Hochschule Aalen
Institut für Informatik
Wintersemester 2022/2023

Mobile and Embedded Software Development - Projekt

„Smart Greenhouse“

Christian Schmidt – 79557@studmail.hs-aalen.de
Robin Röcker - 80193@studmail.hs-aalen.de
Maximilian Zeger - 82800@studmail.hs-aalen.de
Yannick Söll – 80060@studmail.hs-aalen.de

27. Januar 2023

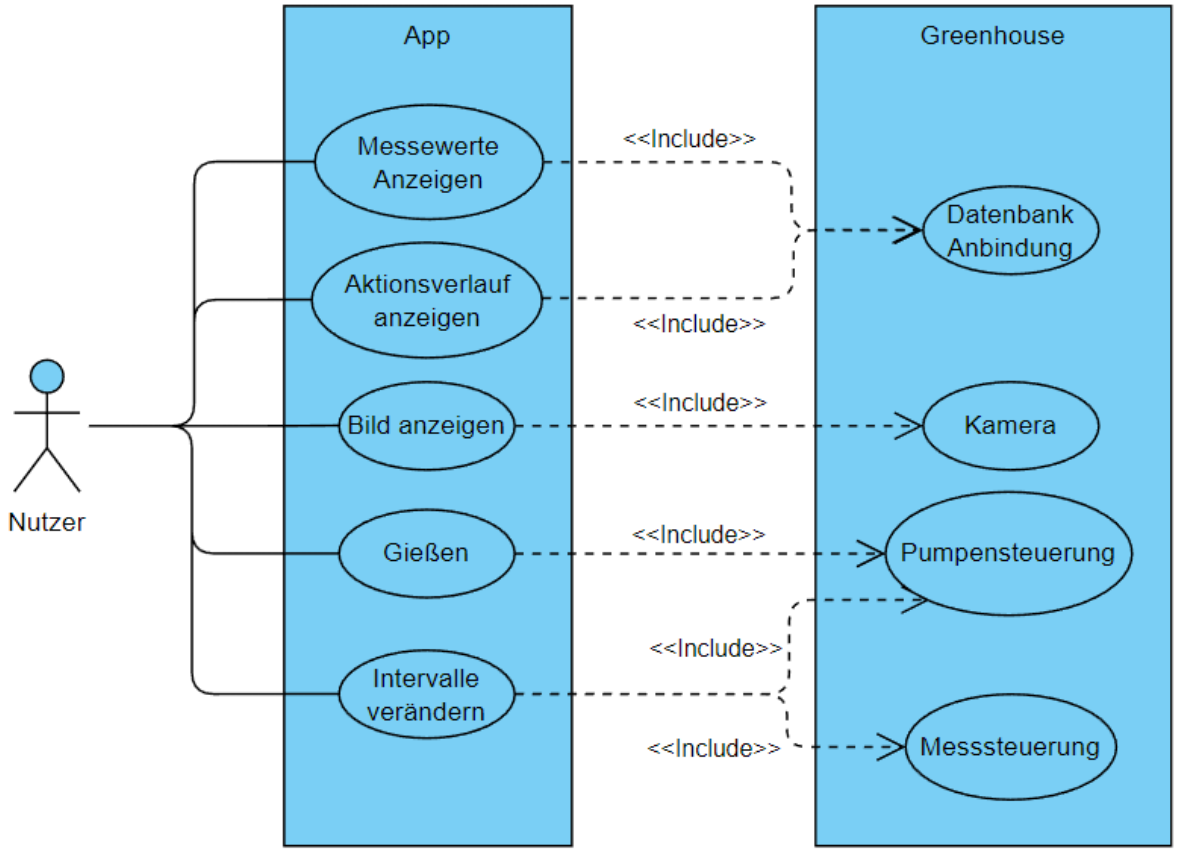
Inhalt

1. Übersicht und Ziel

Das Ziel unseres Projekts ist die Entwicklung und der Aufbau eines durch Sensoren überwachtes und steuerbares Modell-Gewächshaus. Das Projekt wurde mithilfe eines Trello-board geplant und durchgeführt. Die Zusammenarbeit wurde über ein GIT-Repository auf Bitbucket ermöglicht.

- Das Gewächshaus soll in der Lage sein die Temperatur sowie die Boden- und Luftfeuchtigkeit zu messen
- Außerdem verfügt das System über ein steuerbares Gießsystem
- Die Sensorwerte werden in festlegbaren Intervallen in einer Datenbank gespeichert
- Zusätzlich werden die Aktionen des Gießsystems gespeichert
- Durch eine App wird eine Abfrage der gespeicherten Daten, die Steuerung des Gießsystems sowie Änderungen an den verwendeten Intervallen, für Datenspeicherung und Gießen, ermöglicht
- Das Gewächshaus soll eine Kamera verbaut haben, von welcher man über die App, Bilder anzeigen lassen kann

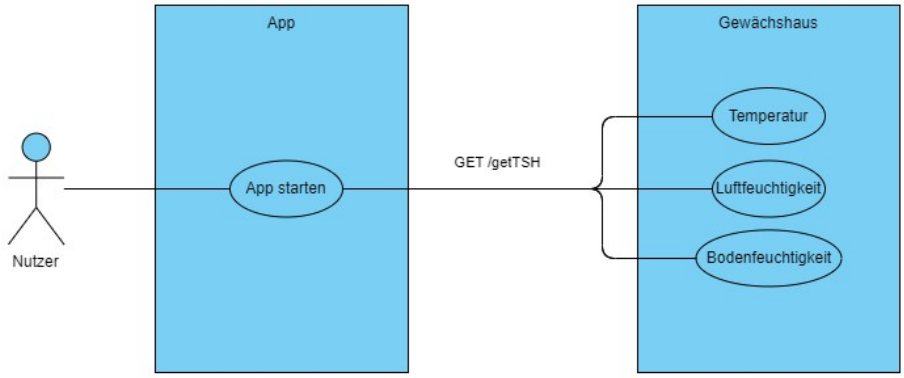
--Zeger--



2. Funktionalitäten und Szenarien

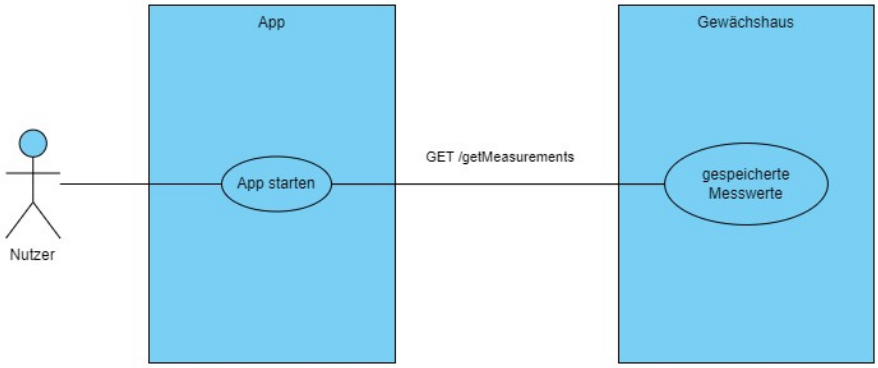
Nutzer möchte aktuelle Messwerte sehen:

