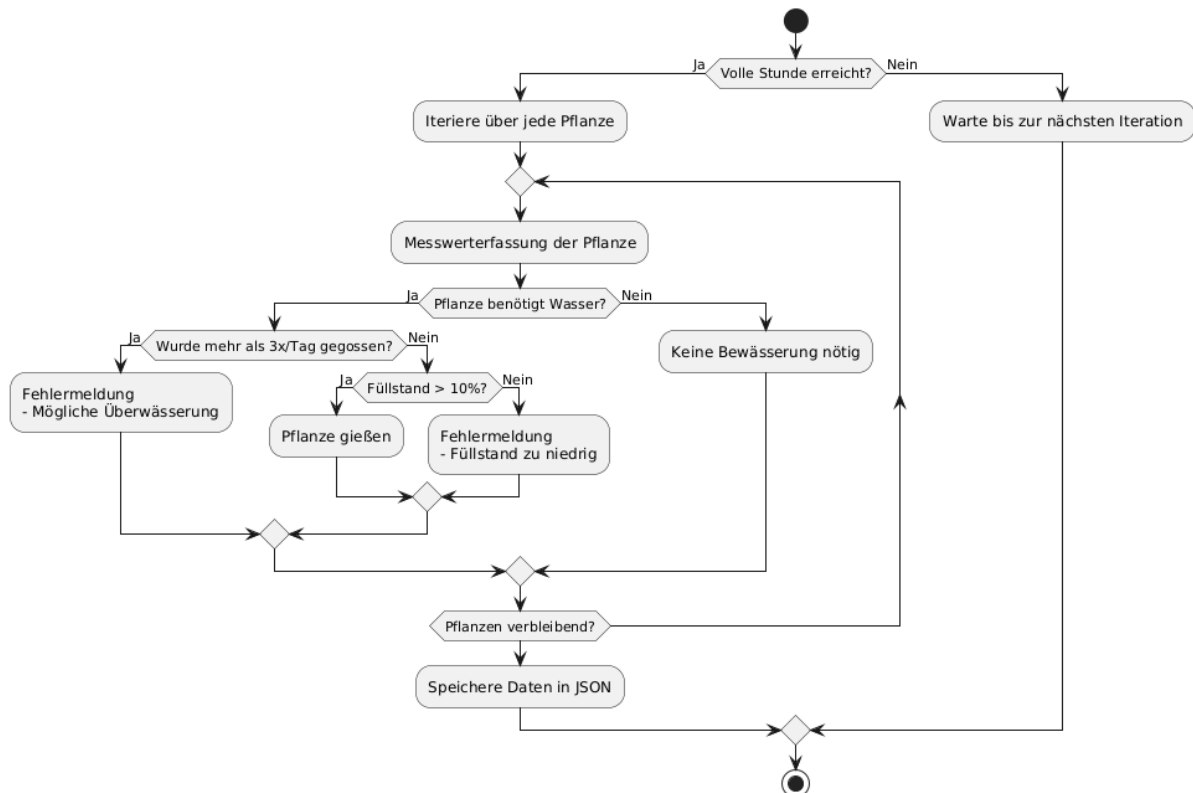


Abbildung 7: Ablaufdiagramm des Gießzyklus innerhalb der loop-Funktion

Quelle: Eigene Darstellung erstellt mit PlantText [20]

Da es sich bei dem Gießzyklus um einen sich wiederholenden Ablauf handelt, wird dieser durch die sogenannten loop-Funktion ausgeführt. Wie in Abbildung 7 dargestellt, ist das Erreichen der vollen Stunde Startbedingung für diesen Zyklus. Grundsätzlich ist dieser Vorgang modular für mehrere Pflanzen anwendbar. Dies wird

über eine zusätzliche Schleife im Ablaufdiagramm dargestellt. Im ersten Schritt wird der Messwert des Bodenfeuchtesensors erfasst. Sofern dieser unterhalb des definierten Schwellenwerts liegt, benötigt die Pflanze Wasser und soll folglich gegossen werden. Die Detektion eines Sensordefektes, welcher wie in Kapitel 3.1 beschrieben über einen Programmabbruch gelöst wird, wird in diesem Fall durch eine Begrenzung der täglichen Gießzyklen realisiert. Hierbei wird eine Begrenzung von maximal 3 Gießzyklen pro Tag definiert. Dies wird über einen Counter gelöst, der bei Beginn eines neuen Tages zurückgesetzt wird. Wird die Anzahl der Zyklen überschritten, wird innerhalb der JSON-Datei eine Fehlermeldung ausgegeben, die dem Benutzer mittels App auffordert, den Zustand seiner Pflanze zu überprüfen und auf einen möglichen Sensordefekt hinweist. Als letzte Bedingung für das Einschalten der Pumpe und damit dem Freischalten des Gießzyklus ist der Füllstand des Wasserbehälters. Sollte dieser zu niedrig sein, könnte die Pumpe Luft ansaugen und dadurch einen Defekt aufweisen. In diesem Fall wird ebenfalls eine Fehlermeldung ausgegeben, die den Benutzer dazu auffordert, den Füllstand des Behälters zu erhöhen. Sofern alle Bedingungen erfüllt sind, wird der Gießzyklus eingeleitet.

```
void loop() {
    static unsigned long lastUpdate = 0;

    // Prüfung zur vollen Stunde
    if (millis() - lastUpdate >= 3600000) { // Für den Funktionstest wird hier
5min (300000) verwendet
        lastUpdate = millis();

        for (Plant& plant : plants) {
            plant.updateMeasurements(); //Messwerterfassung

            // Gießlogik: Füllstand ok, Pflanze benötigt Wasser, Limit an
            Gießzyklen/Tag nicht überschritten
            if (levelSensor->readValue() > 10 && plant.needsWatering() &&
plant.canWaterMoreToday()) {
                plant.waterPlant();
            }
        }
        saveToJson(); // Speichern der aktualisierten Daten in die JSON-Datei
    }
}
```

Listing 2: loop-Funktion des Arduino-Programms

Quelle: Eigene Darstellung

In Listing 2 ist die loop-Funktion dargestellt. Wie im vorherigen Absatz beschrieben, lässt sich hier ebenfalls der im Ablaufdiagramm beschriebene Zyklus erkennen. An dem Code zeigt sich, dass die loop-Funktion durch das Ausgliedern von Funktionen

sehr kompakt und übersichtlich gestaltet ist. Durch eine klare Namensgebung der Funktionen wird sichergestellt, dass ein sofortiges Verständnis des Inhalts ohne Nachlesen möglich ist. Besonders auffällig ist hierbei die Verwendung von Klassen, wie in diesem Fall der Klasse Plant, in der die unterschiedlichen Funktionen eingegliedert sind. Dies hängt mit der Erstellung der JSON-Datei zusammen und wird im folgenden Kapitel ausführlich erläutert. In Anhang 6 ist die Klasse Plant sowie die in der loop-Funktion verwendeten Funktionen aufgeführt.

