

Plaenty

Követelményspecifikáció

Az elkészítendő szoftver egy hydroponic tower felügyelő alkalmazás. A hydroponic növénytermesztés a föld nélküli, vízben való növény termesztés módszere, a torony pedig erre egy vertikális megoldás. A torony alján lévő tápoldatot egy pumpa rendszeresen a tetejére pumpálja, hogy onnan a növények gyökereit megnedvesítve csorogjon le. *(A gyökerek maguk nem ázhatnak folyamatosan tápoldatban, mert akkor megfulladnának.)*



Az alkalmazás képes különböző beavatkozók segítségével szabályozni a növényeknek jutó tápoldat és fény mennyiségét a növényigény-adatbázis alapján. Az alkalmazás logolja és megjeleníti a szenzor értékeket és logolja a beavatkozásokat.

A fejlesztő csapat

Név	Neptun	Email-cím
Ádám Zsófia	SOSK6A	adamzsofi.mail@gmail.com
Mondok Milán	URUGX5	milanmondok1998@gmail.com

A fejlesztő csapat mindkét tagja aktívan részt vesz a feladat minden megoldásának fázisában, dedikált szerepeket nem osztottunk ki előzetesen.

Részletes specifikáció

A projekt során egy olyan hydroponic rendszer felügyelő alkalmazás fejlesztése a célunk, mely képes pumpa és growlight segítségével szabályozni a növényeknek jutó tápoldat és fény mennyiségét. Az intelligens logika a pH és EC szenzoroktól kapott adatok alapján visszajelzést ad a tápoldat állapotáról. A növényeknek a tápoldat adás sűrűségét és fény mennyiségét az alkalmazás az aktuális konfiguráció, a növényigény-adatbázis és a fényszenzor alapján határozza meg.

A szoftver képes tehát:

- Időzített pumpa vezérlésére
- Growlight ki/bekapcsolására
- Adatok fogadására pH, EC és fény szenzoroktól
- A szenzoroktól kapott adatok logolására
- A küldött beavatkozó jelek logolására
- Az aktuálisan mért értékek megjelenítésére a GUI-s kliensben (*grafikonok*)
- A logolt értékek megjelenítésére a GUI-s kliensben (*grafikonok, szélsőértékek*)
- Több különböző konfiguráció tárolására, melyek közül a kliensben kiválasztható az aktív konfiguráció

A kliensben lehetőség van új konfigurációk létrehozására, melyekhez segítséget nyújt a növényigény adatbázis ajánlott default értékek megjelenítésével. A felhasználónak aktív konfiguráció kiválasztásához vagy konfiguráció készítéséhez be kell jelentkeznie.

Az alkalmazásban a szenzorokat egyelőre szimulálni fogjuk.

Technikai paraméterek

Az alkalmazás egy backend és egy frontend komponensből fog állni, melyek egy Rest API-n keresztül kommunikálnak majd. A backendet Java nyelven készítjük el a Spring keretrendszert használva. A backend egy Raspberry PI miniszámítógépre lesz tervezve, ami később majd lehetővé teszi a szenzorokkal való kommunikációt a GPIO interfészen keresztül. A frontend egy Android alkalmazás, szintén Java nyelven írva.

Szótár

Hydroponic növénytermesztés: a növénytermesztés azon módja, amely nem talaj közvetítésével, hanem tápfolyadék használatával működik.

pH szenzor: a tápoldat kémhatását mérni képes szenzor.

EC szenzor: elektromos vezetőképességet mérni képes szenzor.

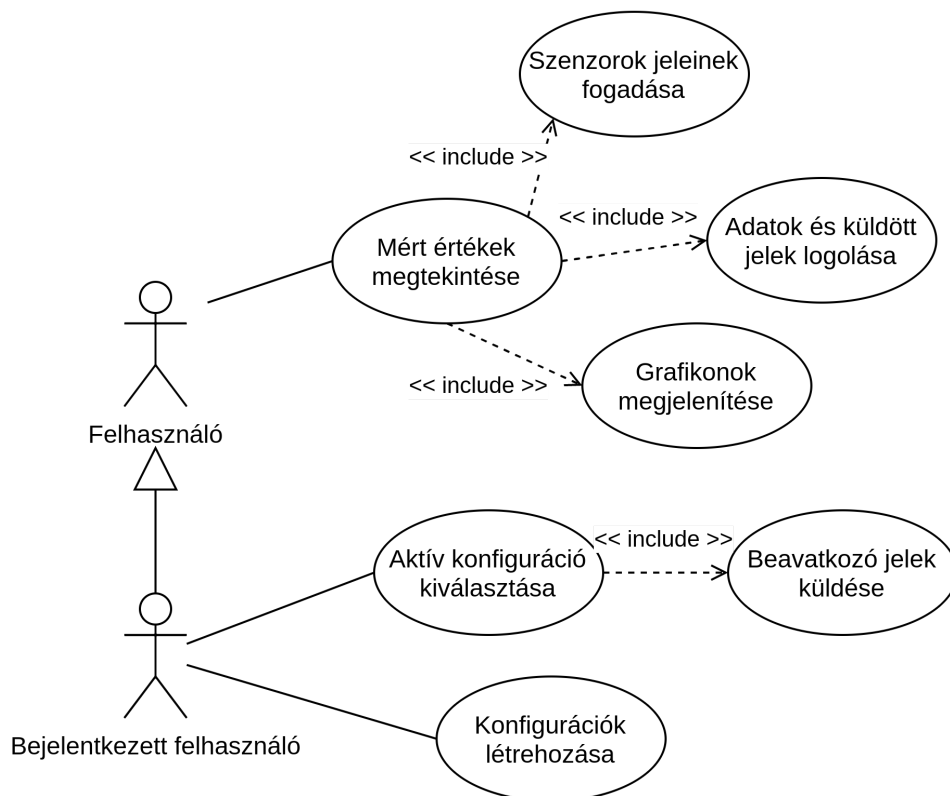
Growlight: napsütést imitáló lámpa, ami segítségével beltérben is növeszthetők növények (esetleg természetes fény mellé is kiegészítést nyújthat).

Növényigény-adatbázis: adatbázis, amelyben például elérhetők a különféle növénytípusokhoz tartozó ideális pH és EC értékek.

Konfiguráció: a visszajelzésekhez és beavatkozók működéséhez szükséges határértékek az adott növényhez kötve

Pumpa: a torony aljában lévő tápoldatot a torony tetejére nyomó pumpa.

Use-case diagram



Plaenty

Dokumentáció

VIAUMA06 Házi Feladat, 2021

Készítette:

Ádám Zsófia (SOSK6A)

Mondok Milán (URUGX5)

Tartalomjegyzék

Követelményspecifikáció	1
A fejlesztő csapat	1
Részletes specifikáció	2
Technikai paraméterek	2
Szótár	2
Use-case diagram	3
Tartalomjegyzék	5
A rendszer funkciói és célja, kontextusa	7
Feladatkiírás	7
Eredeti feladatkiírás	7
Megbeszélés utáni, finomított feladatkiírás	7
A rendszer funkciói	8
A rendszer kontextusa	8
Megvalósítás	9
Architektúra	9
Grafikus felhasználói felület	11
Megjelenés a kódban	11
GUI tervek	11
Az elkészült felhasználói felület	11
Spring Security, UAL, User data	17
Megjelenés a kódban	18
REST API	18
Megjelenés a kódban	18
ActiveConfigurationController	19
ConfigurationController	19
DashboardController	19
UserController	19
SensorController	19
Business Logic (BL)	19
Szabályozási folyamatok	19
Kérések kiszolgálása	20
Megjelenés a kódban	21
DAL (Data Access Layer, Adathozzáférési réteg)	22
Megjelenés a kódban	23
ConfigurationRepository	23
SensorDataRepository	23
SAL	24
Megvalósítás	25
Database	25
Megvalósítás	25
A rendszer telepítése	26

Összefoglalás	26
Továbbfejlesztési lehetőségek	26
Függelék	26
A rendszer megvalósításához használt technológiák	26
A REST API Yaml alapú OpenAPI3-as leírása	26

A rendszer funkciói és célja, kontextusa

Feladatkiírás

Eredeti feladatkiírás

Automatikus szobanövény/kiskert gondozó

A feladat során egy demo rendszert kell megvalósítani szoftveres beavatkozó és szenzor szimulációval. (Természetesen valós eszközökkel is megvalósítható.)

A feladat egy olyan rendszer létrehozása, amely életben tartja a növényeinket a lehetőségekhez mérten. Mindezt távolról megfigyelhető módon, hogy a nyaralás alatt is láthassuk az otthon uralkodó körülményeket, vagy a nyaralás végén visszanezessük az adatok alakulását.

Alapvetően az alábbi beavatkozók képzelhetők el:

- Egy vezérelt szivattyú az öntözéshez.
- Árnyékolás
- Szellőztetés

Példa szenzorok:

- Hőmérséklet
- Páratartalom
- Talajnedvesség
- Továbbá a döntésekhez még fel kellene használni az időjárás előrejelzés adatait.

A konfigurációs felületen célszerű minél nagyobb flexibilitást elérni, hogy a felhasználó be tudja konfigurálni a szabályokat, ami alapján a rendszer működik.

Esetleg még egy növény igény adatbázissal is megtámogatható a konfiguráció.