Der Nutzer startet die App welche dann selbständig die aktuellsten Daten vom Gewächshaus anfragt und diese mithilfe von drei Radialanzeigen darstellt.



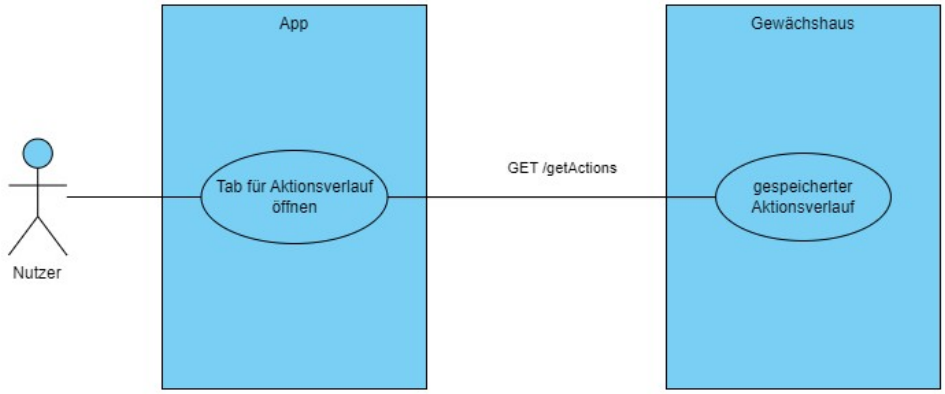
Nutzer möchte alle gespeicherten Messwerte sehen:

Der Nutzer startet die App welche dann selbständig die alle Daten vom Gewächshaus anfragt und im unteren Teil der App auflistet.



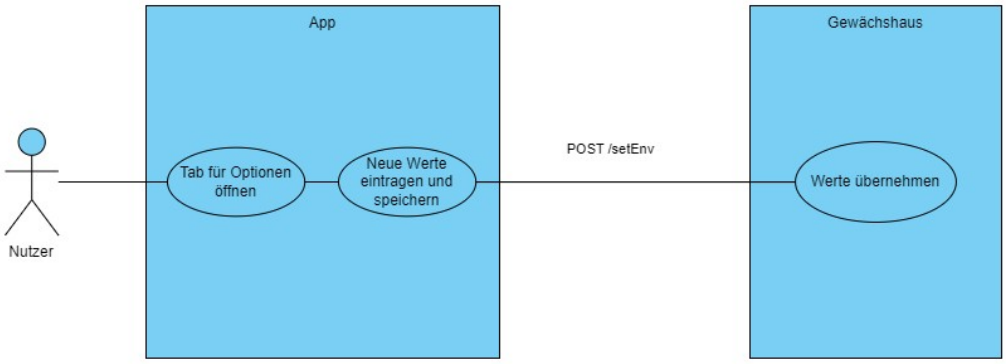
Nutzer möchte den Verlauf der ausgeführten Aktionen sehen:

Der Nutzer navigiert in der App in den zweiten Tab welcher dazu dient den Aktionsverlauf anzuzeigen. Die App fragt dann beim Gewächshaus den Gesamtverlauf aller ausgeführten Aktionen an und stellt diese in einer Liste dar.



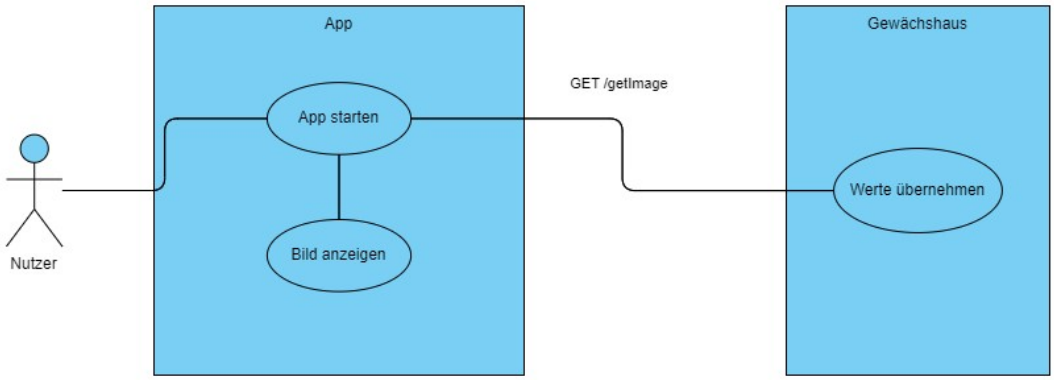
Nutzer möchte das Intervall für Messungen und Gießdauer verändern:

Um das Intervall in welchen die Messwerte gespeichert werden und die Gießdauer zu verändern geht der Nutzer in den dritten Tab und gibt dort die gewünschten Werte in die entsprechenden Felder ein. Nachdem der Nutzer diese bestätigt übermittelt die App diese an das Gewächshaus welches die Werte übernimmt.



