

LeaveLifeLove

Android App für Pflanzen Überwachung


Ziel: Wir wollen eine App programmieren, welche mit Hilfe weiterer Technologien einen Statusbericht über die hinzugefügte Pflanze preisgibt. In dieser App soll es idealerweise möglich sein, die Feuchtigkeit der Erde, Temperatur und Lichteinfall zu tracken. Diese Werte sollen mit hinterlegten Informationen einer Datenbank abgeglichen werden, in welcher die optimalen Werte für jeweilige Pflanzen bereitgestellt sind. Die Werte sollten aber auch einzeln angezeigt werden, in einem wunderschön gestalteten GUI. Des Weiteren verfügt die App über einen Kostenrechner, der anhand der Ausgaben wie Material und Energiekosten unterschiedliche Verkaufspreise für den Ertrag errechnet. Zudem soll der Nutzer der App Benachrichtigungen erhalten und eine Refresh der aktuellen Daten anstoßen können.

Technologien: Das Ziel ist es, einen ESP32 8266 mit den jeweiligen Sensoren zu verbinden und an der Pflanze / im Topf zu befestigen. Strom wird durch einen Akku bereitgestellt. Dieser ESP sollte dann in regelmäßigen Abständen die Werte an das Handy direkt oder an eine externe Datenbank senden. Möglich dafür wären die Technologien WLAN. Da Bluetooth würde zwar durch den Low Energy Modus stromsparender sein, hätte aber nicht die größte Reichweite für unsere App. Optimal werden die einzelnen Sensoren + Akku in einem schicken 3D-gedruckten Design aufbewahrt.

Bonusfeatures: Jeweils einen Account erstellen, sodass mehrere Pflanzen mit Spitznamen dem Nutzer zugefügt werden können. Auch allgemeine Tipps für die jeweilige Pflanze direkt nachlesen. (Bilder aus einer Datenbank mit häufigen "Krankheitssymptomen", Falls möglich sollte auch an jedem Gerät ein NFC-Tag beigelegt werden, sodass Fremde mit Hilfe der App den aktuellen Status der Pflanze (muss man sie gießen?), abrufen können. Auch Benachrichtigungen für schlechte Werte, wie Umtopf Datum, sollen erscheinen. Oder ein Fotoalbum für die jeweilige Pflanze.


Startbildschirm

Welcome to



LeafLifeLove(sosa)

Login



Sign up

e-mail

password


Start

you already have an Account ?

Login

Home

Pflanzenkartei Details



+3°

5/9

70%

70%

Menu

17.04.2024 15:10

Name

Gattung

Herkunftsregion

Schwierigkeitsgrad

zur Galerie

Abweichung von Idealtemperatur

Sonnesstunden

Bodenfeuchtigkeit

CO2 Verfügbarkeit

Technologien

ESP8266 + Sensoren

An diesen Mikrocontroller werden die jeweiligen Sensoren angeschlossen. Im ESP läuft dabei ein Mikrophyton Skript durch, welches diese Werte in regelmäßigen Abständen von den Sensoren abliebt und an die InfluxDB sendet.

MQTT-Broker (HiveMQ) + Telegraf

Der MQTT Broker ist eingerichtet und empfängt jeweils von dem ESP neue Daten. Die Topics sind dabei die einzelnen Sensoren. Der Telegraf wird dabei so eingerichtet, die jeweiligen Werte in einer Datenbank abzuspeichern zusammen mit Wert und Uhrzeit.

--> anstelle der Übertragung mit MQTT und Telegraph übertragen wir die Werte direkt in die Datenbank. MQTT ist sinnvoll, wenn mehrere Sensoren+ESPs auf die Datenbank übertragen sollen. Die App soll aktuell allerdings mit nur einem ESP funktionieren, damit spart man sich einen Server aufzusetzen, der dieses MQTT Übertragungen bearbeitet.