Megbeszélés utáni, finomított feladatkiírás

Plaenty

A projekt során egy hydroponic rendszer felügyelő alkalmazás fejlesztése a célunk. A rendszer szenzorai a pH és EC, illetve fényszenzor, míg az aktuátor egy lámpa (growlight) és egy pumpa, ezeket a házi feladat keretein belül szimuláljuk.

A rendszer használata egy android app-ként elkészülő frontenden keresztül lehetséges, a rendszer konfigurációiban módosítást végrehajtani csak regisztrált (és belépett) felhasználó tud. A belépett felhasználó hozhat létre új konfigurációkat,

illetve szerkezhetheti az általa létrehozottakat. Az aktív konfigurációt szintén csak belépett felhasználó állíthatja át.

A konfigurációk magukban foglalják az ideális értékeket, így a rendszer képes jelezni, ha a mért értékek ezeken kívül esnek, illetve képes ehhez igazítani az aktuátorokat. A rendszer egyben el van látva nagy számú előre megadott konfigurációval egy kiegészíthető növényigény adatbázis formájában.

A rendszer funkciói

Tipikusan elvárt funkciók:

- A legutolsó mért értékek és aktuátorok állapotának összefoglaló bemutatása (tehát a rendszer aktuális állapotának megjelenítése)
- Korábbi szenzor értékek grafikonon való megjelenítése
- Regisztrációs és bejelentkező felület
- Összes konfiguráció listászerű megjelenítése
 - Neveikkel és néhány egyéb információval
- Adott konfiguráció részletes megjelenítése
 - Konfiguráció szerkeszthetősége a létrehozója által
 - Új konfiguráció hozzáadása
 - Utóbbi alpontok csak megfelelő jogosultsággal
- Aktív konfiguráció megváltoztatása
 - Csak bejelentkezett felhasználó által
 - Az időzített szabályozó funkciók (mérések, aktuátor beállítás) az aktív konfigurációhoz igazodnak

A rendszer kontextusa

A rendszer egy szerverből és kliensből áll.

A szerver a szenzorok és aktuátorok kezelésére képes kell legyen (praktikusan GPIO pin-eken keresztül), de Java (11) futtatására is alkalmas kell legyen. A projekt Springet (Spring Boot 2.1.16) használ.

Tehát például egy Raspberry Pi vagy hasonló eszköz alkalmas erre a szerepre, hiszen ez egy könnyen és nem túl drágán beszerezhető eszköz, amely az otthon épített rendszerek részét képezheti.

A szerveroldali adatbázis egy fájl alapú HSQL adatbázis, amelyet a megfelelő spring csomag kezel, ezért magát a Java 11 projektet futtatni képes szerver esetén egyéb követelményt nem támaszt.

A kliens egy minimum android 8.0-t (API level 26) használó okostelefonon futtatható. A kliens a nem bejelentkezett felhasználó számára megjelenítőként, míg a

bejelentkezett felhasználó számára egyben a rendszer konfigurációs eszközeként is szolgál. Az alkalmazás Java 8-ban készült.

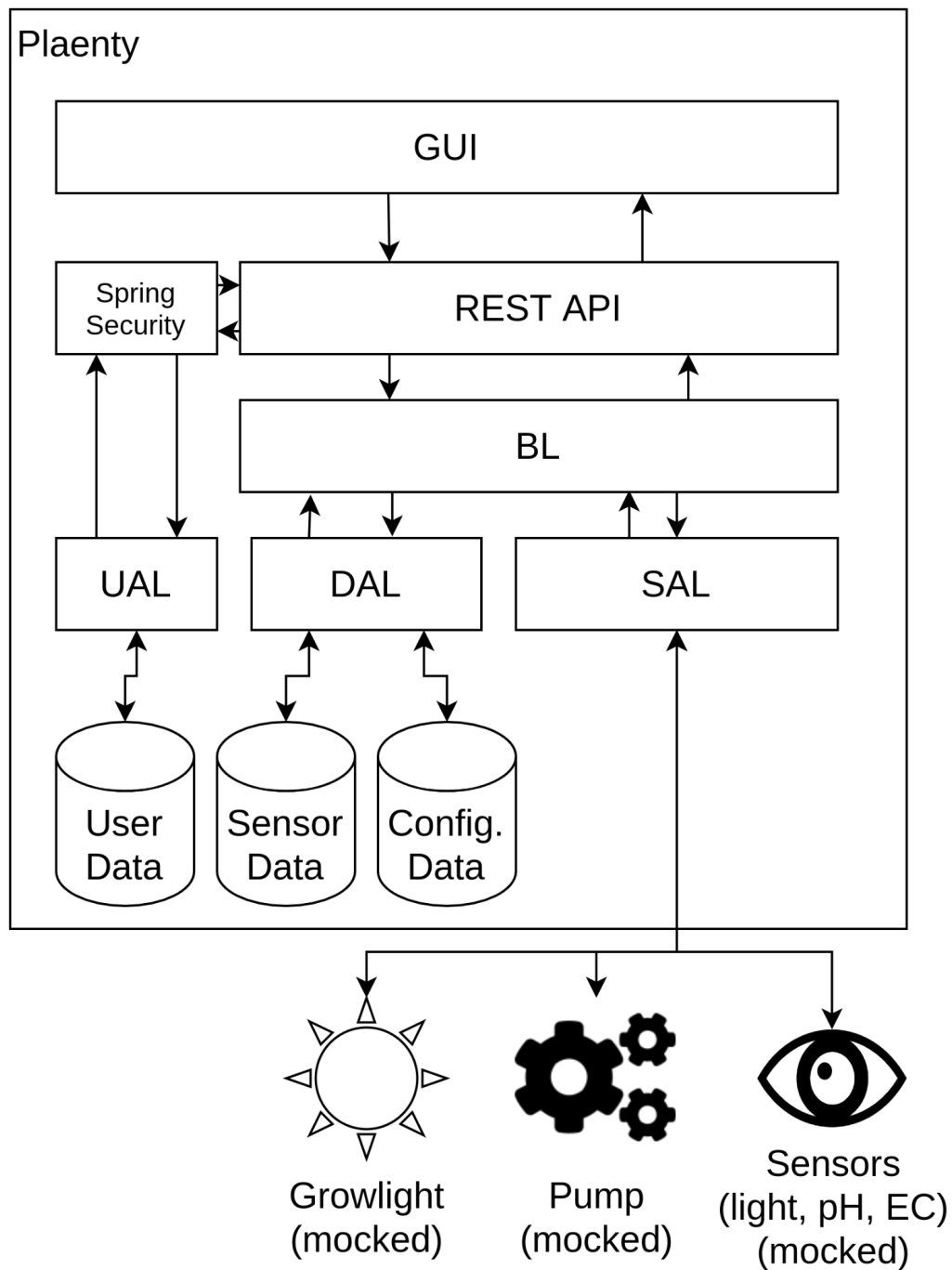
Megvalósítás

A projekt fejlesztésének és eredményeinek részletes dokumentációja alább található, beleértve az architektúra felépítését és a különböző rétegek megvalósítását.

Architektúra

Az alkalmazás egy kliens-szerver architektúrájú és egyben egy többretegű alkalmazás. A rétegek többsége a szerverben található, a GUI funkciói mind egy rétegbe sorolódnak. Az egyes rétegek egyértelműen különválnak, a rétegek a vertikális sorrendjüket nem szegik meg, a kérések a megfelelő "láncot" járják végig.

A rendszer **Plaenty** névre hallgat, hiszen egy jól gyümölcsöző hidropónikus rendszer sok (*plenty*) növényt hoz. A Plaenty rendszer nem foglalja magába a szenzorokat és aktuátorokat, ezek a rendszeren kívüli elemek, amelyeket a megfelelő rétegen keresztül kell illeszteni.



A Plaenty rendszer architektúrája

Az alkalmazás 6 különálló rétegre bontható:

- Grafikus felhasználói felület (GUI)
- REST API és Spring Security
- Üzleti logika (Business Logic, BL)
- Data Access Layer (DAL)
- Sensor (and Actuator) Access Layer (SAL)
- User Access Layer (UAL)
- Database (Configuration, Sensor és User táblák)

A rétegek kapcsolatát és a hozzájuk illesztett, már a rendszeren kívüli komponenseket a fenti ábra mutatja.

Grafikus felhasználói felület

Célja a funkciók elérhetővé tétele. A nem bejelentkezett felhasználók számára elsősorban egy megjelenítőként működik, míg a bejelentkezett felhasználók ezenfelül szerkeszthetnek, hozzáadhatnak és törölhetnek konfigurációkat, illetve beállíthatják az aktív konfigurációt.

Megjelenés a kódban

A projekt teljes kliens része képzi ezt a réteget, tehát a teljes android alkalmazás.

GUI tervek

Az alkalmazás tervezésekor a minél könnyebb használhatóságot tartottuk szem előtt, azaz hogy az alkalmazás az okostelefonokon megszokott felépítést kövesse, a megszokott gesztusokra a megszokott módon reagáljon, minél intuitívabb legyen a használata. A felhasználó felület tervezésekor az androidos fejlesztés során standardként tekintett Material design irányelveket követtük.