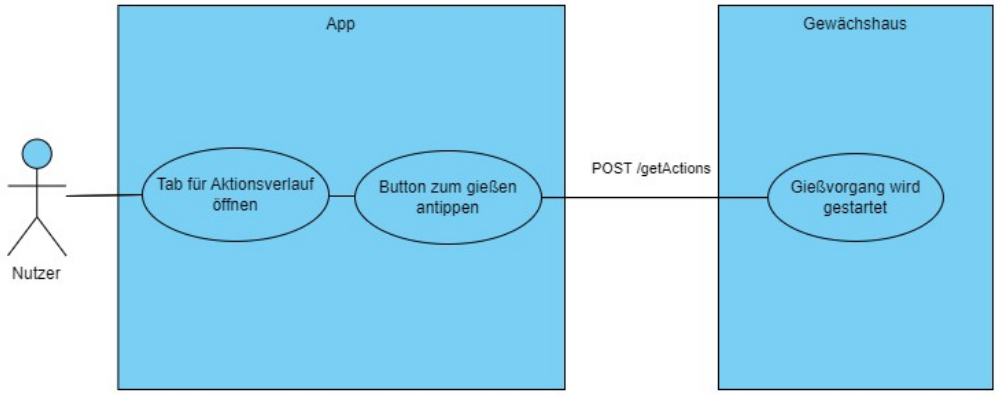
Nutzer möchte ein Bild des Gewächshauses sehen:

Beim Starten der App wird automatisch ein Bild vom Gewächshaus angefragt. Dieses Bild kann der Nutzer öffnen indem er auf das Icon tippt.



Nutzer möchte manuell gießen:

Der Nutzer navigiert in der App in den zweiten Tab welcher am oberen Bildschirmrand einen Knopf hat welcher die manuelle Gießfunktion auslöst. Die App schickt den Gießbefehl dann an das Gewächshaus.



System speichert automatisch Messwerte:

Das System speichert automatisch in dem gesetzten Intervall die Messwerte ab.

System bewässert automatisch:

Sobald der Gewächshausboden zu trocken ist, bewässert das Gewächshaus automatisch für die gesetzte Dauer.

--Zeger--

3. Architektur

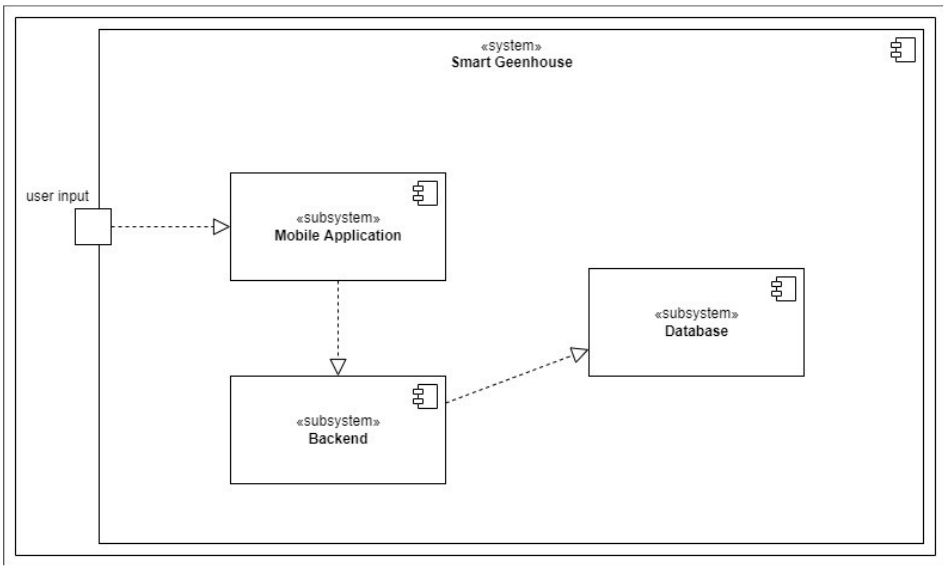
Im folgenden Abschnitt wird die Architektur der Anwendung genauer beschrieben. Dabei werden Aspekte wie die Bausteinsicht und die Softwarearchitektur des Systems genauer beleuchtet und erklärt.

--Schmidt--

3.1 Bausteinsicht - Backend

Die Smart Greenhouse Applikation ist grundlegend in drei Systeme unterteilt. Die Mobile App bildet dabei die Schnittstelle, über die der Benutzer mit dem System interagiert. Es nimmt Input entgegen und leitet diese an das Backend weiter. Des Weiteren stellt es alle wichtigen Informationen für den Benutzer bereit. Somit bildet die mobile Applikation einen essenziellen Baustein der Architektur. Im Backend werden die Anfragen der mobilen App bearbeitet und sendet gegebenenfalls die nötigen Signale an die Aktoren. Eine weitere Aufgabe des Backend ist die automatisiere Regelung des Gewächshauses. Es steuert die essenziellen Aufgaben, wie das Bewässern sowie das Auslesen aller Kontrollsensoren. Außerdem übernimmt das Backend sämtliche Lese- und Schreiboperationen auf der Datenbank. Somit ist das Backend die Kernkomponente der Applikation. Mit ihm steht des Weiteren die Datenbank in Verbindung, welche direkt auf dem Raspberry Pi gehostet wird. Sie sorgt für eine persistente Datenhaltung und garantiert durch den lokalen Host eine hohe Verfügbarkeit. Mit dieser Anordnung können alle Anforderungen an das System umgesetzt werden.