InfluxDB

Die Datenbank enthält die jeweiligen Werte gespeichert vom ESP sowie eine Datenbank mit jeweiligen Musterwerten für einzelne Pflanzenarten, welche von den Admins analog einmalig in eine Bibliothek hinzugefügt werden. Gelöscht werden die Daten automatisch nach einer bestimmten Zeit um die Datenbank nicht zu voll zu haben.

Komponenten

Model

Im Model werden die jeweiligen Werte aus der Datenbank in die jeweilige App übertragen. Dabei werden die vorherigen Werte überschrieben.

Hauptaufgabe dieses Teiles ist es dabei die jeweiligen Werte an die anderen Komponenten weiterzugeben. Oder bei Benutzereingaben, diese zu aktualisieren.

User-Database: Speichert Daten des einzelnen Nutzers.

Network Connection-Library: Speichert die einzelnen Daten, wie Wlan-Name und Passwort und eine Verbindung mit dem Server bereit zu stellen.

Sensor-Plant-Data Values: Bezieht aus einer Datenbank die aktuellen Werte des Sensors.

Plant-Repository: Speichert die jeweiligen vom Benutzer eingefügten Pflanzen so wie Eigenschaften.

Recommended Plant-Value-Library: Eine vom Admin bereitgestellte Datenbank mit Idealwerten für bestimmte Pflanzenarten.

Test: Datenbanken analog überprüfen, ob die jeweiligen Werte mit den Wunschwerten übereinstimmen

Controller

Im Controller werden dann die jeweiligen Werte, welche vom Model gesendet werden, für den entsprechenden Use-Case umzurechnen und an das View zu senden. Außerdem werden dabei noch die jeweiligen Benutzereingaben auch verarbeitet und an das Model gesendet. Der Controller nimmt die Benutzereingaben aus dem View entgegen und löst entsprechende Aktionen im Model aus. Zum Beispiel könnte er das Model anweisen, Daten zu aktualisieren oder spezifische Berechnungen durchzuführen.

User-Data Controller: Verwaltet die Nutzerdaten, welche durch den User-Setting-View geändert werden und passt diesen in der Datenbank User-Database an.

Network-Connection-Controller: Nimmt die Werte aus der Network-Connection-Library und stellt eine Verbindung zum Server her, um dadurch die Sensor-Werte für Sensor-Plant-Data-View bereit zu stellen.

Notification-Controller: Nimmt die Werte vom Sensor-Plant-Data-Values und verarbeitet bei kritischen Werten eine Benachrichtigung, welche von Notification-View angezeigt wird.

Plant-Value-Controller: Rechnet aus dem Plant-Repository und der Recommended Plant-Value-Library, die Werte aus welche von View angezeigt werden sollen.

Test: JUnit-Test schreiben, um alle Möglichkeiten zu überprüfen und für gegebene Werte zu überprüfen.

View

In der View werden die vom Controller weitergegebenen Werte dann anschaulich im GUI dargestellt, sodass der User diese ablesen kann. Die Werte handeln sich dabei um die Werte, welche für die Pflanze wichtig sind.

Der Nutzer kann außerdem durch das GUI, eine jeweilige Pflanze auswählen und dieser Eigenschaften einfügen. Diese werden dann über den Controller im Model gespeichert. Der View reagiert dabei auf alle weiteren Benutzeraktionen und Eingaben.