Az alkalmazás a következő képernyőkkel rendelkezik:

- Kezdőképernyő: 2 lap
 - Dashboard: a legfrissebb mért szenzor adatok és az aktív beállítások áttekintése
 - Konfigurációs lista: a választható konfigurációk felsorolása
- Konfiguráció szerkesztő: felület konfigurációk létrehozására és szerkesztésére (csak bejelentkezett felhasználók érhetik el)
- Bejelentkező képernyő: regisztráció és bejelentkezés
- Diagram képernyő: valamelyik választott szenzor által mért adatok megjelenítése egy vonaldiagramon

A képernyőkről készült vázlatos tervek:

Plaenty

EC:

1.23

dS/m

pH:

7.2

Light:

12300

lux

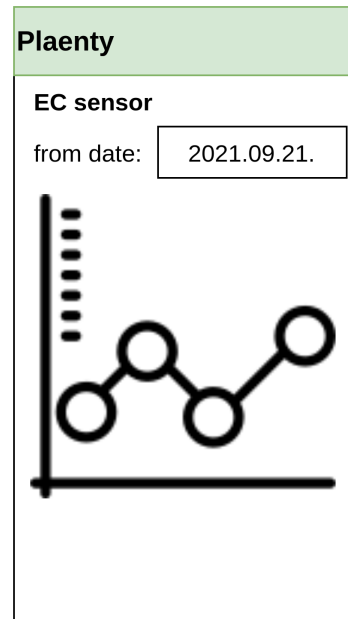
Active configuration:

Basil

User:

Alice

Dashboard



Graphs
(opens from dashboard
by tapping on data)

Plaenty

Username:

Password:

Sign Up

Login



Sign-up/Login

Plaenty

Config1

Config2

Details of Config



Set

Config3

Config4

Config4

Configurations
(scrollable)

Plaenty

Name:

EC range:

pH range:

Light Required:

Low

Pump on:

min

Pump off:

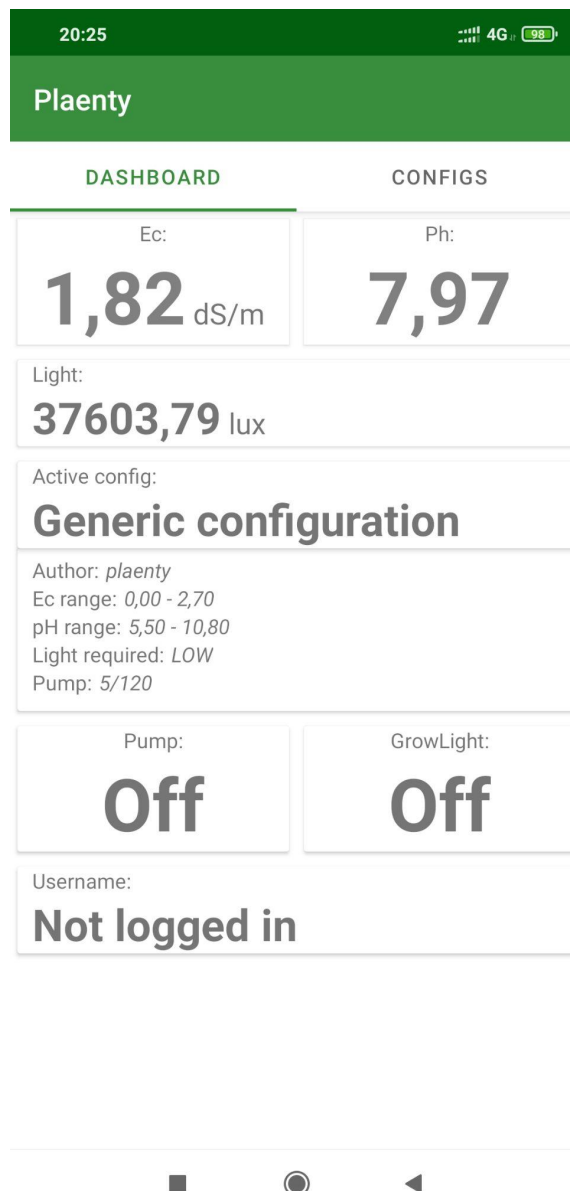
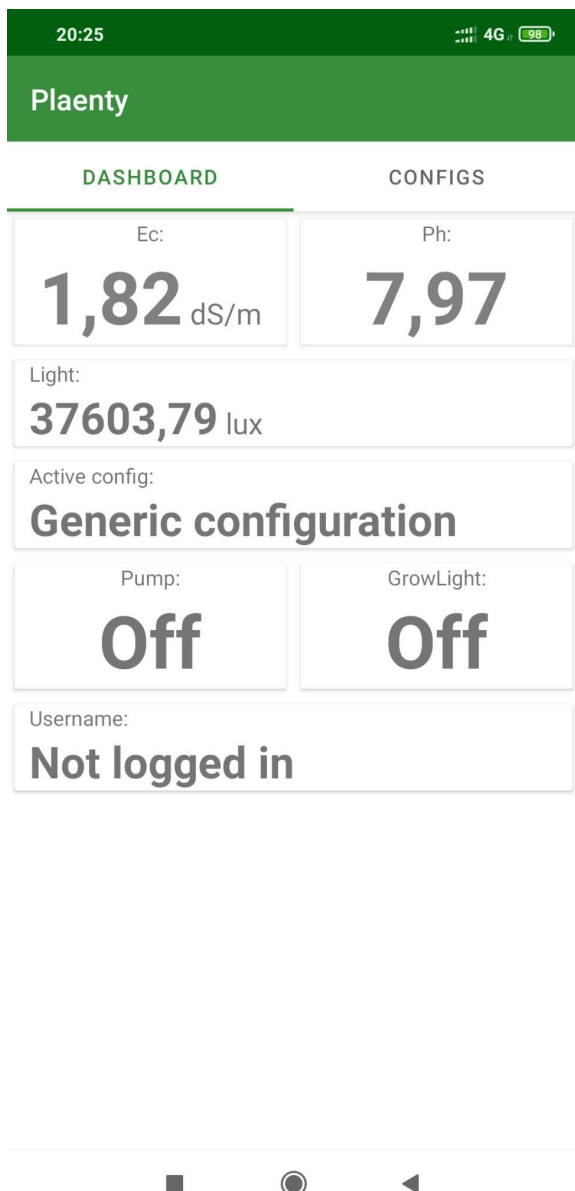
min

Save

Config. edit
(opens from config list)

Az elkészült felhasználói felület

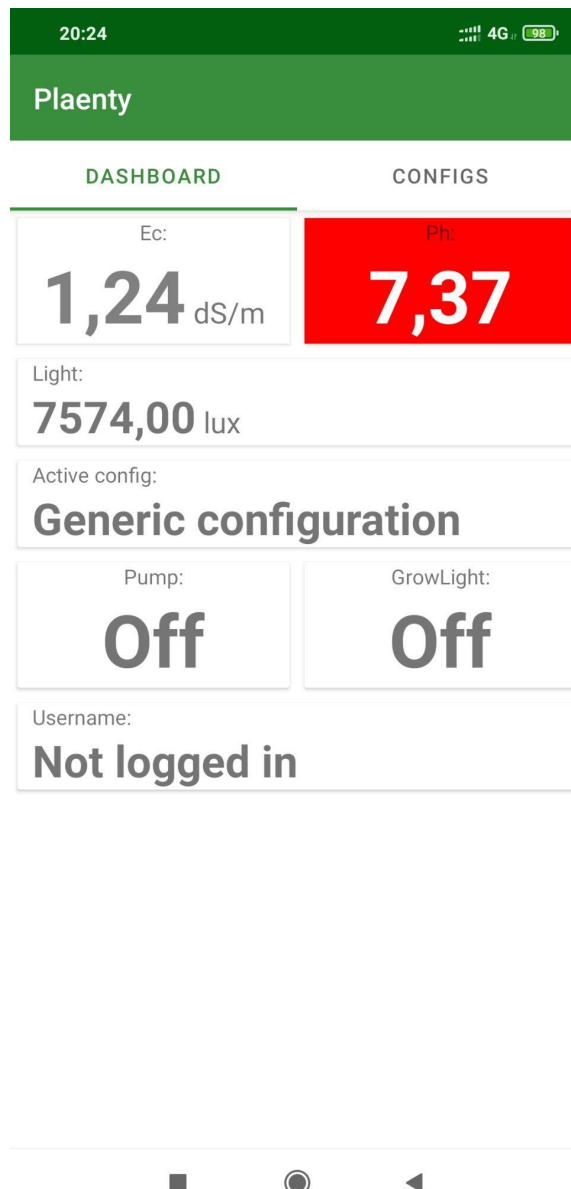
A felhasználót a *dashboard* képernyő fogadja. Ez egy áttekintést ad a hydroponic rendszer aktuális állapotáról. A képernyőn megtekinthetők a különböző szenzorok által mért legfrisebb értékek, az aktuális konfiguráció, illetve az aktuátorok állapota (on/off). Az értékek az okostelefonokon megszokott fentről lefele húzással frissíthetők.