Wie bereits im vorigen Absatz beschrieben, ist ein essenzielles Element des Gießzyklus das Ansteuern der Pumpe. Deshalb wird das Vorgehen der Funktion im Folgenden ausführlich beschrieben.

```
// Pumpensteuerung
class Pump {
private:
    int pin; // Pin für Pumpensteuerung
public:
    Pump(int p) : pin(p) {
        pinMode(pin, OUTPUT);
        stop(); // Standardmäßig Pumpe aus
    }
    void start() { digitalWrite(pin, HIGH); } // Pumpe einschalten
    void stop() { digitalWrite(pin, LOW); } // Pumpe ausschalten
};
```

Listing 3: Klasse Pump

Quelle: Eigene Darstellung

Wie in Listing 3 dargestellt, wird auch für die Pumpe eine eigene Klasse erstellt, in der die Pinbelegung, sowie das Ansteuern fest definiert wird. Im Schaltungsaufbau von Tabelle 3 auf Seite 14 wird festgelegt, dass die Pumpe an den Digitalpin 3 angeschlossen wird. Da es sich dabei um eine unveränderliche Information handelt, wird die Pinbelegung in diesem Bereich auf das Zugriffsrecht „private“ gesetzt. Da die Funktionen zum Ansteuern der Pumpe innerhalb der Klasse Plant aufgerufen werden, wird sich hier für ein öffentliches Zugriffsrecht entschieden.

```
// Pflanze bewässern
void waterPlant() {
    DateTime now = rtc.now();
    String entry = formatTimestamp(now);
    watering.update(entry);
    pump.start(); // Pumpe starten
    delay(10000); // 10 Sekunden Bewässerung
    pump.stop(); // Pumpe stoppen
}
```

Listing 4: Funktion waterPlant innerhalb der Klasse Plant

Quelle: Eigene Darstellung

Innerhalb der Klasse Plant wird entsprechend Listing 4 auf die Funktionen der Klasse Pump zugegriffen und die Pumpe angesteuert werden. Für das Gießen wird zusätzlich für die JSON-Datei in *watering.update(entry)* der Zeitpunkt entsprechend dem in der Klasse Plant festgelegten Zeitstempel-Format zwischengespeichert und übergeben. Die Dauer des Pumpimpulses wird hierbei auf 10s definiert. Somit ergibt sich mit dem in Kapitel 2.5 angegebenen Durchfluss von 4,08ml/s eine Gesamtgießmenge von etwa 40ml.

3.3 Schreiben und Speichern der JSON-Datei

Als Grundlage für die App-Schnittstelle dient die Abspeicherung der Messdaten auf der SD-Karte. Wie bereits in der Konzepterstellung in Kapitel 2.1 festgelegt, wird sich hierbei für das Dateiformat JSON entschieden. Für die Struktur zur Verwendung in der App-Erstellung aber auch für das Abspeichern der Messwerte wird sich an einer Beispieldatei orientiert. Diese wird in Anhang 7 abgebildet. Um die in der JSON-Datei verwendeten Objekte strukturiert einfügen zu können, findet im gesamten Arduino Code die Klassenstruktur Verwendung. Damit ist es möglich einen strukturierten Code für die JSON-Datei zu erstellen.

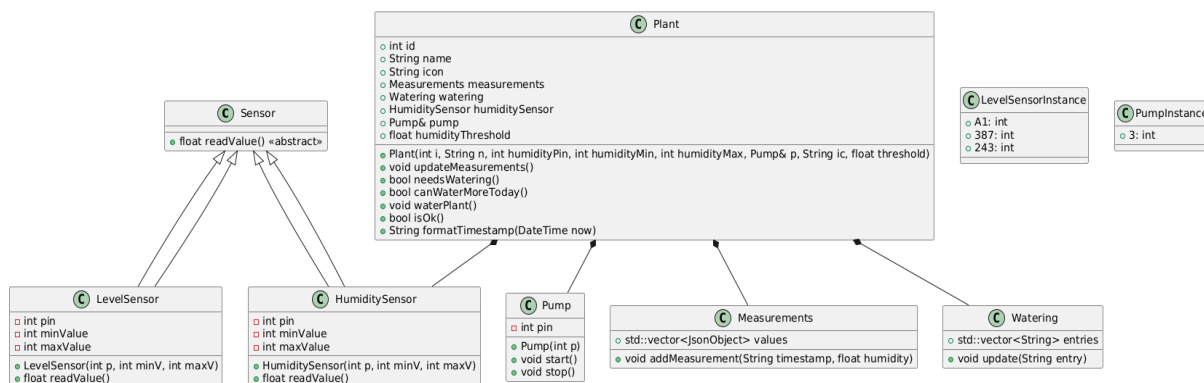


Abbildung 8: Übersicht der verwendeten Klassen im Arduino Code

Quelle: Eigene Darstellung erstellt mit PlantText [20]

Die im Code verwendeten Klassen werden in Abbildung 8 grafisch dargestellt. Dabei werden auch die zugehörigen Variablendeklarationen und Funktionen aufgelistet. Über die einfachen Pfeile werden Aufrufe der Klassen innerhalb einer anderen Klasse dargestellt. Die unausgefüllten Pfeile stellen die Vererbung dar.

Für das Schreiben der JSON-Datei werden hauptsächlich zwei Funktionen verwendet. Mit der im Anhang 8 aufgeführten `loadFromJson`-Funktion, werden die Informationen aus den bisherigen Messungen ausgelesen. Dies bietet die Grundlage für die `saveToJson`-Funktion. Leider war es nicht möglich in eine JSON-Datei zu schreiben, da hier selbst in einem einfachen Test-File eine Fehlermeldung auftritt. Aus diesem Grund werden die Daten im JSON-Format in eine TXT-Datei geschrieben, die anschließend in eine JSON-Datei umgewandelt wird.

```
void saveToJson() {
    StaticJsonDocument<2048> doc;
    JsonArray plantsArray = doc.createNestedArray("plants");

    for (Plant& plant : plants) {
        JsonObject plantObj = plantsArray.createNestedObject();
        plantObj["id"] = plant.id;
        plantObj["name"] = plant.name;

        JsonArray measurementsArray =
plantObj.createNestedArray("measurements");
        for (const JsonObject& m : plant.measurements.values) {
            measurementsArray.add(m);
        }

        JsonArray wateringArray = plantObj.createNestedArray("watering");
        for (const String& entry : plant.watering.entries) {
            JsonObject wateringEntry = wateringArray.createNestedObject();
            wateringEntry["entry"] = entry;
        }
        plantObj["icon"] = plant.icon;
    }

    [...]

    File file = SD.open(jsonFilePath, O_WRITE);
    if (file) {
        serializeJsonPretty(doc, file);
        file.close();
        Serial.println("JSON gespeichert.");
    } else {
        Serial.println("Fehler beim Öffnen der JSON-Datei zum Schreiben.");
    }
}
```