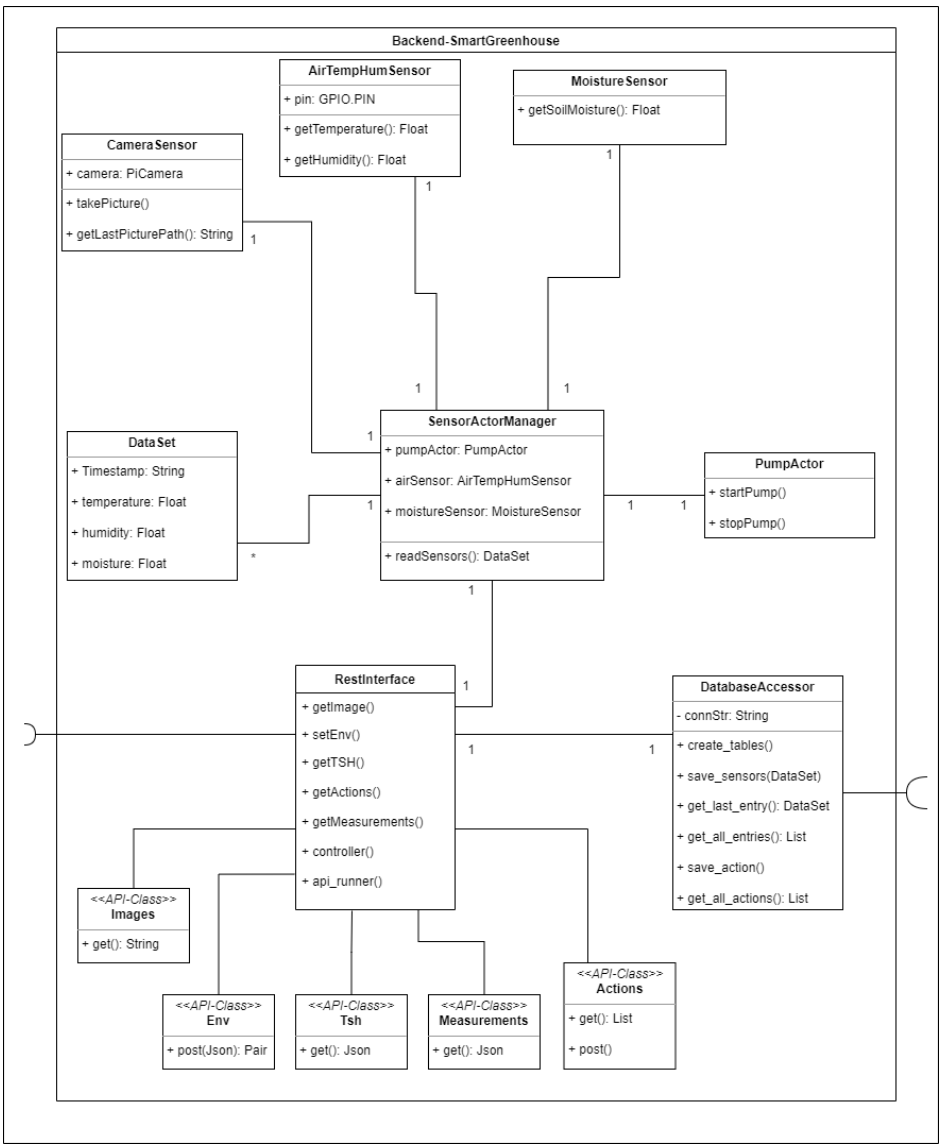
--Schmidt--



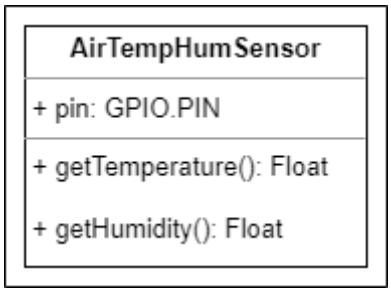
3.2 UML-Diagramm – Backend

Das folgende Diagramm zeigt den Aufbau des Backend und visualisiert die einzelnen Komponenten und ihre Abhängigkeiten. Grundsätzlich ist das Backend in vier verschiedene Klassentypen unterteilt. Es gibt Klassen wie den HumiditySensor oder PumpActor, die ausschließlich die Aufgabe haben, die eingebauten Sensoren bzw. Aktoren zu lesen und anzusteuern. Außerdem gibt es die Datenklasse DataSet, welche eine reine Datenklasse darstellt und keine Methoden oder Businesslogik implementiert. Des Weiteren gibt es die Schnittstellen, wie das RestInterface oder der DatabaseAccessor. Diese kommunizieren mit anderen Systemen, wie etwa einer mobilen Anwendung oder einer Datenbank. Zuletzt gibt es noch die Rest-API-Klassen, die sogenannte Helper-Klassen sind, die vom Rest-Interface verwendet werden, um Get- und Post-Requests verarbeiten zu können.

--Schmidt--

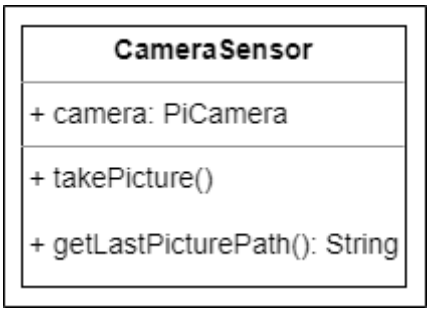


3.2 Klassenbeschreibungen – Backend



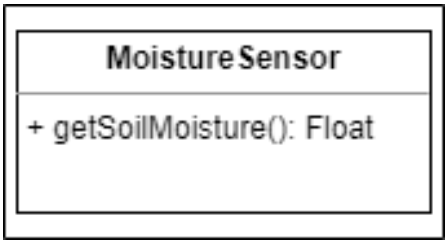
Die AirTempHumSensor-Klasse stellt alle Funktionalitäten bereit, die benötigt werden, um die Lufttemperatur und die Luftfeuchtigkeit im Gewächshaus zu bestimmen. Die Klasse besitzt eine pin-Variable, mit der angegeben wird über welchen GPIO-Pin des Raspberry Pi die Daten des Sensors empfangen werden.

--Schmidt--



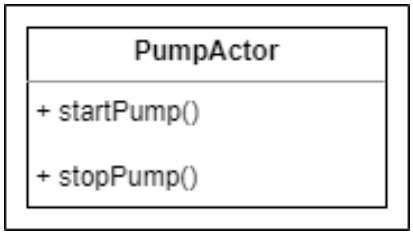
Die CameraSensor-Klasse stellt die Funktionalität bereit, die benötigt wird, um Aufnahmen vom Gewächshaus zu machen und diese zu speichern. Die Klasse besitzt eine camera-Variable, die Funktionen bereitstellt, um Bilder mit dem Raspberry Pi aufzunehmen.

--Schmidt--



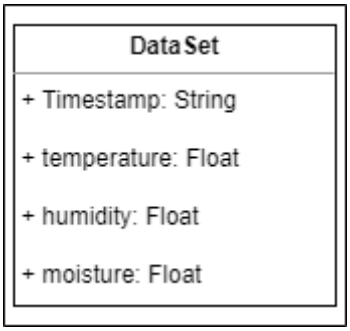
Die MoistureSensor-Klasse stellt alle Funktionalitäten bereit, die benötigt werden, um die Bodenfeuchtigkeit im Gewächshaus zu bestimmen. Die Klasse besitzt die Funktion, die analoge Daten des Sensors in digitale Daten zu interpretieren und umzuwandeln.

--Schmidt--



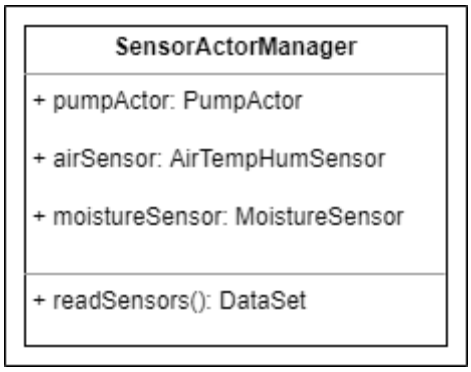
Die PumpActor-Klasse stellt die Funktionalität zur Verfügung, die benötigt wird, um die angeschlossene Pumpe An- und Auszuschalten. Dabei wird über einen GPIO-Pin des Raspberry Pi ein Signal gesendet, dass einen zweiten Schaltkreis öffnet und so den Stromkreis der Pumpe öffnet.