A dashboard képernyő rejtett és megjelenített konfigurációs részletekkel

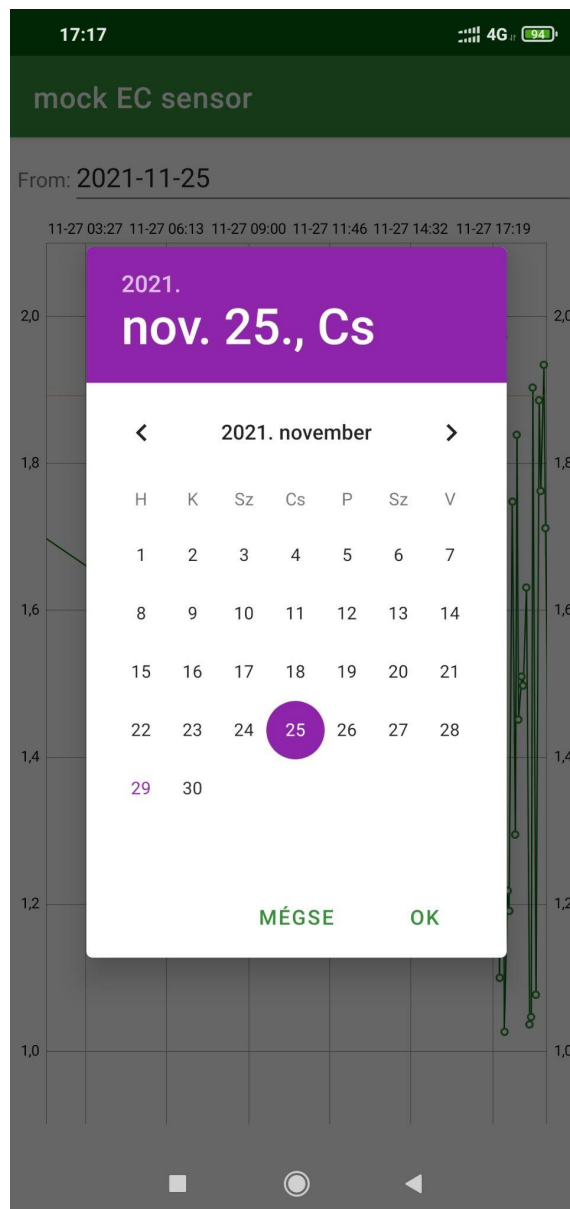
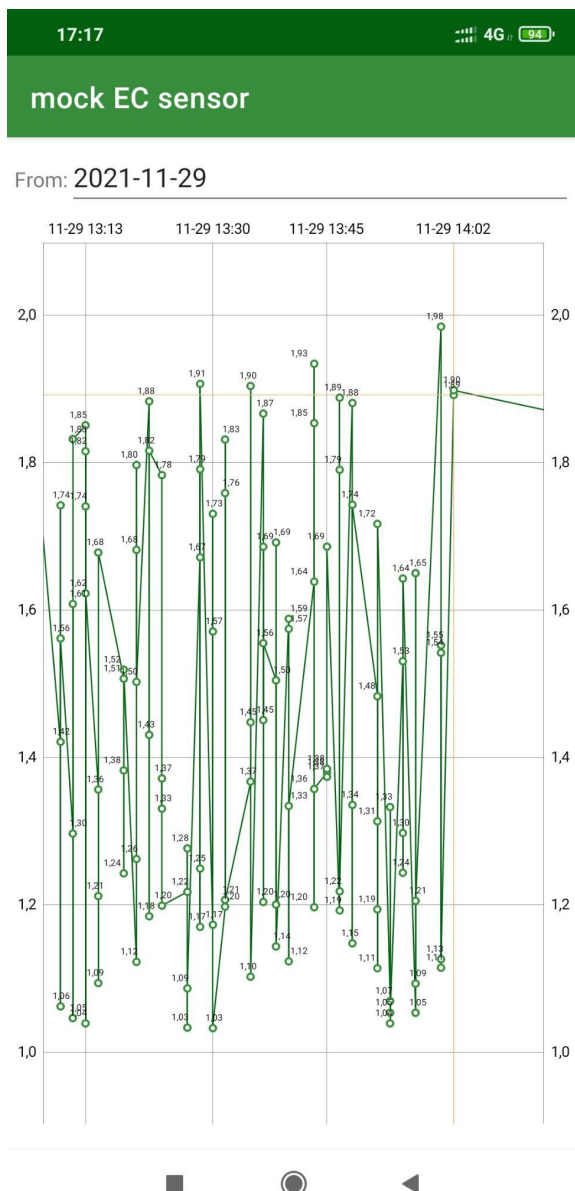
Az aktuális konfigurációra kattintva megtekinthetők annak részletei is a lenyíló panelen. (A konfiguráció szerzője, a konfiguráció aktív ec és ph intervalluma, az adott növény fényigénye, illetve a pumpa beállítás).

Ha valamelyik szenzor által mért érték az aktív konfigurációban megadott határértéken kívül esik, akkor a dashboardon piros színnel figyelmeztetés jelenik meg.



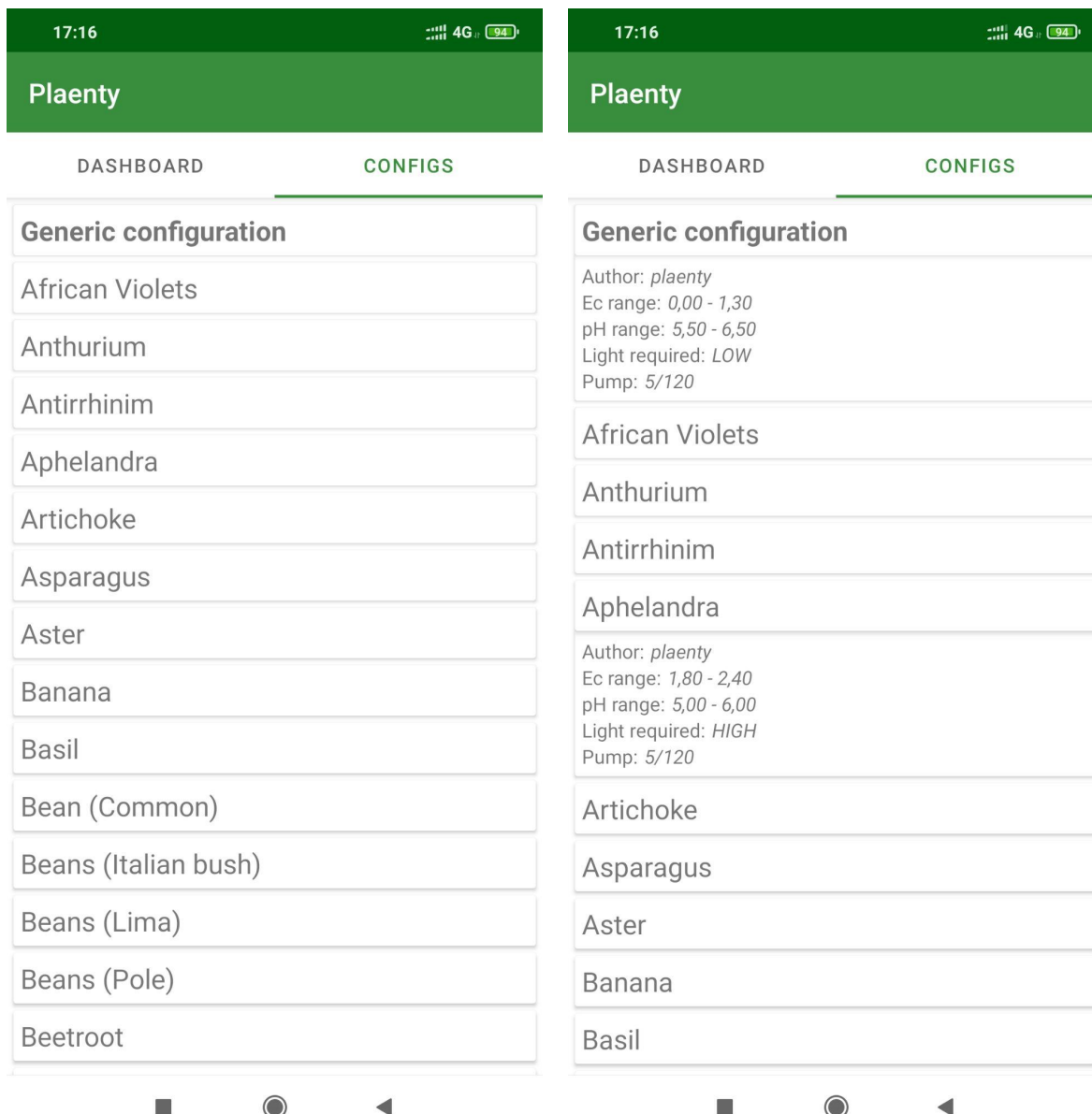
Figyelmeztetés határértéken kívül eső szenzorértékről

A szenzorokra kattintva megtekinthetők az adott szenzorok által mért adatokat ábrázoló grafikonok. Alapértelmezés szerint az aktuális nap értékei jelennek meg. A fenti dátum mezőre kattintva hosszabb intervallum is beállítható. A grafikon vízszintes tengelyén az idő, a függőleges tengelyén pedig a mért szenzor értékek vannak ábrázolva. Két ujjal a diagram mindkét tengely mentén nagyítható/kicsinyíthető illetve eltolható.