Listing 5: Ausschnitt aus `saveToJson`-Funktion

Quelle: Eigene Darstellung

In Listing 5 wird ein Ausschnitt der `saveToJson`-Funktion dargestellt. Im ersten Schritt wird ein statisches JSON-Dokument mit einem Speicher von 2048 Bytes erstellt. Anschließend wird mit dem Befehl `createNestedArray()` ein Array mit dem Namen „plants“ erstellt, in dem die unterschiedlichen Messwerte und Gießvorgänge abgespeichert werden. Mittels der `for`-Schleife wird über die potentiell vorhandenen Pflanzen iteriert. Dies ist für den modularen Aufbau des Codes essentiell. Innerhalb

der Schleife wird anschließend für jede Pflanze ein eigenes Objekt-Array erstellt, in dem entsprechend Anhang 7 die Grunddaten der Pflanze, wie Name und Identifikationsnummer, hinzugefügt werden. Für die Speicherung der Messwerte wird wiederum erneut ein Array erstellt, in dem für jede Messung der Zeitstempel sowie der zugehörige Messwert in Prozent angehängt wird. Dafür wird über die erfassten Messwerte der Pflanze in der Funktion *plant.measurements.values()* aus Anhang 6 iteriert. Für die erfolgten Gießzyklen wird im selben Schema vorgegangen. Unabhängig von der Pflanzenstruktur werden die möglichen Fehlermeldungen, die während dem Zyklus auftreten können, ebenfalls in einem Array abgespeichert. Dieser Code ist in Anhang 9 abgebildet. Die Besonderheit dabei, ist, dass die Fehlermeldungen aktiv entfernt werden müssen. Das heißt, wenn beispielsweise der Wasserbehälter aufgefüllt wird, soll die Fehlermeldung aus der Datei gelöscht werden. Dazu wird der Status erneut abgefragt. Bei der Anzahl der Gießzyklen pro Tag wird die Meldung ebenfalls bei Anbruch eines neuen Tages entfernt. Abschließend wird das File über *SD.open()* geöffnet und mit dem Befehl *O_WRITE* überschrieben. *serializeJsonPretty(doc, file)* schreibt die JSON-Daten in die TXT-Datei und formatiert sie für eine bessere Lesbarkeit. Mit dem Befehl *close.file()* wird die Datei geschlossen und damit sichergestellt, dass alle Daten korrekt geschrieben wurden.

3.4 Funktionstest

Mit dem Durchführen mehrerer Funktionstests werden die in den letzten Kapiteln beschriebenen Abläufe mittels Debuggings optimiert und entsprechend der Darstellung finalisiert. Dabei handelt es sich um einen stetigen Prozess des Testens und Überarbeiten des Codes.

Um die Funktionsfähigkeit auch in Betracht auf die Fehlermeldungen zu testen, wird für den finalen Testlauf im Programm die Zykluszeit auf 5 Minuten begrenzt. Da die Sensorkalibrierung des Füllstandsensors nicht definiert werden konnte, wird diese Fehlermeldung in der finalen Testphase nicht betrachtet. Für die Testumgebung wird eine trockene Bodenfeuchte simuliert, indem der Sensor in trockener Erde platziert wird, und der Schlauch zum Gießen nicht in den Topf geführt. Dadurch soll die Fehlermeldung nach dreifachem Gießen ausgelöst werden.

```
{
  "plants": [
    {
      "id": 1,
      "name": "Beaucarnea",
      "measurements": [
        {
          "timestamp": "2024-12-29T17:6:52Z",
          "humidity": 33
        },
        {
          "timestamp": "2024-12-29T17:26:7Z",
          "humidity": 36
        },
        {
          "timestamp": "2024-12-29T17:27:7Z",
          "humidity": 36
        },
        {
          "timestamp": "2024-12-29T17:28:7Z",
          "humidity": 36
        }
      ],
      "watering": [
        {
          "entry": "2024-12-29T17:6:52Z"
        },
        {
          "entry": "2024-12-29T17:26:7Z"
        },
        {
          "entry": "2024-12-29T17:27:7Z"
        }
      ],
      "icon": "beaucarnea"
    }
  ],
  "notifications": [
    {
      "code": "soil_too_wet",
      "status": false,
      "error_message": "One of your plants is drowning :("
    }
  ]
}
```

Listing 6: JSON-Datei des Funktionstests

Quelle: Eigene Darstellung

Nach der durchgeführten Messung wird die JSON-Datei auf der MicroSD-Karte am Computer ausgelesen. Der angezeigte Inhalt der Datei wird in Listing 6 abgebildet. Im Skript ist abzulesen, dass zum einen die Struktur entsprechend der Beispiel-Datei aus Anhang 7 eingehalten wurde. Die Messwerte schwanken leicht, sind jedoch konstant und somit verwertbar. Ebenso ist auffällig, dass das Gießen nach drei erfolgten Gießzyklen ausbleibt und die Fehlermeldung im Skript eingefügt wird. Dementsprechend gilt der Funktionstest als erfolgreich durchgeführt.

4 Datenaustausch zwischen App und Arduino

Die geplante Schnittstelle soll den Datenaustausch zwischen dem Arduino und der Android-App ermöglichen. Die verwendete Datei zur Übermittlung enthält die Sensordaten, Statusmeldungen und andere relevante Informationen wie Name und Icon der Pflanzen. Während der Arduino kontinuierlich Messdaten und Fehlerprotokolle übermittelt, kann die App auf diese Datei zugreifen, um Benutzereinstellungen wie das Pflanzen-Icon oder die Namen zu ändern. Zudem wird die Möglichkeit zur manuellen Bewässerung über die App vorgesehen.

4.1 Technische Optionen für die Umsetzung

Grundsätzlich gibt es mehrere Möglichkeiten den Datenaustausch zwischen Arduino und App zu realisieren. Die unterschiedlichen Kommunikationsschnittstellen werden aufgelistet und hinsichtlich des Anwendungsfalles bewertet.

Tabelle 6: Übersicht der Optionen für den Datenaustausch zwischen dem Arduino und der App

Quelle: Eigene Darstellung

Option	Kosten	Reichweite	Implementierungsaufwand
Externer Server	Hoch	Weltweit (Internet erforderlich)	Hoch (erfordert Servereinrichtung, laufende Kosten)
Arduino als Server (WLAN)	Mittel	Lokal (nur im WLAN)	Mittel (Konfiguration des Arduino als Server)
Gleiches WLAN (Arduino und Handy)	Niedrig	Lokal (nur im WLAN)	Niedrig (einfache Integration in die App)
Bluetooth	Niedrig	Kurz (typischerweise bis 100m)	Mittel (Bluetooth-Verbindung aufbauen und stabil halten)
Direkter USB-Anschluss	Niedrig	Sehr kurz (physische Verbindung)	Niedrig (einfache Verbindung ohne zusätzliche Hardware)

Option	Kosten	Reichweite	Implementierungsaufwand
SD-Karte	Niedrig	Keine (keine Netzwerke erforderlich)	Mittel (manuelle Datenübertragung notwendig)
Cloud-Dienste	Hoch	Weltweit (Internet erforderlich)	Mittel (API-Integration und Cloud-Setup notwendig)

In Tabelle 6 werden relevante Kommunikationsschnittstellen aufgelistet und hinsichtlich Kosten, Reichweite und Implementierungsaufwand ausgewertet.