User-Data-View: Stellt die Daten des Users bereit

User-Settings-View: Agiert als Schnittstelle um Benutzereingaben zu bearbeiten

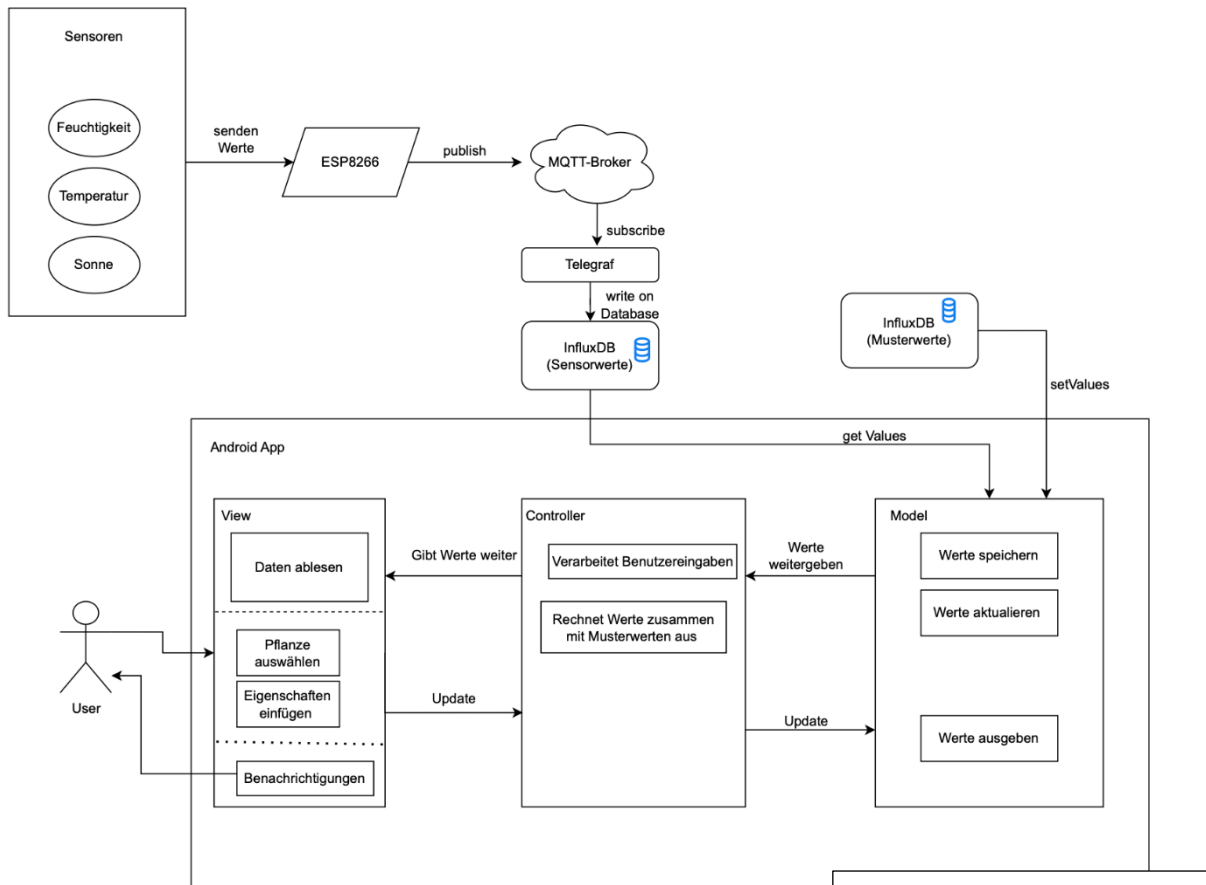
Notification-View: Zeigt Benachrichtigungen bei kritischen Werten an, welche von dem Notification-Controller verarbeitet werden.

Plant-Data-View: Zeigt die berechneten Werte für die jeweilige Pflanze an.