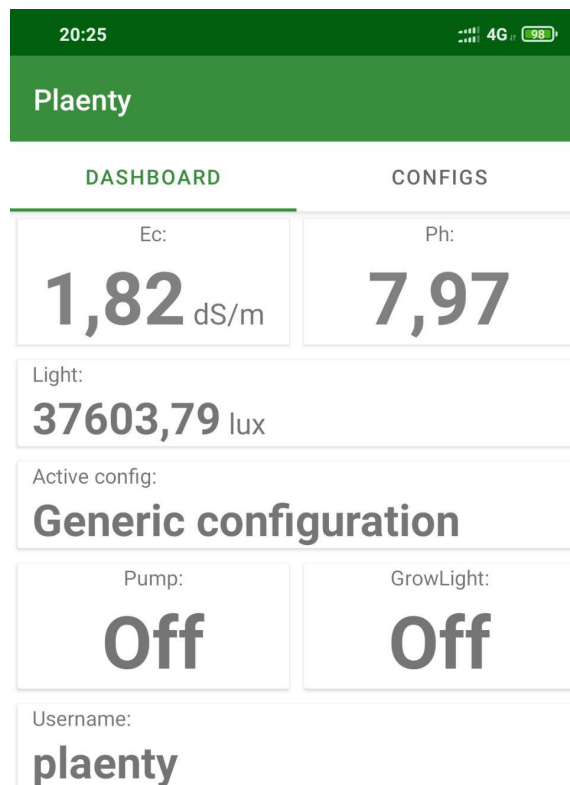
*Az EC szenzor által a megnyitás napján mért adatokat ábrázoló grafikon,
illetve a dátumválasztó képernyő*

A dashboard képernyőn jobbra lapozva a konfigurációs képernyő tekinthető meg. Ezen a képernyőn a rendszer összes konfigurációja látható egy listán megjelenítve. Az aktív konfiguráció félkövér névvel jelenik meg. A konfigurációkra kattintva azok részletei tekinthetők meg, (a konfiguráció szerzője, a konfiguráció aktív ec és ph intervalluma, az adott növény fényigénye, illetve a pumpa beállítás).



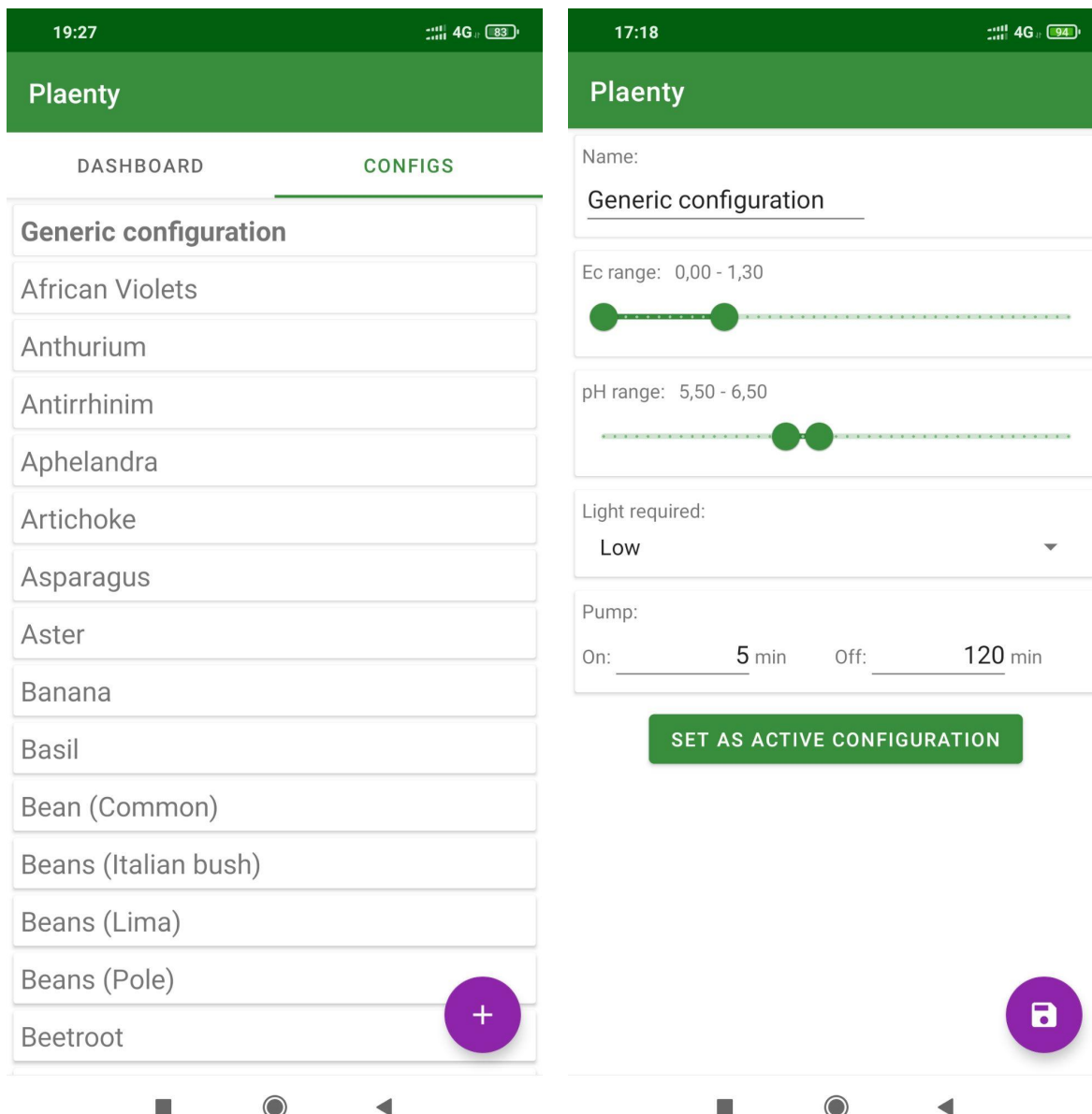
A konfigurációs képernyő (a jobb oldalon néhány konfiguráció részleteivel)

Bejelentkezni, illetve regisztrálni a dashboard képernyőn a username kártya megnyomásával lehet. Ilyenkor megnyílik a login activity, ahol bevihető a felhasználónév és a jelszó, majd lehetőség van bejelentkezésre és regisztrálásra. Új felhasználó regisztrációjakor a bejelentkezés is megtörténik automatikusan. Bejelentkezés után a dashboardra kerül vissza a felhasználó, ahol megjelenítésre kerül felhasználóneve is.



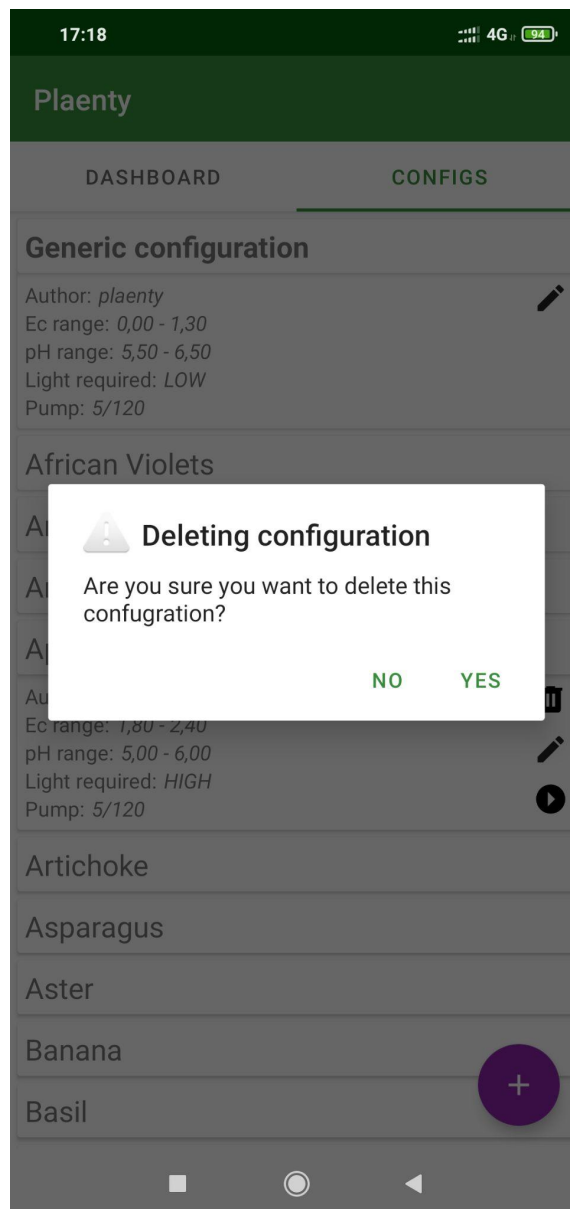
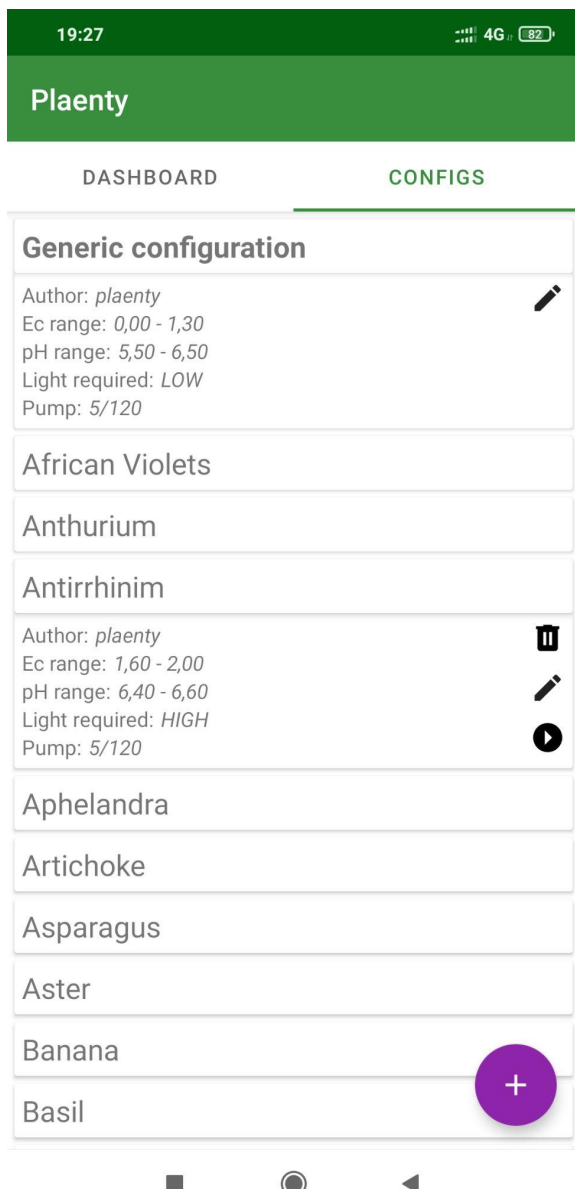
A bejelentkező képernyő, illetve a dashboard sikeres bejelentkezés után