Die Verwendung eines externen Servers ermöglicht eine weltweite Verfügbarkeit der Daten und eine einfache Integration in die App. Dies erfordert jedoch eine stabile Internetverbindung und kann zusätzliche Kosten durch Serverdienste verursachen.

Bei dem Einsatz eines Arduinos als Server wird dieser über ein eigenes WLAN-Netzwerk angesteuert. Diese Lösung ermöglicht eine direkte Verbindung zwischen der App und dem Arduino, ohne dass ein zusätzlicher externer Server benötigt wird. Die Kommunikation ist jedoch auf das Arduino-WLAN beschränkt, was den Wechsel zwischen Netzwerken erschwert und die Benutzerfreundlichkeit einschränkt.

Eine Alternative ist es sowohl den Arduino als auch das mobile Gerät im selben WLAN zu betreiben. Diese Lösung ermöglicht eine einfache lokale Kommunikation, die ohne Internetzugang funktioniert und keine externen Server oder zusätzliche Hardware erfordert. Ein Nachteil dieser Variante ist, dass beide Geräte stets im gleichen Netzwerk sein müssen, was bei einer instabilen WLAN-Verbindung problematisch ist.

Bluetooth ermöglicht die Kommunikation ohne WLAN und ist energieeffizient. Die Reichweite von Bluetooth ist jedoch begrenzt und die Datenübertragungsrate ist im Vergleich zu WLAN langsamer, weshalb diese Option für größere Datenmengen ungeeignet ist.

Ein direkter Anschluss über USB bietet eine zuverlässige und stabile Verbindung. Allerdings ist dieser Ansatz für den Dauerbetrieb oder für mobile Anwendungen unpraktisch, da dauerhaft eine physische Verbindung erforderlich ist.

Die Speicherung von Daten auf einer SD-Karte ermöglicht eine unabhängige Lösung, die keine Netzwerke oder Server erfordert. Der Nachteil dieser Methode ist, dass die Daten manuell übertragen werden müssen, weshalb diese Lösung für den Nutzer einen höheren Aufwand erfordert.

Cloud-Dienste ermöglichen eine weltweite Verfügbarkeit der Daten und eine einfache Integration in die App. Jedoch besteht eine Abhängigkeit von externen Anbietern und potenziellen Kosten, was diese Lösung für private Anwendungen weniger attraktiv macht.

Die Analyse der verschiedenen Optionen zeigt, dass die Option, bei der sowohl der Arduino als auch das mobile Gerät im gleichen WLAN eingebunden sind, sich als die praktikabelste und kostengünstigste Variante herausstellt. Diese Lösung ermöglicht eine einfache lokale Kommunikation ohne Internetzugang und vermeidet zusätzliche Kosten durch externe Server oder Hardware. Besonders im Kontext eines privaten Bewässerungssystems, bei dem die Geräte in einem stabilen WLAN betrieben werden, bietet diese Lösung eine gute Balance zwischen Benutzerfreundlichkeit, Zuverlässigkeit und Kosteneffizienz. Daher wird diese Methode für die Umsetzung empfohlen.

4.2 Geplante Umsetzung der Kommunikation mit REST-API

Der Arduino wird so konfiguriert, dass er sich mit einem bestehenden WLAN-Netzwerk verbindet. Nach der Verbindung agiert der Arduino als Webserver, welcher mittels Kommunikation über Hypertext Transfer Protocol (HTTP)-Anfragen Daten sendet und empfängt. Die REST-API auf dem Arduino wird so implementiert, dass sie HTTP-Methoden wie GET und POST unterstützt, um das Abrufen und Schreiben der JSON-Datei zu ermöglichen. In der App wird ein HTTP-Client integriert, der die REST-API des Arduinos anruft. Beispielsweise kann die App mit der GET /data-Anfrage die aktuellen Sensordaten abrufen und mit der POST /data-Anfrage neue Daten an den Arduino senden. Diese Anfragen ermöglichen es der App, mit dem Arduino zu kommunizieren und dadurch das Bewässerungssystem in Echtzeit zu überwachen und zu steuern.

5 Fazit

Nach abschließender Betrachtung lässt sich sagen, dass diese Ausarbeitung eine funktionsfähige Grundlage zur Realisierung eines Pflanzenbewässerungssystems darstellt. Mit Hilfe der Programmierung eines Arduino und der Einbindung von Sensoren und Aktoren, wie einer Pumpe, konnte dies umgesetzt werden. Bei der Erarbeitung ergaben sich Probleme wie die Stromversorgung der unterschiedlichen Komponenten, die Kalibrierung des Füllstandsensors sowie das Schreiben in eine JSON-Datei. Die beiden letzteren konnten innerhalb des Umfangs der Arbeit nicht vollständig gelöst werden. Trotz dessen bietet das System einen vollständig funktionierenden Bewässerungsablauf, welcher auch zukünftig mittels aufgeführter Schnittstelle in ein App-Interface eingebunden werden kann. Darüber hinaus bieten sich ebenfalls Optimierungsmöglichkeiten hinsichtlich des Designs, wie beispielsweise ein an die Komponenten angepasster Deckel. Durch die Verwendung des modularen Aufbaus des Programmcodes ist es zusätzlich ohne großen Aufwand möglich, weitere Sensorik, wie etwa ein Temperaturfühler oder ein pH-Wert-Messgerät, aber auch weitere Pflanzen in das System einzubinden.