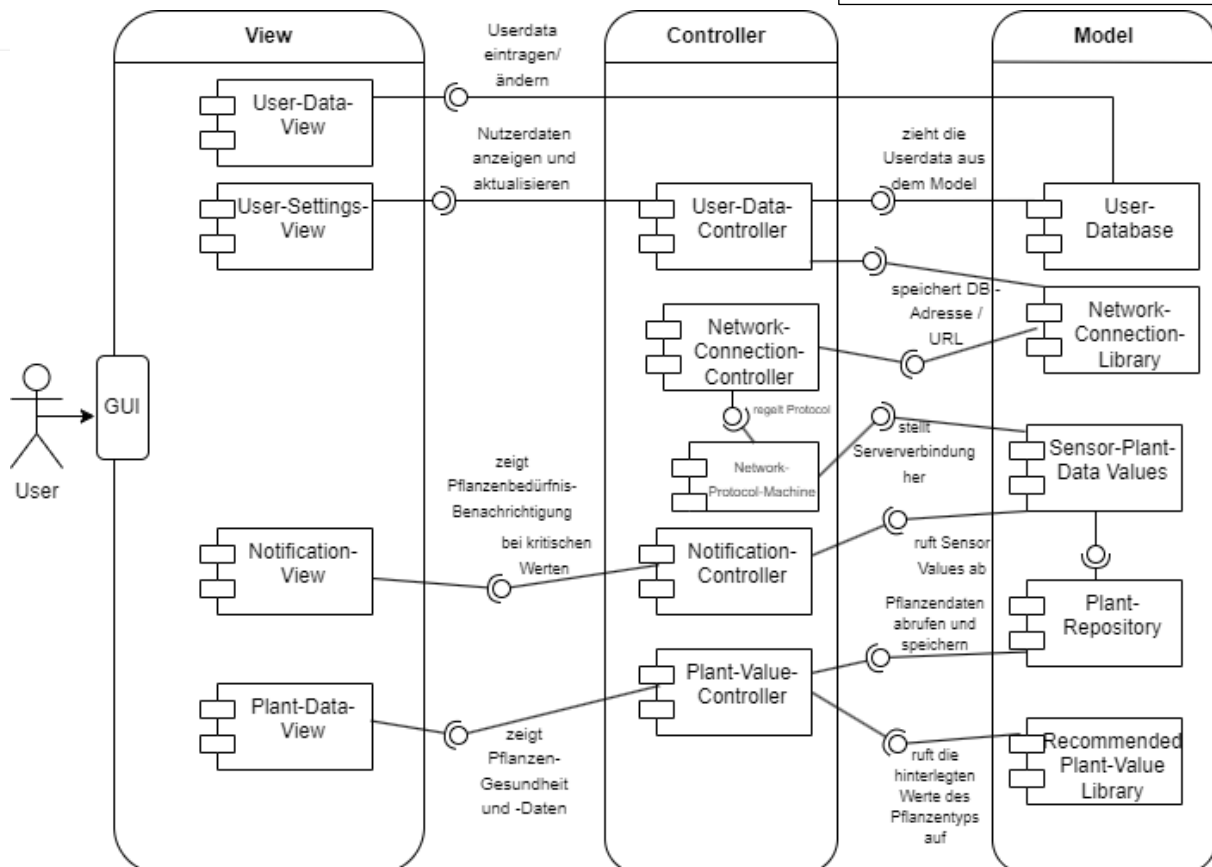
View: Selbstständiges Prüfen aller Werte mit Beispielen, um Richtigkeit zu vergleichen.