Bejelentkezett felhasználók számára a konfigurációs képernyőn megjelenik a konfiguráció létrehozás gomb, ez a képernyő jobb alsó sarkában látható pluszjel. A gombra kattintva a konfiguráció-szerkesztő oldal nyílik meg, ahol megadhatók a létrehozandó konfiguráció adatai. Az EC és pH intervallumok csúszkákkal, a fényigény legördülő menüvel, a pumpa beállításai pedig szövegmezőkkel adhatók meg. A konfiguráció a jobb alsó sarokban látható gombbal menthető el.



A konfiguráció létrehozó gomb és a konfiguráció-szerkesztő oldal

A konfigurációs listán a bejelentkezett felhasználók ha lenyitják a valamelyik konfiguráció részleteit, akkor 3 gombot láthatnak megjeleníteni különböző feltételek fennállása esetén. A legalsó play gombbal aktív új aktív konfigurációt tudnak választani. A felette lévő toll gombbal a saját konfigurációikat tudják szerkeszteni. A legfelső kuka gombra kattintva saját konfigurációkat tudnak törölni. Az aktív konfiguráció nem törölhető.



A 3 konfiguráció manipulációs gomb (törlés, szerkesztés, kiválasztás aktívként), illetve a törléskor megjelenő jóváhagyó ablak

Spring Security, UAL, User data

Ezen rétegek megvalósítása elsősorban csak a Spring Security megfelelő konfigurálását jelenti. A bejelentkezés és a felhasználók kezelése a business logic rétegről le van választva, hiszen nem megfelelő jogosultsággal ebben nem is szabad engednünk semmiféle kérés érkezését. A felhasználói adatok szintén el vannak választva a többi adattól.

A kérések érkezésekor a szükséges jogosultsági szintet szintén a Spring Security ellenőrzi. Az aktuális felhasználó nevére a Rest API-nak bizonyos kérések esetén

szüksége lehet, ezt a Spring Security segítségével tudja lekérni, amely a kéréshez küldött megfelelő token alapján tudja ezt visszaadni.

Megjelenés a kódban

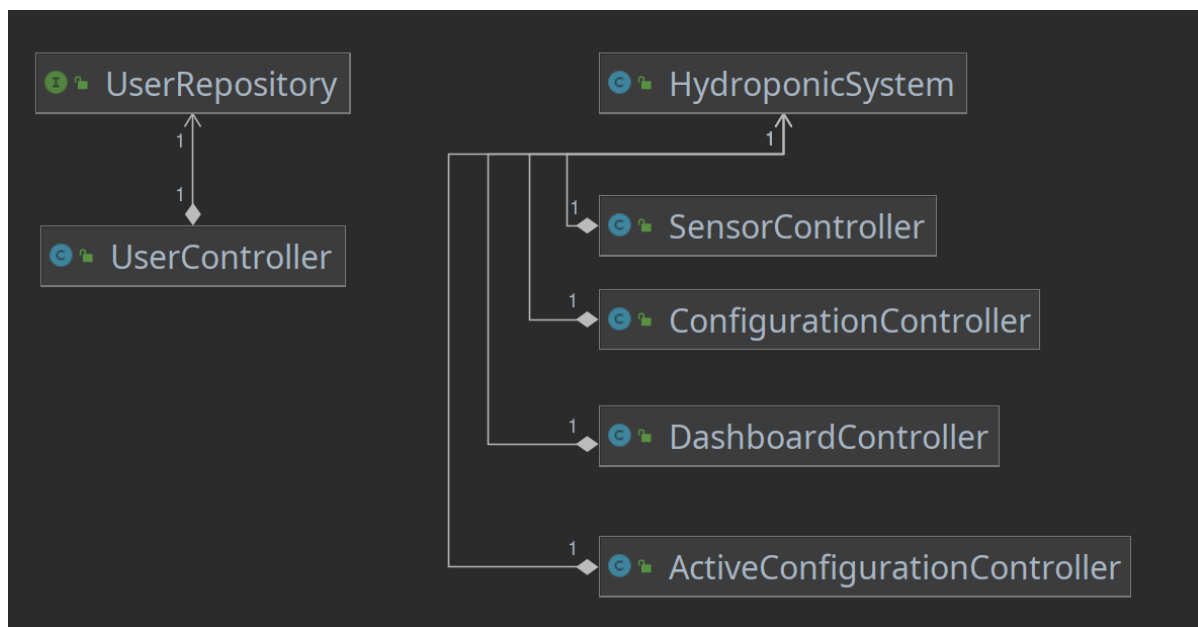
A security packageben találhatóak a Spring Security működéséhez szükséges beállítások és minimális logika, de az autentikációs és autorizációs feladatokat az ebből generálódó funkciók végzik el. Az UAL feladatait pedig a repository package UserRepository interface-n megadott műveletek alapján generálódik.

REST API

A REST API a kliens és a szerver közötti kommunikációt biztosító réteg a szerverben. Ez szolgáltatja a kliens számára az összes megjelenítendő adatot, illetve a kliensben tett módosítások is ezen keresztül haladnak a megfelelő rétegek felé.

A REST API és a kliens fejlesztésének megkönnyítéséhez egy, az OpenAPI 3.0 szabványnak megfelelő YAML leírást készítettünk, amely a REST API-nak adható pontos kérések és az adatok formátumának leírását tartalmazza. Ezt a leírást a projekten felül ennek a dokumentációnak a függeléke is tartalmazza. Az adatok JSON formátumban adódnak át.

Megjelenés a kódban



Controllerek Class diagramja: a Controllerek kapcsolódása a Business Logic magját képező HydroponicSystem-hez, illetve a UserController (autentikáció) erről való leválasztása

A REST API a megvalósított szerver része, a hozzá tartozó osztályok a `plaenty.restapi` package-ben található controller osztályok. Ezek az üzleti

logika segítségével szolgálják ki a GUI felől érkező kéréseket. A következőekben ismertetjük ezeket.

ActiveConfigurationController

Az aktív konfiguráció lekérésére és módosítására ad lehetőséget a GUI-nak, mely kéréseket az üzleti logika felé irányít tovább.

ConfigurationController

Képes olyan kérések kiszolgálására, mint a konfigurációk kilistázása, vagy id alapján való lekérése. Id alapján lehet létező konfigurációt módosítani, illetve újat hozzáadni (megfelelő jogosultsággal).

A jogosultságok kezelése ennél a controllernél nem merül ki annyiban, hogy az adott felhasználó be van-e jelentkezve, hanem azt is megvizsgálja, hogy jogosult-e egy adott konfigurációt szerkeszteni az adott felhasználó, illetve gondoskodik róla, hogy a megfelelő felhasználó neve kerüljön szerzőként az újonnan hozzáadott konfigurációba.

DashboardController

Feladata a rendszer általános állapotának közlése a klienssel, ha az kéri ezt. Ez magában foglalja az aktuátorok állapotát (ki vagy bekapcsolt pumpa és lámpa), a szenzorok legutóbb mért értékeit. és az aktív konfigurációt.

UserController

Itt kerül megvalósításra a /users/sign-up endpoint, mellyel a felhasználó regisztrálhat. A felhasználók jelszava hashelve kerül tárolásra. (A /login endpoint-ot viszont a Spring Security generálja.)

SensorController

A szenzorokkal kapcsolatos kéréseket képes feldolgozni, mint:

- Az összes szenzor és ezek tulajdonságainak kilistázása
- Adott szenzor adott időpontig visszamenő méréseinek listázása (speciális esetben adott szenzor összes eddigi méréseinek listázása.

Business Logic (BL)

Az üzleti logika részben az időzített szabályozó folyamatokat látja el, részben pedig a megfelelő adatok lekérésével összekapcsolja az alatta és felette lévő rétegeket. Magának a rendszernek a modellje is itt jelenik meg megfelelő szenzor és aktuátor objektumok formájában.