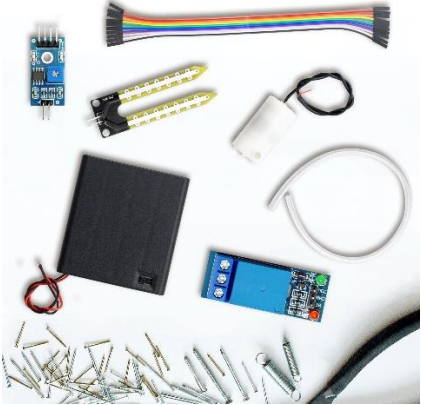
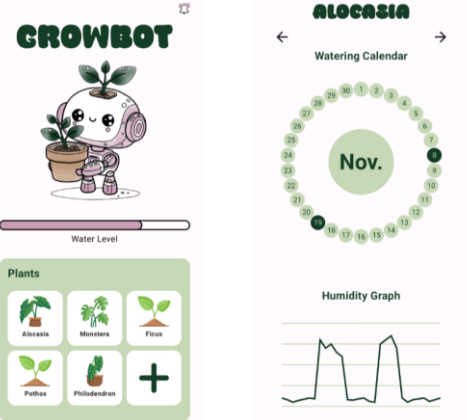
Anhangsverzeichnis

Anhang 1: Übersicht Pflanzenbewässerungssysteme	X
Anhang 2: Python-Skript zur Mittelwertbildung und grafischen Darstellung der Grenzwerte	XI
Anhang 3: Messwerte zur Grenzwertfassung des Bodenfeuchtesensors (A0) - Zustand nass	XII
Anhang 4: Messwerte zur Grenzwertfassung des Füllstandsensors (A1) - Zustand leer	XII
Anhang 5: Messwerte zur Grenzwertfassung des Füllstandsensors (A1) - Zustand voll	XIII
Anhang 6: Klasse Plant mit Variablendeklaration, Konstruktor und zugehörigen Funktionen	XIII
Anhang 7: Beispiel für JSON-Datei	XV
Anhang 8: Funktion loadFromJson zum Auslesen der JSON-Datei	XVI
Anhang 9: Funktion saveToJson zum Schreiben und Speichern der JSON-Datei..	XVII

Anhang

Anhang 1: Übersicht Pflanzenbewässerungssysteme

Quellen: Vgl. [21], [22], [23], [24]

Produkt:	Blumat für Zimmerpflanzen - Classic	Pflanzkübel Blumentopf Ø 32 cm	1 Beam + Wi-Fi Hub	Pflanzen-Bewässerung-Set mit Bodenfeuchte-erkennung Sensor	Growbot
					
Hersteller:	Blumat	Lazy Leaf	FYTA	AZ-Delivery	-
Bewässerungsart:	Bewässerungskegel (mit Tropfschlauch)	Selbstgießender Blumentopf; 10 Gießstufen je nach Pflanzenart	KI-gestützter, smarter Pflanzensensor inklusive App-Verknüpfung	Automatisches Bewässerungsmodul DIY-Kit automatische Wasserpumpe	Steuerung der Sensoren und Aktoren über Mikrocontroller-Anbindung zu App-Interface
Energieverbrauch:	keiner	Akkulaufzeit etwa 8 Wochen	Hybride Stromversorgung, mit einer Batterielaufzeit bis zu 12 Monate	Externe Stromversorgung durch Netzteil	Externe Stromversorgung durch Netzteil
Aufwand/ Installation:	Niedrig / Niedrig	Niedrig / Niedrig	Niedrig / Mittel	Niedrig / Hoch	Niedrig / Mittel
Preis:	Ab € 3,99	Ab € 24,90	Ab € 69,99	Ab €27,99	-

Anhang 2: Python-Skript zur Mittelwertbildung und grafischen Darstellung der Grenzwerte

Quelle: Eigene Darstellung

```
import matplotlib.pyplot as plt

# Funktion zum Einlesen von Messdaten aus einer Datei
def lese_messdaten(dateiname):
    try:
        with open(dateiname, 'r') as file:
            daten = [float(zeile.strip()) for zeile in file if zeile.strip()]
            return daten
    except FileNotFoundError:
        print(f"Datei '{dateiname}' wurde nicht gefunden.")
        return []
    except ValueError:
        print("Fehler beim Verarbeiten der Datei. Stellen Sie sicher, dass die
Datei nur Zahlen ent
        return []

# Mittelwert berechnen
def berechne_mittelwert(daten):
    if not daten:
        return None
    return sum(daten) / len(daten)

# Funktion zum Plotten der Messdaten
def plot_daten(daten, mittelwert):
    plt.figure(figsize=(10, 6))
    plt.plot(daten, label='Messwerte', marker='o')
    plt.axhline(mittelwert, color='red', linestyle='--', label=f'Mittelwert:
{mittelwert}
    plt.title('Messwerte und Mittelwert')
    plt.xlabel('Index')
    plt.ylabel('Wert')
    plt.legend()
    plt.grid(True)
    plt.show()

# Hauptfunktion
def ausfuehren(dateiname):
    daten = lese_messdaten(dateiname)

    if not daten:
        return

    mittelwert = berechne_mittelwert(daten)

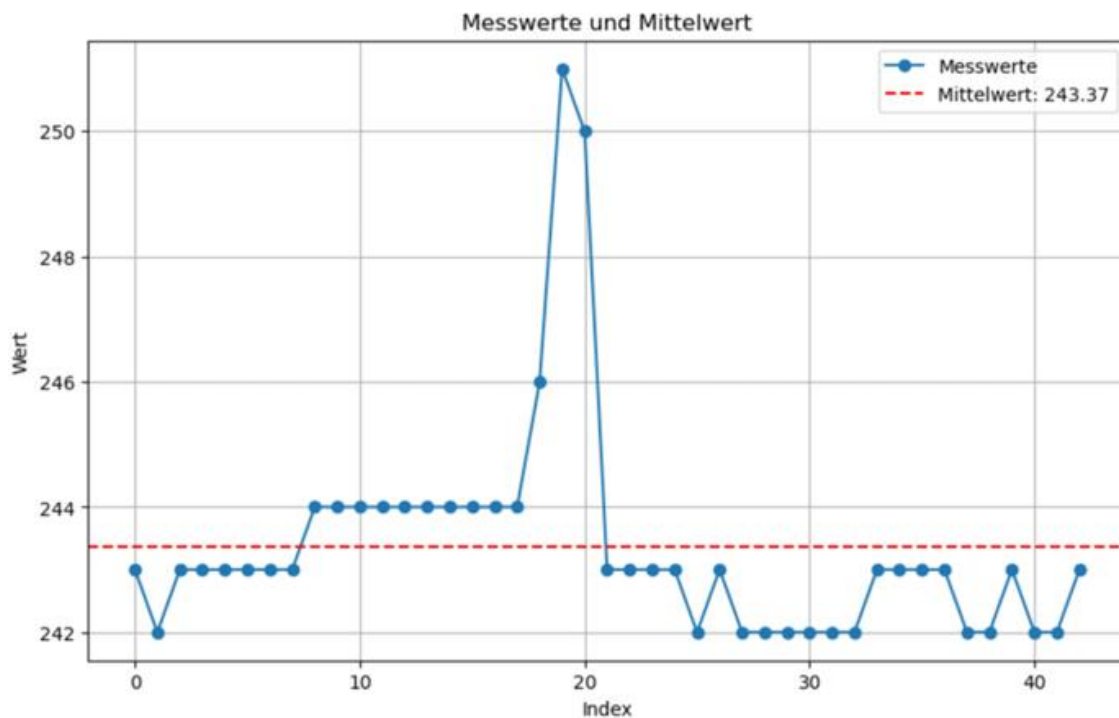
    if mittelwert is not None:
        print(f"Mittelwert der Messreihe: {mittelwert:.2f}")
        plot_daten(daten, mittelwert)
    else:
        print("Es wurden keine gültigen Daten gefunden.")

# Aufruf der Funktion mit den Daten des Bodenfeuchtesensors
ausfuehren('A0_trocken.txt')
ausfuehren('A0_nass.txt')

# Aufruf der Funktion mit den Daten des Füllstandsensors
ausfuehren('A1_leer.txt')
ausfuehren('A1_voll.txt')
```