--Schmidt--



Die DataSet-Klasse ist eine reine Datenklasse, die verwendet wird um alle Messdaten innerhalb des Systems einheitlich überreichen zu können. Außerdem werden somit die Persistenz bezogenen Operationen vereinfacht.

--Schmidt--

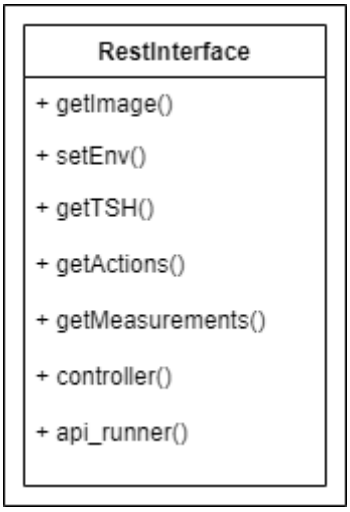


Die SensorActorManager-Klasse stellt die Funktion bereit, alle Daten der Sensoren auszulesen. Da die Sensoren mit Callback-Funktionen arbeiten, die aufgerufen werden, wenn eine Messung nicht erfolgreich war, müssen diese behandelt werden. Diese Aufgabe übernimmt die readSensors-Methode, indem sie validiert, ob jeder Sensor einen gültigen Wert geliefert hat.

--Schmidt--

Die DatabaseAccessor-Klasse stellt die Funktionalität zur Verfügung, die benötigt wird, um die Datenbankoperationen ausführen zu können. Mit ihr können neue Tabellen angelegt werden und sowohl die Aktionen der Wasserpumpe als auch die Daten der Sensoren dokumentiert und persistiert werden. Die connStr-Variable enthält dabei den String der für die Verbindung zur PostgreSQL-Datenbank benötigt wird.

--Schmidt--



Die RestInterface-Klasse ist die Hauptkomponente der Anwendung. Sie stellt alle Endpunkte der zur Verfügung, über die die mobile Anwendung alle Daten abfragen und alle Aktionen ausführen kann. Für das Interface wird dabei das Flask-Modul von Python verwendet. Als Wrapper wird zusätzlich das restx-Modul von Python verwendet, dass das Erstellen von API-Klassen ermöglicht und eine automatische Swagger-Dokumentation bereitstellt. Die einzelnen API-Klassen können dann Methoden implementieren, welche über http-Anfragen ausgeführt werden.

Des Weiteren stellt die RestInterface-Klasse einen Controller bereit, der das Verhalten der Anwendung definiert. Durch ihn wird bestimmt, in welchen Intervallen das Gewächshaus reguliert werden soll und wann Daten erfasst und gespeichert werden sollen.

--Schmidt—

3.2 Aufbau – Frontend

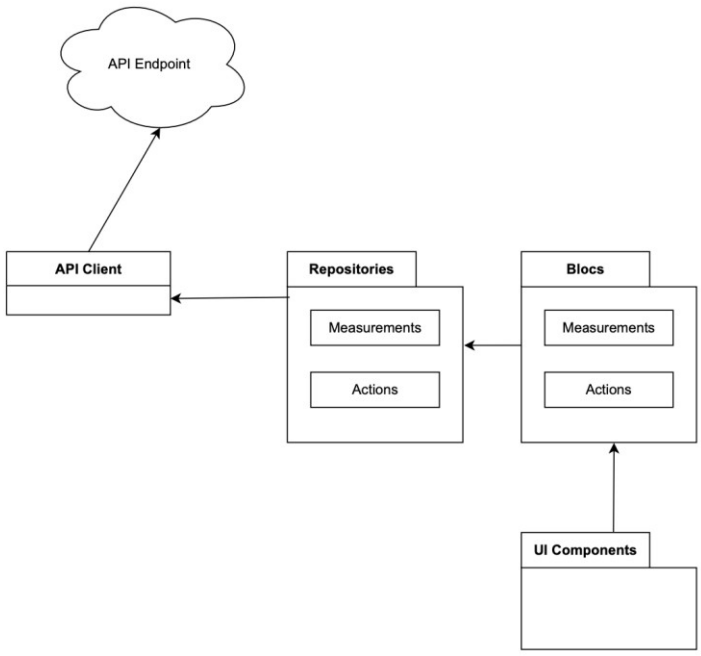
Der Aufbau des Frontend der Applikation, welche wie bereits erwähnt mit Hilfe des Multiplattform-Dart-Framework Flutter implementiert wurde, erfolgt nach dem sogenannten BLoC-Pattern. Dieses dient zum State-Management und trennt ähnlich wie das MVC-Pattern UI-Komponenten von der Datenhaltung und Business-Logik der App.

Dazu kommen die sogenannten BLoCs (und zugehörige Hilfsklassen) zum Einsatz, welche den aktuellen State speichern und alle Operationen zu dessen Veränderung bereitstellen.

Zur Kommunikation mit dem Backend werden von den Funktionen der BLoCs dann Kommunikations-Funktionen welche über sogenannte Repositories bereitgestellt werden verwendet. Diese führen unter Verwendung des API-Clients wiederum HTTP-Requests an die Backend-API aus.

Die UI-Komponenten werden bei jeglicher Änderung innerhalb des States eines mit ihnen verbundenen BLoCs informiert und bei Bedarf neu von der Grafik-Engine der Flutter-Plattform auf den Bildschirm des Geräts gezeichnet.