Szabályozási folyamatok

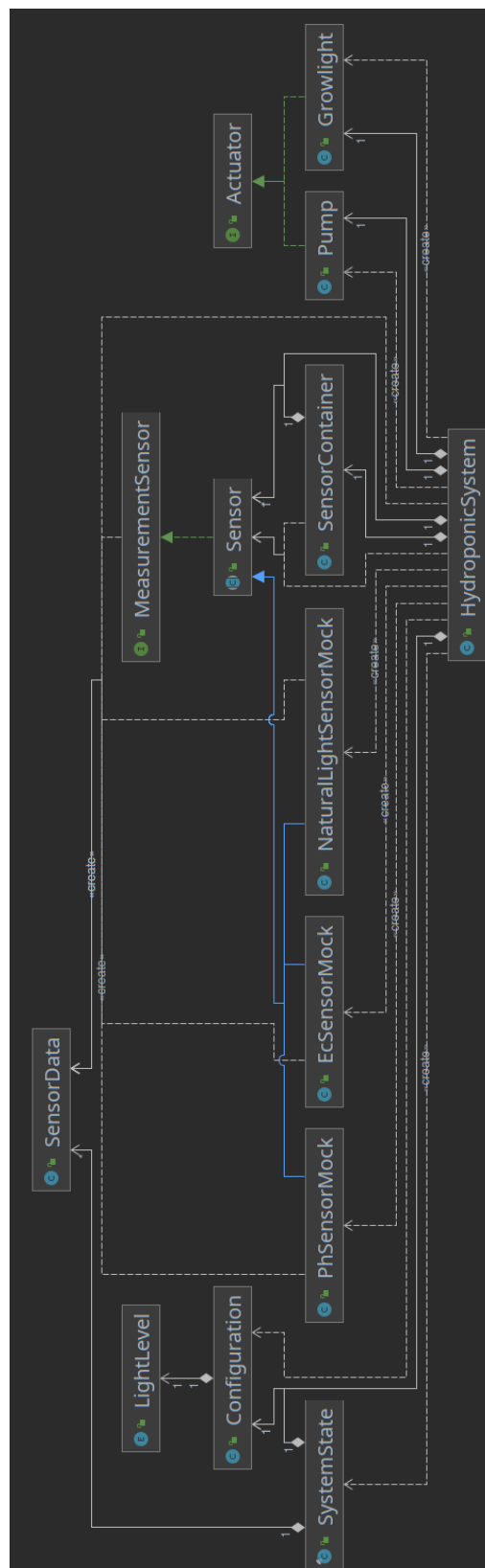
A szabályozási folyamatok a rendszer egyik legfontosabb részét képezik. Ezek látják el a mérési és aktuátor kezelési folyamatokat mint időzített szolgáltatások. Két különálló folyamatot különböztethetünk meg:

- Az egyik a fix időintervallumonként méréseket indító szenzor- és lámpakezelő folyamat, mely elvégzi a friss mérést, elmenti az adatokat, majd ezen adatok alapján beállítja a lámpát.
- A másik a konfigurációtól függően időzített pumpa kezelés, mely a konfigurációnak megfelelő sűrűséggel kapcsolja ki és be a pumpát

Kérések kiszolgálása

A különböző adatok lekérésére szolgáló kérésekhez az adatok összegyűjtése és megfelelő formában való visszaadása is itt történik - például a `SystemState` objektum összeállításával, amely a rendszer jelenlegi állapotát (szenzorok utolsó mérései, aktuátorok állapota, aktív konfiguráció) jelképezi, vagy a szenzor lista lekérésével, amely a `SensorContainer` osztály tárol, mint a modell része.

Megjelenés a kódban



Az üzleti logikát és modellt képző `model` package class diagramja. A képen más rétegek osztályai is fel vannak tüntetve, hogy a megfelelő kapcsolatokat ábrázolhassuk (lásd: a leírásban)

Mint a fenti osztálydiagramon látható, az üzleti logika központi része maga a Hydroponic System, amely az időzített szabályozó funkciók helye, illetve a rendszer modelljét képező objektumok létrehozója és menedzsere.

Magát az üzleti logikát a HydroponicSystem osztály, illetve ezt segítve a SensorContainer és SystemState valósítják meg. A Configuration és a SensorData elsősorban a DAL alá tartozó entitások, míg a MeasurementSensor és Actuator, illetve az ezt megvalósító osztályok és az io package egyéb osztályai a SAL réteghez tartoznak.

A szenzorokat létrehozza és átadja a SensorContainer-nek, illetve létrehozza, tárolja és irányítja az aktuátorokat. A SystemState segédosztály segítségével képes a rendszer állapotát is visszaadni.

Egy SystemState objektum a következőket tartalmazza pontosan:

- Aktív konfiguráció
- Szenzorok állapota - Egy Map a szenzorokról és ezek utolsó mérési eredményeiről
 - EC szenzor
 - Ph szenzor
 - Fényszenzor
- A pumpa állapota *(ki vagy bekapcsolt)*
- A lámpa állapota *(ki vagy bekapcsolt)*

DAL (Data Access Layer, Adathozzáférési réteg)

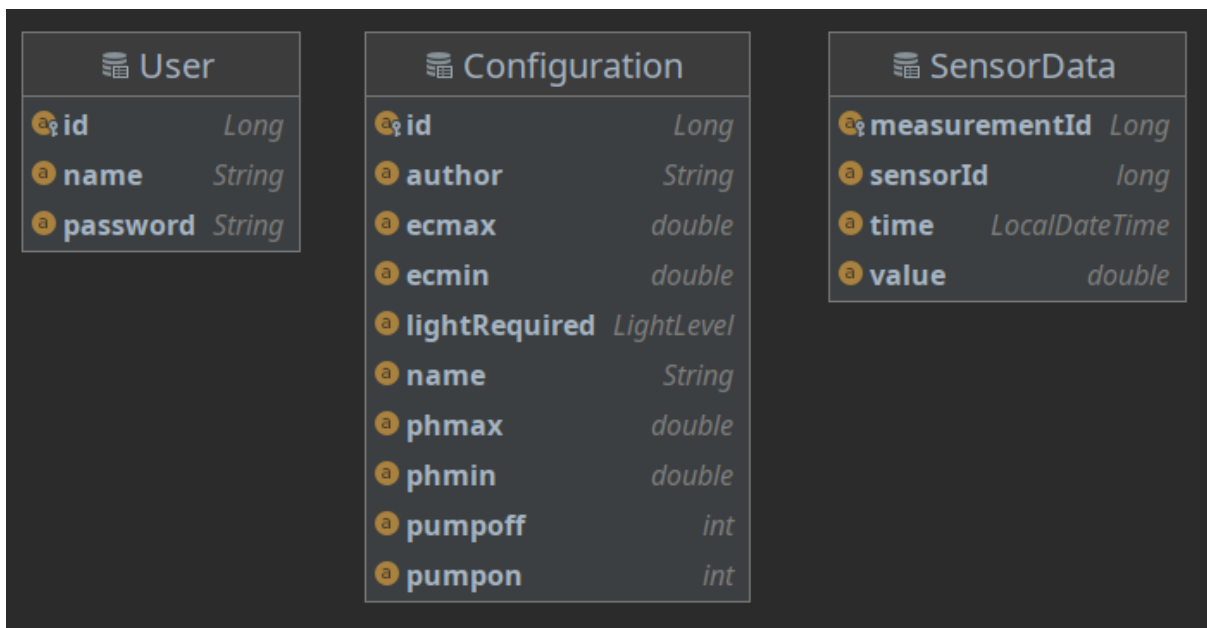
Az adathozzáférési réteg feladata a kommunikáció és a megfelelő lekérések előállítása az adatbázis felé.

Feladatai:

- Új entitások létrehozása az adatbázisban
 - Szenzor adat (timestamp, szenzor id és a mért érték)
 - Konfigurációk létrehozása
- A már létező konfiguráció entitások frissítése

Az adatbázis létrehozása nem feladata, mivel az adatbázis fájl a rendszerrel együtt érkezik, benne a növényigény adatbázissal

Megjelenés a kódban



IntelliJ által generált ER diagram, rajta az összes repository-val. A User entitás és az ehhez tartozó repository és tábla a User Data és az UAL részei, de ezen a képen is feltüntettük a teljesség kedvéért.

A képen látható generált ER diagram ábrázolja a HSQL adatbázis tábláit. Az entitások típusai a fent leírtaknak megfelelően a User, Configuration és SensorData.

A réteghez tartozó osztályok alapvetően a repository ConfigurationRepository és SensorDataRepository osztályai, illetve az Entitás osztályok: Configuration, SensorData.