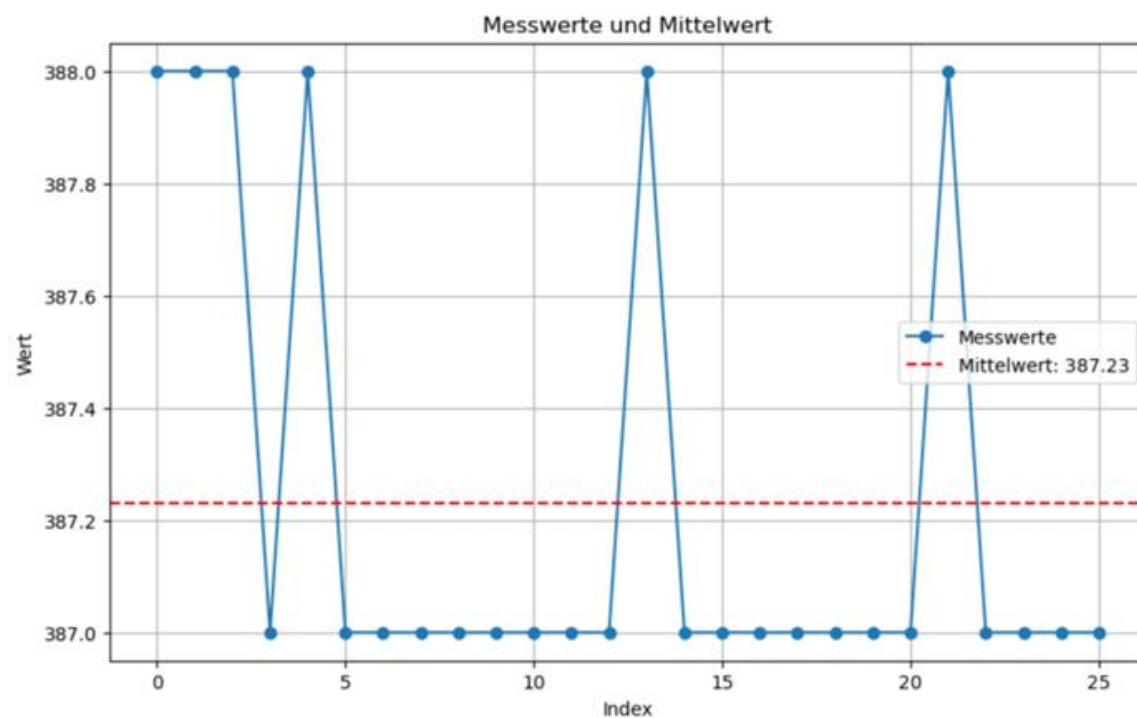
Anhang 3: Messwerte zur Grenzwert erfassung des Bodenfeuchtesensors (A0) - Zustand nass

Quelle: Eigene Darstellung



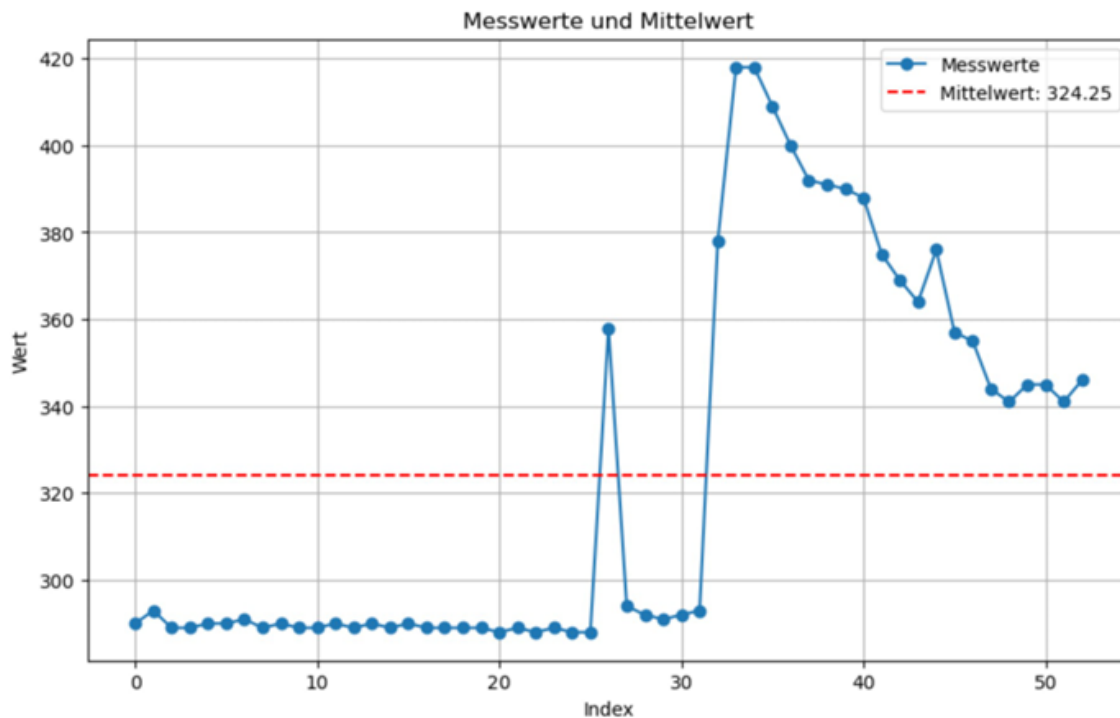
Anhang 4: Messwerte zur Grenzwert erfassung des Füllstandsensors (A1) - Zustand leer

Quelle: Eigene Darstellung



Anhang 5: Messwerte zur Grenzwert erfassung des Füllstandsensors (A1) - Zustand voll

Quelle: Eigene Darstellung



Anhang 6: Klasse Plant mit Variablendeklaration, Konstruktor und zugehörigen Funktionen