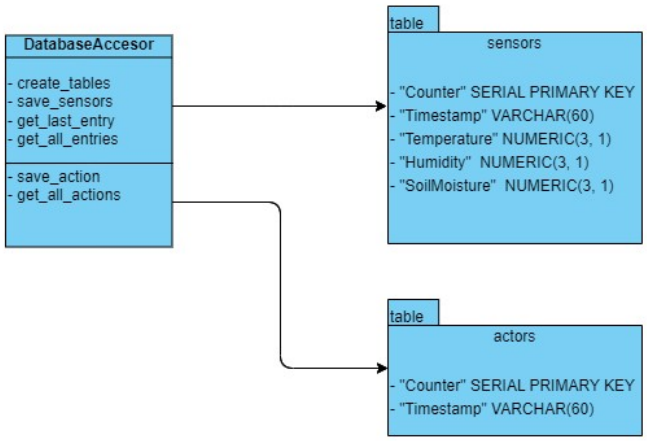
--Söll--



3.3 Aufbau Datenbank

Die Datenbank besteht aus zwei Tabellen. Diese werden erstellt, gelesen oder beschrieben von der DatabaseAccessor-Klasse. Die Tabelle sensors enthält alle Messdaten der Sensoren, wobei der Primärschlüssel ein sogenannter Counter vom Typ Serial ist. Über ihn kann ganz einfach der letzte Eintrag gefunden werden und erleichtert somit die Abfragen. Die Tabelle actors enthält dagegen alle Zeitpunkte, an denen eine Bewässerung stattgefunden hat. Hier ist ebenfalls ein Counter der Primärschlüssel, um Anfragen zu erleichtern.

--Schmidt--



4. Erreichte Funktionen

Im Folgenden Abschnitt werden die erreichten Funktionaltäten genau beschrieben. Sie bilden die Hauptbestandteile der Applikation und sind essenziell für erfolgreichen Betrieb der Anwendung. Sie sind das Ergebnis der in Punkt zwei aufgeführten Funktionalitäten.

--Schmidt--

4.1 Datenhaltung

Die verwendete Datenbank ist eine PostgreSQL Datenbank welche auf dem Raspberry Pi läuft. Da unser Steuerungsskript in Python geschrieben ist, bietet das psycopg2 Modul eine gute Möglichkeit die Datenbank anzubinden. Hierfür haben wir eine Klasse „DatabaseAccessor“ welche über folgende Methoden verfügt:

- create_tables: Erstellt die benötigten Tabellen.
- save_sensors: Speichert die übergebenen Sensorwerte in der Tabelle für die Messwerte ab.
- get_last_entry: Liefert den aktuellsten Messwerteintrag aus der Tabelle zurück.
- get_all_entries: Liefert eine Liste aller Messwerteinträge aus der Tabelle zurück.

- save_action: Fügt einen Eintrag für eine ausgeführte Aktion der Tabelle für die Aktionen hinzu.
- get_all_actions: Liefert eine Liste aller Aktionen aus der Tabelle zurück.

--Zeger--

4.2 Schnittstelle

Um die Kommunikation zwischen App und Backend zu ermöglichen benötigen die Anwendung eine Schnittstelle, über die das Backend und die mobile App kommunizieren können. Dafür wird eine Rest-API implementiert, die verschiedene Endpunkte besitzt und die jeweils eine Aufgabe haben:

- /getMeasurements: Endpunkt über welchen eine Liste aller aufgezeichneten Messwerte zur Verfügung gestellt wird.
- /getActions: Endpunkt über welchen eine Liste aller aufgezeichneten Messwerte zur Verfügung gestellt wird. Zusätzlich besitzt der Endpunkt eine POST-Funktion, über die ein Bewäserungszyklus gestartet werden kann.
- /getImage: Endpunkt über welchen ein Bild der eingebauten Kamera angefragt werden kann.
- /getTSH: Endpunkt über welchen die aktuellsten Messwerte der Sensoren abgefragt werden können.
- /setEnv: Endpunkt mit POST-Funktion, über die das Verhalten des Gewächshauses neu eingestellt werden kann. Dabei wird angegeben, wie lange ein Bewässerungszyklus dauern soll und in welchen Intervallen das Gewächshaus die Sensordaten auslesen soll.

--Zeger—

4.3 Elektronik

Die Messwerte die von der Anwendung verarbeitet werden, stammen von verschiedenen Sensoren, die über eine Elektroschaltung an den Raspberry Pi angeschlossen ist. Ein DHT22 AM2302 Sensor misst die Lufttemperatur und die Luftfeuchtigkeit. Das Hygrometer Modul V1.2 ist ein kapazitativer Sensor, der die Bodenfeuchtigkeit bestimmt. Da dieser Sensor jedoch nur analoge Daten liefert, wird ein MP3002 ADC Microchip verwendet, der die analogen Daten in digitale Daten umwandelt. Eine 12 Volt Wasserpumpe dient als Aktor und ermöglicht die Bewässerung. Die Funktionsfähigkeit der Schaltung ist für den Betrieb der Anwendung ebenfalls essenziell, da ohne Sie keine Daten erhoben werden können.

--Schmidt--

4.4 Mobile Anwendung

Die mobile Applikation ist ebenfalls ein Hauptbestandteil der Anwendung. Sie stellt dem Benutzer alle möglichen Interaktionen mit dem System zur Verfügung. So kann der Benutzer über verschiedene Anzeigen, Eingabefelder oder Buttons sowohl die Daten eines Gewächshauses bestimmen, als auch Aktionen durchführen wie etwa die Bewässerung. Sie kommuniziert dabei mit der Rest-Schnittstelle des Backend und ist zusätzlich in der Lage live Bilder aus dem Gewächshaus anzuzeigen. Dadurch ist der Benutzer jeder Zeit in der Lage den Zustand des Gewächshauses zu kontrollieren.

--Schmidt--

5. Hardware, Software und Bibliotheken

Im folgenden Abschnitt wird jede verwendete Hardware, Software und sämtliche externe Bibliotheken, die verwendet werden aufgelistet und erläutert.

--Schmidt--

5.1 Frontend

Der Programmcode für das Frontend ist wurde in Flutter 3.3.10 geschrieben.

Hardware:

- iPhone 12

Software:

- Xcode als Entwicklungsumgebung.
- IntelliJIDEA mit Flutter-Plugin.
- Virtual Device Simulator (Apple).

Bibliotheken:

- Flutter_bloc, um Programmcode für die Benutzeroberfläche von logischen Programmcode zu trennen.
- Equatable, um den ==-Operator und HashCode-Funktionen zu überschreiben.
- Dio, um einen http-Client zu erstellen.
- http, um http-Anfragen zu erstellen.
- Flutter_speedometer, um die Tachobestandteile der Benutzeroberfläche zu erstellen.

--Söll, Röcker, Schmidt--

5.2 Backend

Der Programmcode des Backend wurde in Python 3.7 geschrieben.