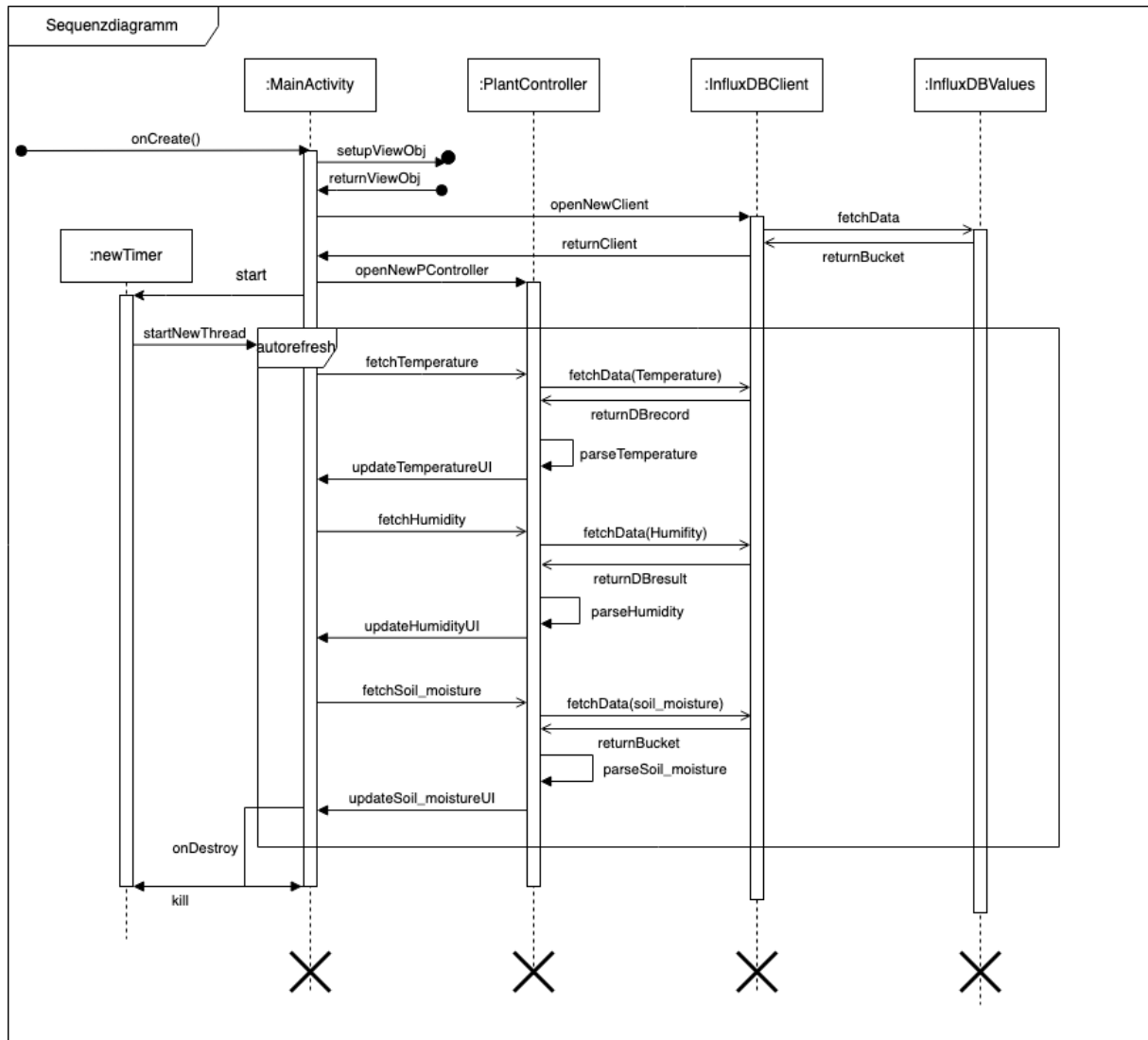
Darstellung:

Technologien im Zusammenspiel



Komponentendiagramm





Sequenzdiagramm der MainActivity beim Werte anzeigen

Tests:

Use Cases innerhalb der MVC Komponenten:

GUI/VIEW:

Die neue Pflanze Anlegen:

- Die Daten für die Pflanze abfragen: Kann der Chip mit der Datenbank kommunizieren?
- Aktiv: Können die aktuellen Werte abgefragt werden?
- Passiv: Werden die Werte periodisch neu abgefragt?

Espresso Tests:

Version:	Anwendung:	Erläuterung:	Name oder Code:	Erfolgreich (J/N)
Espressolisten	Plant-Data-View: Sensordatenübertragungswerte Monatsansichten	Es gibt eine Ansicht, in der die gesamten Messwerte (Bsp. von einem Monat). Da kommen mitunter sehr lange Listen zu Stande, welche stickprobenweise mit Espressolisten getestet werden können.	hasEntry(equalTo("VALUE_FROM_XY_SENSOR").is("item: 5"));	TODO
Espresso	User-Data-View: Pflanze neu anlegen Button	Wenn der Button betätigt wird, muss der Prozess zum Anlegen einer Pflanze gestartet werden -> im Scope hier: Neues Feld geht auf, um die Eingabe der Spezifikationen entgegenzunehmen	onView(withId(R.id.button_add_new_plant)).perform(click()); onView(withId(R.id.editTextPlantType)).check(matches(isDisplayed()));	TODO
Espresso	User-Data-View: Kartei_View Button	Blendet die Kartei der Pflanzen ein ?	onView(withId(R.id.pflanzenkarteiButton)).perform(click());	TODO
Espresso	User-Settings-View: Auswahl_Pflanzen Button (alternativer View zu Kartei_View)	Blendet die Übersicht der Gespeicherten Pflanzen ein?	onView(withId(R.id.pflanzenuebersichtButton)).perform(click());	TODO