Quelle: Eigene Darstellung

```
class Plant {
public:
    int id; // Pflanz-ID
    String name, icon; // Pflanzename und Icon
    Measurements measurements; // Messwerte
    Watering watering; // Gießhistorie
    HumiditySensor humiditySensor; // Feuchtigkeitssensor
    Pump& pump; // Pumpe
    float humidityThreshold; // Schwellenwert für Bewässerung

    // Konstruktor
    Plant(int i, String n, int humidityPin, int humidityMin, int humidityMax,
    Pump& p, String ic, float threshold)
        : id(i), name(n), humiditySensor(humidityPin, humidityMin,
    humidityMax), pump(p), icon(ic), humidityThreshold(threshold) {}

    // Messungen aktualisieren
    void updateMeasurements() {
        DateTime now = rtc.now();
        String timestamp = formatTimestamp(now);
        float humidity = humiditySensor.readValue();
        measurements.addMeasurement(timestamp, humidity);
    }
}
```

```
// Überprüfung, ob Bewässerung benötigt wird
bool needsWatering() {
    float humidity = humiditySensor.readValue();
    return humidity < humidityThreshold;
}

// Überprüfung, ob überwässert wird - Begrenzung auf 3 Gießzyklen
bool canWaterMoreToday() {
    DateTime now = rtc.now();
    String today = String(now.year()) + "-" + String(now.month()) + "-" +
String(now.day());
    int waterCount = 0;

    for (const String& entry : watering.entries) {
        if (entry.startsWith(today)) {
            waterCount++;
        }
    }

    return waterCount < 3; // Maximal 3 Mal pro Tag gießen
}

// Pflanze bewässern
void waterPlant() {
    DateTime now = rtc.now();
    String entry = formatTimestamp(now);
    watering.update(entry);
    pump.start(); // Pumpe starten
    delay(10000); // 10 Sekunden Bewässerung
    pump.stop(); // Pumpe stoppen
}

// Zeitstempel formatieren
// Bei der vollen Stunde können die Minuten und Sekunden genullt werden
String formatTimestamp(const DateTime& now) {
    return String(now.year()) + "-" + String(now.month()) + "-" +
String(now.day()) +
        "T" + String(now.hour()) + ":00:00Z";
}

// Zeitstempel für Testprogramm:
/*String formatTimestamp(const DateTime& now) {
    return String(now.year()) + "-" + String(now.month()) + "-" +
String(now.day()) +
        "T" + String(now.hour()) + ":" + String(now.minute()) + ":" +
String(now.second()) + "Z";
}*/

// Überprüfen, ob die Pflanze in Ordnung ist
bool isOk() {
    return !needsWatering();
}
};
```

Anhang 7: Beispiel für JSON-Datei

Quelle: Eigene Darstellung

```
{
  "plants": [
    {
      "id": 1,
      "name": "Alocasia",
      "measurements": [
        {
          "timestamp": "2024-11-01T12:00:00Z",
          "temperature": 22.5,
          "humidity": 55.0
        },
        {
          "timestamp": "2024-11-02T12:00:00Z",
          "temperature": 20.5,
          "humidity": 59.0
        },
        {
          "timestamp": "2024-11-03T12:00:00Z",
          "temperature": 20.5,
          "humidity": 59.0
        },
        {
          "timestamp": "2024-11-04T12:00:00Z",
          "temperature": 20.5,
          "humidity": 59.0
        },
        {
          "timestamp": "2024-11-06T12:00:00Z",
          "temperature": 20.5,
          "humidity": 59.0
        },
        {
          "timestamp": "2024-11-07T12:00:00Z",
          "temperature": 20.5,
          "humidity": 59.0
        }
      ],
      "watering": [
        {"entry": "2024-11-01T12:00:00Z"},
        {"entry": "2024-11-02T12:00:00Z"},
        {"entry": "2024-11-03T12:00:00Z"},
        {"entry": "2024-11-07T12:00:00Z"}
      ],
      "icon": "alocasia"
    },
    {
      "id": 2,
      "name": "Monstera",
      "measurements": [
        {
          "timestamp": "2024-10-02T12:00:00Z",
          "temperature": 23.0,
          "humidity": 60.0
        }
      ],
      "watering": [
        {"entry": "2024-11-02T12:00:00Z"},
        {"entry": "2024-11-08T12:00:00Z"}
      ]
    }
  ]
}
```

Anhang 8: Funktion loadFromJson zum Auslesen der JSON-Datei

Quelle: Eigene Darstellung

```
// JSON-Daten laden
void loadFromJson() {
    File file = SD.open(jsonFilePath, FILE_READ);
    if (!file) {
        Serial.println("JSON-Datei konnte nicht geöffnet werden.");
        saveToJson();
        return;
    }

    StaticJsonDocument<2048> doc;
    DeserializationError error = deserializeJson(doc, file);
    if (error) {
        Serial.print("Fehler beim Lesen der JSON-Datei: ");
        Serial.println(error.f_str());
        file.close();
        return;
    }

    JsonArray plantsArray = doc["plants"].as<JsonArray>();
    for (JsonObject plantObj : plantsArray) {
        int id = plantObj["id"];
        String name = plantObj["name"];
        JsonArray measurements = plantObj["measurements"];
        JsonArray watering = plantObj["watering"];

        for (Plant& plant : plants) {
            if (plant.id == id) {
                for (JsonObject measurement : measurements) {
                    String timestamp = measurement["timestamp"];
                    float humidity = measurement["humidity"];
                    plant.measurements.addMeasurement(timestamp, humidity);
                }
                for (JsonObject waterEntry : watering) {
                    String entry = waterEntry["entry"];
                    plant.watering.update(entry);
                }
                break;
            }
        }
    }

    file.close();
    Serial.println("JSON geladen.");
}
```

Anhang 9: Funktion saveToJson zum Schreiben und Speichern der JSON-Datei

Quelle: Eigene Darstellung

```
// JSON-Daten speichern
void saveToJson() {
    StaticJsonDocument<2048> doc;
    JsonArray plantsArray = doc.createNestedArray("plants");

    for (Plant& plant : plants) {
        JsonObject plantObj = plantsArray.createNestedObject();
        plantObj["id"] = plant.id;
        plantObj["name"] = plant.name;

        JsonArray measurementsArray =
plantObj.createNestedArray("measurements");
        for (const JsonObject& m : plant.measurements.values) {
            measurementsArray.add(m);
        }

        JsonArray wateringArray = plantObj.createNestedArray("watering");
        for (const String& entry : plant.watering.entries) {
            JsonObject wateringEntry = wateringArray.createNestedObject();
            wateringEntry["entry"] = entry;
        }
        plantObj["icon"] = plant.icon;
    }

    JsonArray notifications = doc.createNestedArray("notifications");

    if (levelSensor->readValue() <= 10) {
        JsonObject lowWaterNotification = notifications.createNestedObject();
        lowWaterNotification["code"] = "water_level_too_low";
        lowWaterNotification["status"] = true;
        lowWaterNotification["error_message"] = "Please refill your water!";
    }

    for (Plant& plant : plants) {
        if (!plant.isOk()) {
            if (!plant.canWaterMoreToday() && plant.needsWatering()) {
                JsonObject wetSoilNotification =
notifications.createNestedObject();
                wetSoilNotification["code"] = "soil_too_wet";
                wetSoilNotification["status"] = false;
                wetSoilNotification["error_message"] = "One of your plants is
drowning :(";
            }
        }
    }

    JsonArray updatedNotifications = doc["notifications"].as<JsonArray>();

    for (int i = updatedNotifications.size() - 1; i >= 0; i--) {
        JsonObject notification = updatedNotifications[i];

        if (notification["code"] == "water_level_too_low" && levelSensor-
>readValue() > 10) {
            updatedNotifications.remove(i);
        }
        if (notification["code"] == "soil_too_wet") {
            bool plantOk = true;
            for (Plant& plant : plants) {
```