Hardware:

- RaspberryPi 4
- 12V Wasserpumpe
- DHT22 AM2302 Temperatursensor
- Hygrometer Modul V1.2 kapazitativ
- PiCamera
- MP3002 ADC Chip

Software:

- Visual Studio Code als Entwicklungsumgebung.

Bibliotheken:

- Flask, um eine Rest-Schnittstelle zu erstellen.
- Postgresql, um CRUD-Operationen auf einer PostgreSQL-Datenbank auszuführen.
- Psycopg2, um einen PostgreSQL-Driver zu erstellen für die Verbindung mit einer PostgreSQL-Datenbank.
- Adafruit_dht, um den DHT22 AM302 Sensor auszulesen.
- Spidev, um die digitalen Daten des MP3002 ADC Chip auszulesen.
- Flask-restx, um eine Swagger-Dokumentation zu erhalten.

--Schmidt--

6. Fazit

Im folgenden Abschnitt werden die Erfahrungen und das Fazit der einzelnen Gruppenmitglieder aufgeführt.

6.1 Christian Schmidt

Im Zuge des Projekts, habe ich gelernt mich mit technischen Bauteilen wie dem DHT22 AM2302 Temperatursensor, einem kapazitativen Hygrometer Modul V1.2 oder einem MP3002 ADC Chip auseinanderzusetzen, deren Funktionsweise zu verstehen und sie mit Hilfe eines RaspberryPi 4 auszulesen. Des Weiteren habe ich durch den Aufbau der elektrischen Schaltungen, grundlegende Kenntnisse in der Elektronik erarbeitet. Darunter beispielsweise, wie man einen zweiten Stromkreis in einen Hauptstromkreis integriert und warum eine gemeinsame Erdung der Schaltkreise nötig ist. Ich habe außerdem gelernt wie man Transistoren bzw. Mosfets als Schalter verwenden kann. All diese Erfahrungen waren sehr spannend zu machen und bereiteten spaß. Neben den elektronischen Aspekt, habe ich das Backend für die Applikation entwickelt und dabei gelernt, wie man mit Python die zuvor

genannten Sensoren und Aktoren ansteuern und auslesen kann. Ebenfalls habe ich gelernt wie man eine Rest-Schnittstelle unter Verwendung des Python-Moduls Flask erstellt.

Das Arbeiten mit Git und Trello hat den gesamten Entwicklungsprozess unterstützt und hat eine gute Kommunikation innerhalb der Projektgruppe ermöglicht.

Da ich all dies und mehr durch das Projekt gelernt habe, bin ich mit dem Ergebnis sehr zufrieden.

6.2 Robin Röcker

Ich habe gelernt wie man die bloc-Bibliothek verwendet, um den Programmcode der Benutzeroberfläche von dem Programmcode der Systemlogik zu trennen. Des Weiteren habe ich gelernt, wie man das speedometer-Widget in Flutter verwendet. Zusammenfassend kann ich sagen, dass das Projekt trotz den ein oder anderen Schwierigkeiten ein Erfolg war. Wir haben alle Funktionen implementieren können und waren in der Lage eine ansprechende App zu erstellen.

6.3 Maximilian Zeger

Das Projekt hat mir gezeigt wie wichtig frühzeitige Planung und Vorbereitung sein kann da sich dabei kleine Fehler schnell ausbreiten können. Außerdem habe ich gelernt wie man eine PostgreSQL Datenbank installiert, aufsetzt und die Verbindung dazu über ein Python Skript herstellt. Dabei gab es zunächst Schwierigkeiten mit der PostgreSQL Syntax, welche aber durch die umfassende Dokumentation schnell zu beheben war. Für mich persönlich waren viele unserer benutzten Technologien, wie z.B. Flutter, Elektronische Bauteile und REST nur wenig bekannt und ich habe nun einiges darüber gelernt.

6.4 Yannick Söll

Ich habe gelernt, wie man in Flutter mit Code-Generierung in Verbindung mit der BloC-Library arbeitet (auch wenn dies in der Endversion nicht enthalten ist) und wie man eine korrekte Kommunikation mit einer API-Schnittstelle eines Backend via Repositories implementiert. Dazu kommen weitere Erfahrungen im Bereich des UI-Buildings, durch die ich meine bisherigen Kenntnisse in diesem Feld erweitern konnte.

Auch wurde mir bewusst, welche Schwierigkeiten durch ein Gruppenarbeit entstehen können, wenn man sich selten persönlich sieht und gemeinsame Termine in einem vollen Zeitplan finden muss.

Im Großen und Ganzen war das Projekt ein Erfolg, auch wenn nicht ganz alle Funktionalität erreicht werden konnte, haben wir am Ende unsere Anfangs-Vision umsetzen können.

6.5 Team

Als Team haben wir gelernt, unsere Arbeit klar zu strukturieren, so wie Aufgaben untereinander zu verteilen. All dies unter Verwendung eines Trelloboards und einem Git-Repository, welche diesen Prozess unterstützen. Im Laufe des Projekts sind wir immer wieder auf Probleme gestoßen, die wir überwinden konnten, um daraus zu lernen. Ein Großteil davon bildete der Elektronische Teil, wodurch uns klar wurde, dass dieser Teil einen erheblichen Anteil bei der Entwicklung von eingebetteten Systemen darstellt.