Espresso	Update Button	<p>Der Button um eine außerperiodische Neuankündigung die aktuell zuletzt gemessenen Daten aus der Datenbank anfordert. Es müssen hier neue Werte angezeigt werden,</p> <p>ggf. Ändert sich auch der Timestamp der Letzen Messung. Um einen Aussagekräftigen Test durchzuführen müssen die zu messenden Werte am Sensor manipuliert werden, dass eine Nicht Veränderung der Werte durch Nicht Veränderung der gemessenen Bedingungen ausgeschlossen werden kann.</p> <p>bei Refresh Buttons ist es aus Sicht der UX wichtig ein Feedback zu geben. Dieses kann ebenfalls getestet werden.</p> <p>Verschwindet dieses wieder?</p>	<p>testUpdateButtonShowsSnackBar()</p> <p>testUpdateButtonShowsSnackBar()</p> <p>testSnackBarDisappears()</p>	<p>Ja</p> <p>TODO</p> <p>Ja</p> <p>Ja</p>
Espresso	Notification-View:	Wenn aus der Datenbank die Werte mit den Sensorwerten auch aus der DB ablesen und verrechnen können Dringliche Nachrichten an den Nutzer ausgegeben werden (Bsp. Licht ist unplanmäßig ausgegangen, Temperatur wurde ausserhalb des Toleranzbereichs gemessen, Pflanze muss getrocknet werden)	onView(withId(R.id.showTrackingNotification)).check(matches(isDisplayed()));	TODO
Espresso	DisplayedValues	<p>Wenn die App geöffnet wird, wird „... Loading...“ Angezeigt Dies kann getestet werden</p> <p>Nachdem der View der MainActivity geupdated wird, werden der Name der</p>	<p>testLoadingIsDisplayedAtStart()</p> <p>testValuesContainOnlyAllowedSymbols()</p>	<p>Ja</p> <p>Ja</p>

		<p>Messwerte sowie ein Wert aus den Ziffern von 0-9 und der Einheit mit °C und % angezeigt</p> <p>Da wir in den Espresso tests nicht testen wollen, ob die Werte Korrekt vom Sensor erfasst und übertragen wurden, sondern den View testen, wird nach plausiblen werten (s.o.) geprüft</p>		
--	--	--	--	--

Sensoren: Sensoren werden in Ihren realen Bedingungen getestet. Bsp. Temperatur-Sensor wird mit einem Thermostat verglichen. Danach kann man bei allen Sensoren davon ausgehen, dass Sie funktionieren und die richtigen Werte senden.

Parallel kann man dabei überprüfen, ob die Datenbank und die verschiedenen Libraries, die für diese Datenbank erstellt wurden für Arduino und Java funktionieren. Wir gehen davon aus, dass diese auch weiterhin funktionieren werden.

User-Data Controller:

Was	Testname	Wie	Erfolgreich? (J/N)
Werte richtig in User-Data-Library eingetragen	SendUserDataTest()	JUnit-Test	TODO

Network-Connection-Controller:

Was	Testname	Wie	Erfolgreich? (J/N)
Liefert der ConnectionController bei fehlerhafter Verbindung „Keine Werte vorhanden“ zurück ins View.	ConnectionErrorValues()	Selbst	Ja (liefert „Loading...“ zurück als Wert)
Liefert der ConnectionCreator eine Exception, wenn die Connection nicht hergestellt werden konnte.	ConnectionErrorException()	JUnit-Test	Ja
Liefert der ConnectionCreator eine Exception, wenn ihm null als Parameter weitergegeben wird	ConnectionNullError()	JUnit-Test	Ja