Literaturverzeichnis

- [1] Statista Research Department, „Number of users of smart homes worldwide from 2019 to 2028“, statistica. Zugriffen: 23. Oktober 2024. [Online]. Verfügbar unter: <https://www.statista.com/forecasts/887613/number-of-smart-homes-in-the-smart-home-market-in-the-world>
- [2] Prof. Dr. O. Bendel, „Smart Home“, Gabler Wirtschaftslexikon. Zugriffen: 23. Oktober 2024. [Online]. Verfügbar unter: <https://wirtschaftslexikon.gabler.de/definition/smart-home-54137>
- [3] F. Suhr, „Zimmerpflanzen sind beliebt“, statistica. Zugriffen: 23. Oktober 2024. [Online]. Verfügbar unter: <https://de.statista.com/infografik/21230/umfrage-zu-pflanzen-zuhause-und-im-garten/>
- [4] Statista Research Department, „Würden Sie von sich behaupten, einen grünen Daumen oder ein Talent für Pflanzenzucht zu haben?“, statistica. Zugriffen: 24. Oktober 2023. [Online]. Verfügbar unter: <https://de.statista.com/statistik/daten/studie/1326821/umfrage/umfrage-zur-befaehtigung-fuer-die-pflanzenzucht/#:~:text=In%20einer%20aktuellen%20Umfrage%20gaben,f%C3%BCr%20die%20Pflanzenzucht%20zu%20haben.>
- [5] R. Finger, S. M. Swinton, N. El Benni, und A. Walter, „Precision Farming at the Nexus of Agricultural Production and the Enviroment“, Review Article Volume 11, Apr. 2019. [Online]. Verfügbar unter: <https://www.annualreviews.org/content/journals/10.1146/annurev-resource-100518-093929#right-ref-B143>
- [6] Prof. Dr. H. W. Griepentrog, „7. Agrarwissenschaftliches Symposium - Digitale Landwirtschaft: Big Data – Smart Data - Datenmanagement“, Hans Eisenmann-Zentrum, Zentralinstitut für Agrarwissenschaften der Technischen Universität München, Zukünftige Entwicklungen im Precision Farming, Sep. 2016.
- [7] Arduino Docs, „Arduino® UNO R4 WiFi Datasheet“. 20. Dezember 2024. Zugriffen: 22. Dezember 2024. [Online]. Verfügbar unter: <https://docs.arduino.cc/resources/datasheets/ABX00087-datasheet.pdf>

-
- [8] Espressif Systems, „ESP32 Series Datasheet Version 4.7“. September 2024. Zugriffen: 22. Dezember 2024. [Online]. Verfügbar unter: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- [9] Arduino Docs, „Arduino® Nano 33 BLE Sense Datasheet“. 20. Dezember 2024. Zugriffen: 22. Dezember 2024. [Online]. Verfügbar unter: <https://docs.arduino.cc/resources/datasheets/ABX00031-datasheet.pdf>
- [10] Raspberry Pi Ltd, „Raspberry Pi Pico W Datasheet“. 15. Oktober 2024. Zugriffen: 22. Dezember 2024. [Online]. Verfügbar unter: <https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf>
- [11] o.V., „Automatische Bewässerung Arduino“, Arduino-Garten.de. Zugriffen: 5. Dezember 2024. [Online]. Verfügbar unter: <https://arduino-garten.de/>
- [12] „Mögliche Anwendungen der DVS Bodenfeuchte-Sensoren“, DVS BERECHNUNG. Zugriffen: 6. Dezember 2024. [Online]. Verfügbar unter: <https://dvs-berechnung.de/sensoren-ueberwachung-steuerung-bewaesserung>
- [13] Prof. Dr.-Ing. J. Korthals, „Sensorik“. Vorlesungsunterlagen, WS 2024.
- [14] P. Müller, „DIY Automatisches Bewässerungssystem“, cryCode.de. Zugriffen: 5. Dezember 2024. [Online]. Verfügbar unter: <https://crycode.de/diy-automatisches-bewaesserungssystem/>
- [15] pollux lux, „Automatische Pflanzenbewässerung“. Zugriffen: 5. Dezember 2024. [Online]. Verfügbar unter: <https://polluxlabs.net/arduino-projekte/automatische-pflanzenbewaesserung/>
- [16] „Füllstandmesstechnik: Grundlagen“, Automation24. Zugriffen: 7. Dezember 2024. [Online]. Verfügbar unter: <https://www.automation24.de/fuellstandmesstechnik-grundlagen>
- [17] Maxim Integrated Products, „DS1307 64 x 8, Serial, I 2 C Real-Time Clock Datasheet“. 2015. Zugriffen: 22. Dezember 2024. [Online]. Verfügbar unter: <https://www.analog.com/media/en/technical-documentation/datasheets/DS1307.pdf>
- [18] „WOKWI“. Zugriffen: 5. Dezember 2014. [Online]. Verfügbar unter: <https://www.wokwi.com>
-

-
- [19] M. Stöckel, „Entwicklung eines Pflanzenbewässerungssystems mit einem Arduino Mikrocontroller-Board“. 23. Dezember 2023.
- [20] Astra WordPress Theme, „PlantText“. Zugriffen: 22. Dezember 2024. [Online]. Verfügbar unter: <https://www.planttext.com>
- [21] bambach GbR, „Blumat für Zimmerpflanzen - Classic“, Blumat. Zugriffen: 2. Dezember 2024. [Online]. Verfügbar unter: <https://www.blumat.de/blumat-classic/>
- [22] Lazy Leaf, „Pflanzkübel Blumentopf Ø 32 cm - Wassertankgröße 6,35 l, Selbstgießend“, OTTO. Zugriffen: 1. Dezember 2024. [Online]. Verfügbar unter: <https://www.otto.de/p/lazy-leaf-pflanzkuebel-blumentopf-o-32-cm-wassertankgroesse-6-35-l-selbstgiessend-S0U2M0XW/#ech=28874335&variationId=S0U2M0XW0UD7>
- [23] FYTA, „1 Beam + Wi-Fi Hub“, FYTA. Zugriffen: 1. Dezember 2024. [Online]. Verfügbar unter: <https://fyta.de/products/test-single-hub>
- [24] AZ-Delivery, „Pflanzen-Bewässerung-Set mit Bodenfeuchteerkennung Sensor“, AZ-Delivery. Zugriffen: 1. Dezember 2024. [Online]. Verfügbar unter: <https://www.az-delivery.de/products/pflanzen-bewasserung-set>