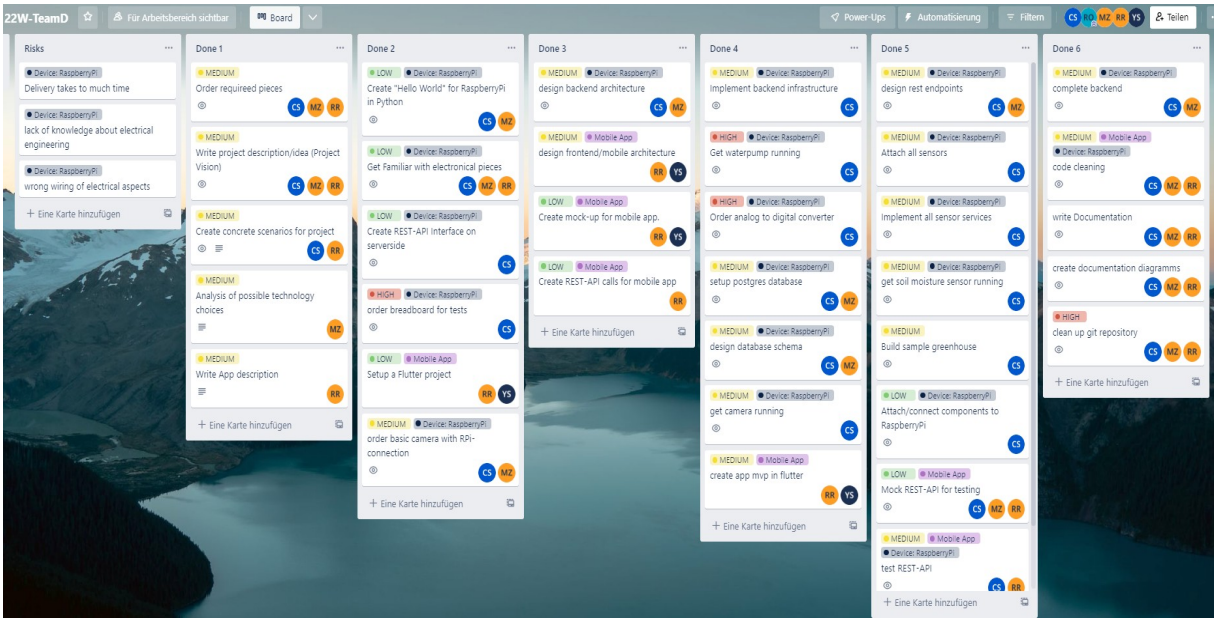
Für uns alle war dies das erste Projekt im Bereich der eingebetteten Systeme und so war das Projekt für alle von uns eine spannende Erfahrung. Ganzheitlich lässt sich sagen, dass wir uns immer noch in der Kommunikation innerhalb eines Teams verbessern können, um unnötige Missverständnisse zu vermeiden.

Mit dem Ergebnis des Projekts sind wir dennoch sehr Zufrieden und freuen uns, dass wir alles umsetzen konnten was wir uns vorgenommen haben.

7. Anhang

Im folgenden Abschnitt befindet sich das finale Trelloboard und alle Aufschriebe der jeweiligen Sprint Reviews. Dabei enthalten die Reviews lediglich den Text der Reports und keine Bilder.

7.1 Finales Trelloboard



7.2 CONOPS MS Report

<u>Project vision</u>

- The Smart Greenhouse automatically regulates the temperature and soil-moisture.
    - It is connected to a mobile App.
    - In the App you can see the temperature, soil-moisture, humidity and a live view.
    - The App also has a diary, in which you can see which actions were took and a history of the measured parameters.
    - Optionally you can see the light intensity and the nutrient enrichment of the soil.

<u>Concept of Operations</u>

- The Smart Greenhouse regulates itself:
    - When the temperature is below a certain point
        - The heating pad is switched on.
    - When the soil-moisture is below a certain point

        - The water pump is switched on.
    - The heating pad & water pump will switch off when a certain value is reached.
- In the app you can see:
    - The current temperature, moisture & soil-moisture.
    - A live view of your plant.
    - A history of the temperature, humidity & soil-moisture at an hourly interval.
    - The actions took by the smart greenhouse for each day.

Technology choices

- Rasperry Pi
    - The Arduino platform isn’t suitable for complex features like a live camera view.
    - The Arduino platform doesn’t have a wifi/lan connection by default.
    - The Arduino platform doesn’t support high level programming languages like python.
    - The Arduino platform doesn’t have a OS.
- Python
    - Python is an easy language with many modules.
    - Disadvantage of being an interpreter language doesn't affect our project.
    - Python-Flask allows to build a simple REST-API.
    - Provides a high level language support.
- Flutter
    - Biggest community of all cross-platform technologies.
    - Runs on mobile, desktop & web.
    - Best performance of all cross-platform technologies.
    - Offers the greatest freedom in designing the user interface.
    - Offers the best third-party support of all cross-platform technologies.

7.3 Tech Demo MS Report

Technology choices

- Rasperry Pi
    - The Arduino platform isn’t suitable for complex features like a live camera view.
    - The Arduino platform doesn’t have a wifi/lan connection by default.
    - The Arduino platform doesn’t support high level programming languages like python.
    - The Arduino platform doesn’t have a OS.
- Python
    - Python is an easy language with many modules.
    - Disadvantage of being an interpreter language doesn't affect our project.
    - Python-Flask allows to build a simple REST-API.

    - Provides a high level language support.
- Flutter
    - Biggest community of all cross-platform technologies.
    - Runs on mobile, desktop & web.
    - Best performance of all cross-platform technologies.
    - Offers the greatest freedom in designing the user interface.
    - Offers the best third-party support of all cross-platform technologies.
- PostgreSQL-Database
    - Easy setup
    - Provides a easy data query language
    - Ensures data persistence
    - Team members are already familiar with SQL
    - Has great community support
    - Widespread popularity

7.4 Architectural Spike MS Report

- Only Images-

7.5 Alpha MS Report

What did we accomplish?

- Communication between mobile app and backend via REST-API
- Setup of the database is done and the required transactional methods are implemented
- The backend was designed according to the UML diagrams
- Air temperature and humidity sensor is working
- Water pump can be controlled via python script (on/off mode)
- Camera is attached to the Raspberry Pi and can be controlled via python script
- Mock up for the REST-API can be used for testing and demonstration

What to do next?

- Build the sample greenhouse
- Attach all actors and sensors to the Raspberry Pi
- Build the electrical circuit
- Attach the electrical circuit to the greenhouse
- Implement the final backend infrastructure
- Register all sensors as services in the backend
- Implement the main control loop of the backend
- Finish the mobile app design and functionality

Challenges/risks

- Electrical pieces are missing (ADC- analogue to digital converter), which is needed for the soil moisture sensor
- Sensors have call-back that need to be handled

7.6 Beta MS Report

What did we accomplish?

- Communication between mobile app and backend via REST-API
- Changes of the Database to reach better communication and less latency
- Full implementation of the Backend
- Added multithreading for REST-API and control aspects
- All sensors and actors are integrated and are working
- Greenhouse sample is built and the sensors and actors are attached to it
- Mobile app design is finished

What to do next?

- Fix parsing problem of /getMeasurements-Endpoint
- Fix transaction issue with images, use correct base64 encoding
- Write documentation - Make a video of all scenarios
- Provide mobile functionality to set parameters of the control loop (watering duration, control interval)
- Do code cleaning and author documentation
- Update Git

Challenges/risks

- Electrical pieces take damage due to transportation
- Sensors have call-back that need to be handled
- Mobile hotspot is stable and grants good connection