Network-Protocol-Machine:

Was	Testname	Wie	Erfolgreich? (J/N)

Protocol zur richtigen Erstellung einer Connection zur InfluxDB	protocolConnectionTest()	JUnit-Test	Ja
Liste aus DB-Werten wurde Erfolgreich erstellt.	testFetchDataList()	JUnit-Test	Ja
Exception bei keinen Werten	noListValuesTest()	JUnit-Test	Ja (Fehler wird ausgegeben)

Notification-Controller:

Was	Testname	Wie	Erfolgreich? (J/N)
Benachrichtigung wird gesendet bei eingestellten Werten	NormalNotificationTest()	UIAutomator	TODO

Plant-Value-Controller:

Was	Testname	Wie	Erfolgreich? (J/N)
Analoge Werte von Bodenfeuchtigkeit auslesen und in Prozent umrechnen für den User	plantValueCalcMoisture()	JUnit-Test	Ja
Temperatur auslesen	plantValueCalcTempt()	JUnit-Test	Ja
Luftfeuchtigkeit auslesen	plantValueCalcHUm()	JUnit-Test	Ja
Fehler ausgeben, wenn die Werte außerhalb der von den Sensoren zulässigen Werten ist.	wrongPlantValueCalc()	JUnit-Test	Ja
Exception bei keinen Werten	noListValuesTest()		Ja

Network Connection-Library:

Speicherung der Parameter, welche für die Verbindung mit der InfluxDB benötigt werden. Tests, wie beispielsweise richtige Datentypen, verlaufen dabei durch den „Network-Connection-Controller“, sodass hier keine weiteren nötig sind.

User-Database: Vertrauen, dass die Datenbank richtig agiert.

Sensor-Plant-Data Values: Mit realen Messgeräten vergleichen, um richtige Messwerte zu überprüfen.

Plant-Repository: Vertrauen, dass die Datenbank richtig agiert.

Recommended Plant-Value-Library: Vertrauen, dass die Datenbank richtig agiert.

Aktueller Stand: Git : <https://github.com/muriius/LeafLiveLove.git>

Screenshots:



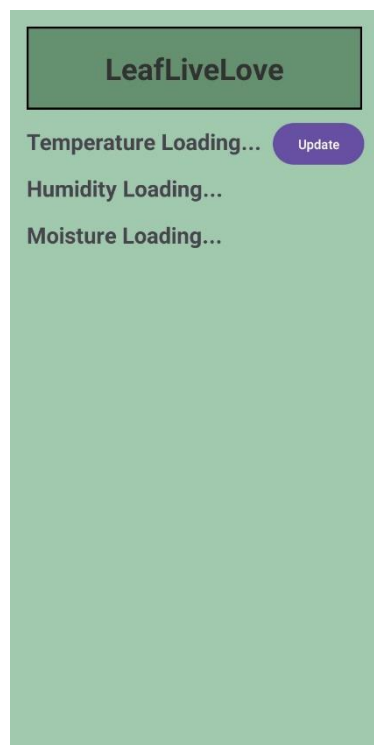
Startscreen



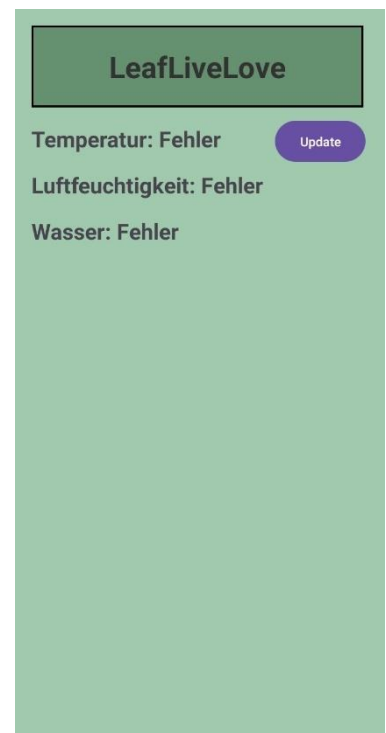
Normaler Screen mit Werten



Splashscreen beim drücken von Update



Fehler mit der Verbindung

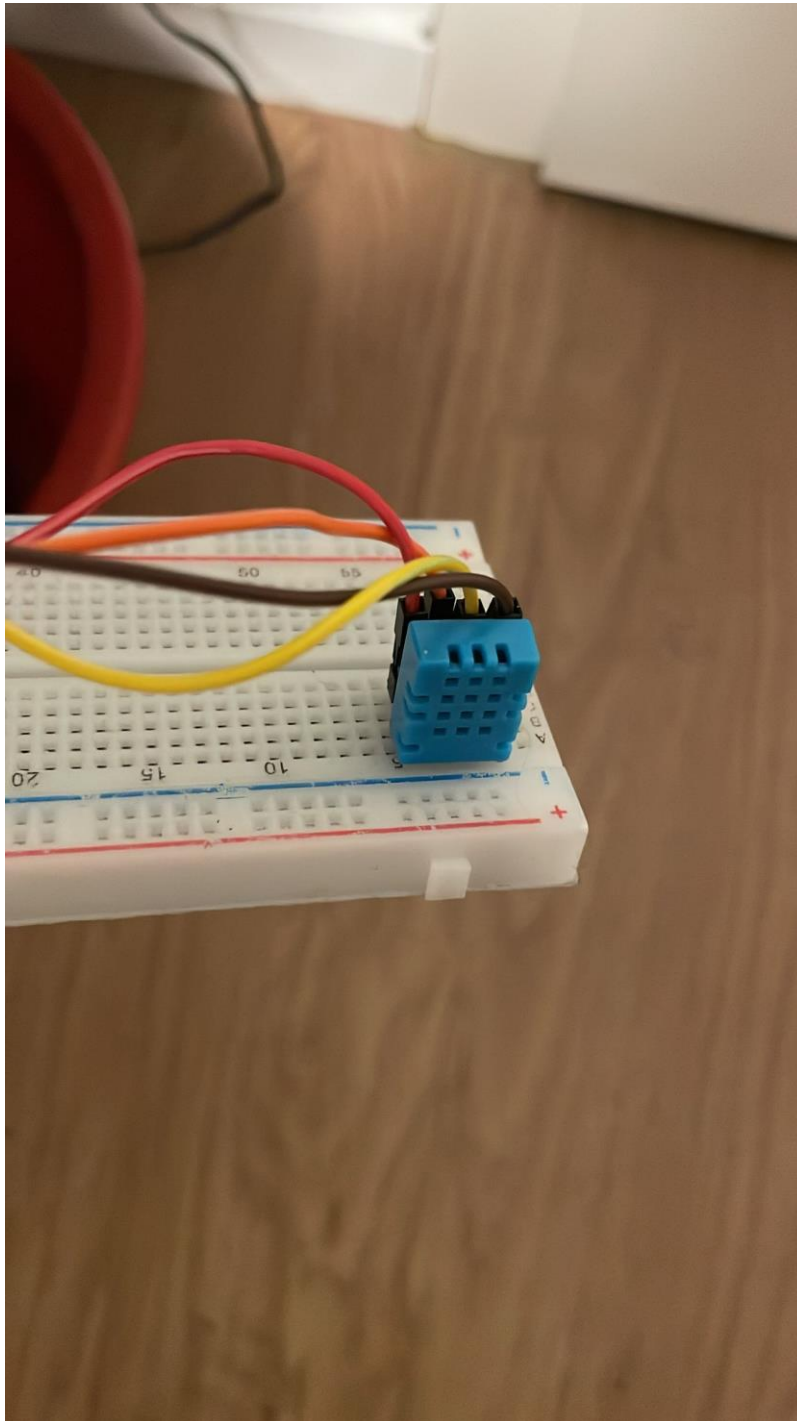


Fehler bei den Werten

Sensor für die Erde



Sensor für Temperatur und Luftfeuchtigkeit



Breadboard mit Esp8266