ConfigurationRepository

Ez az osztály felelős a Konfigurációkat tároló tábla kezeléséért, úgy mint:

- Új konfiguráció mentése
- Létező konfiguráció frissítése
- Id alapján konfiguráció lekérése
- Összes konfiguráció listában való lekérése

SensorDataRepository

Ez az osztály felelős a szenzor adatokat tároló tábla kezeléséért, úgy mint:

- Új szenzor adat mentése
- Adott szenzor (id alapján) összes mért adatának lekérése

- Adott szenzor (id alapján) által adott időpont és jelen pillanat között mért értékek lekérése
- Adott szenzor (id alapján) által utoljára mért adat lekérése

SAL

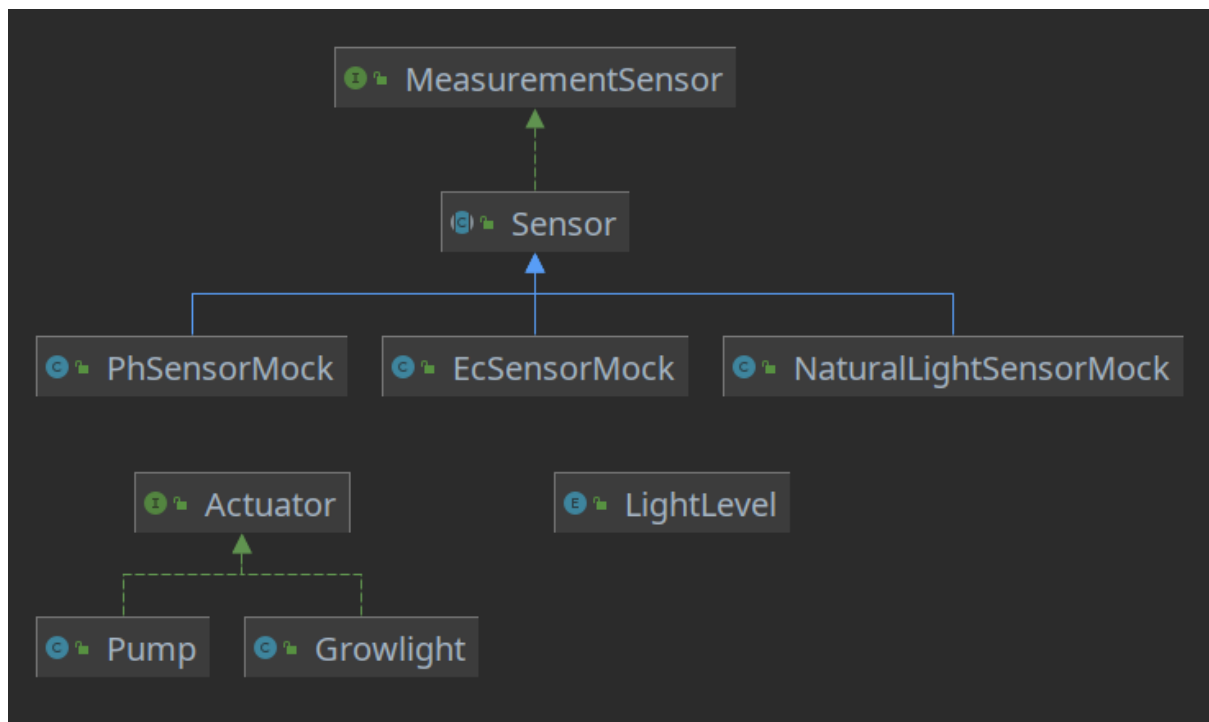
A Sensor (and Actuator) Access Layer feladata interfészt biztosítani az EC, pH és fényszenzorok, illetve lehetséges lámpák és pumpák számára.

Maga a hardver specifikus implementáció nem tartozik a rendszerhez, jelenleg ez csak mock-okat tartalmaz.

- A beavatkozótól azt várjuk, hogy le lehessen kérni az állapotukat (ki- vagy bekapcsolt), illetve hogy ki és bekapcsolhatóak legyenek.
- A szenzoroktól azt várjuk, hogy képesek legyenek a megfelelő metódus hívásakor mérést kezdeni és eredményt visszaadni, illetve legyen nevük és dimenziójuk (pl. lux fénynél, dS/m EC esetén, de más is lehet), melyet kiírhatunk megjelenítéskor

A mock osztályok lecseréléséhez csak egy azokhoz hasonló megfelelő leszármazottat kell elkészíteni, de a rendszer szenzorai és aktuátorai akár át is rendezhetők, ehhez azonban már módosítani kellhez az üzleti logika működését is, hiszen ez ez esetben nem egyértelmű.

Megvalósítás



A SAL-hoz tartozó osztályok class diagramja (zöld nyíl: megvalósítás, kék nyíl: leszármazás)

A `LightLevel` enum a fényszensor értékeinek intervallumokra osztását végzi (milyen lux dimenziójú érték magas, normál vagy alacsony), ennek értékeit más dimenziót visszaadó szensor esetén megváltoztathatjuk.

A `MeasurementSensor` interface a mérések végrehajthatóságát követeli meg, míg az absztrakt `Sensor` osztály a szensor nevét és dimenzióját.

Az `Actuator` interfész a fent ismertetett ki/bekapcsoló funkciókat és az állapot lekérhetőségét követeli meg.

Az ábrán lévő többi osztály pedig a jelenlegi mock megvalósítások, melyeket nem demo rendszer esetén rendes megvalósításra cserélhetünk majd.

Database

Maga az adatbázis három táblában tárolja a fent már ismertetett adatokat és a DAL-on keresztül elérhető.

Megvalósítás

Az adatbázis egy HSQL adatbázis, fájl alapon. Azért választottuk ezt, mert egy egyszerű és lightweight megoldás, melynek felkonfigurálása és telepítése minimális.

A rendszer telepítése

A rendszer telepítéséhez egyszerűen csak az android alkalmazás apk-ból való telepítésére van szükség, illetve a szerverhez pedig a megfelelő maven build után a Java projekt elindítására. Pontosabb követelmények a dokumentáció elején a *Kontextus* pont alatt találhatóak.

A telepítés során a kliensben szükséges a szerver URL-jének beállítása. Ez a NetworkManager osztály SERVICE_URL fieldjének szerkesztésével tehető meg. A backend telepítéskor a plaentymb.script.log, plaentymb.script.properties és plaentymb.script.script fájlok a bináris mellé másolandók. (Ezek tartalmazzák a fájlalapú adatbázis működéséhez szükséges adatokat.)

A mock-ok helyett valós szenzorok telepítéséhez szükség van a megfelelő interfészek (MeasurementSensor, Actuator) megvalósítására és a HydroponicSystem osztályban ezek lecserélésére, mivel ez erősen rendszer és hardver specifikus.

Összefoglalás

Házi feladatunk célja egy hidroponikus növényfelügyelő rendszer fejlesztése volt. Munkánk során megterveztük, implementáltuk és dokumentáltuk a rendszert.

A rendszer pH, EC és fény szenzorok kezelésére képes, beavatkozni pedig pumpa és growlight segítségével tud. A kifejlesztett rendszer egy Java nyelven írt Spring alapú backendből, egy Java nyelven írt android kliensből, az adatbázisokból és a szenzorokat kezelő rétegből áll. A felhasználó a kliens segítségével képes a rendszer aktuális állapotának nyomon követésére (szenzoradatok, aktuátorok állapota). Bejelentkezett felhasználók képesek konfigurációkat szerkeszteni, törölni, új konfigurációkat létrehozni és kiválasztani az aktuális konfigurációt. A rendszer képes figyelmeztetések megjelenítésére, ha valamelyik szenzor által mért érték a megadott határértéken kívül esik. Az alkalmazásban megjeleníthetők historikus szenzoradatok grafikonon ábrázolva.

A házi feladat során végzett tervezési és implementációs munka eredményeként egy olyan alkalmazás jött létre, mely megkönnyíti a hidroponikus növénytermesztő rendszerek felügyeletét és teljesíti a kitűzött követelményeket. A rendszerhez készült dokumentáció megkönnyíti a rendszer konfigurálását, használatát, illetve továbbfejlesztését.

Továbbfejlesztési lehetőségek

A rendszerből elsősorban a hardveres megvalósítás hiányzik, így természetesen a közvetlen következő lépés ez lenne.

Ezenfelül érdemes lehet olyan irányba rugalmasan növelhetővé tenni a rendszert, hogy több, kisebb rendszert fogjon össze - vagyis több olyan vezérlési egységet, amelyben van például egy lámpa, egy pumpa és a szokásos szenzorok (például több, egy szerver által kezelt hidropónikus torony).

A rendszer tervezésekor az ilyen irányú fejlesztések megkönnyítésének figyelembevételével alakítottuk az architektúrát (például SAL réteg bevezetése).

Függelék

A rendszer megvalósításához használt technológiák

[Spring Boot](#)

[Spring Security](#)

[Android Java](#)

[Retrofit library](#)

[HSQLDB](#)

A REST API Yaml alapú OpenAPI3-as leírása

```
openapi: 3.0.0
```

```
info:
```

```
  description: "This is the REST API of the Plaenty project."
```

```
  version: "1.0.0"
```

```
  title: "Plaenty API"
```

```
  license:
```

```
    name: "Apache 2.0"
```

```
    url: "http://www.apache.org/licenses/LICENSE-2.0.html"
```

```
components:
```

```
  schemas:
```

```
    configuration:
```

```
      properties:
```

```
        id:
```

```
          type: integer
```

```
        name:
```

```
          type: string
```

```
        author:
```

```
          type: string
```

```
        ecmin:
```

```
          type: string
```

```
          format: double
```

```
        ecmax:
```

```
          type: string
```

```
          format: double
```

```
        phmin:
```

```
          type: string
```

```
        format: double
    phmax:
        type: string
        format: double
    lightrequired:
        type: string
        enum: [low, normal, high]
    pumpon:
        type: integer # minutes
    pumpoff:
        type: integer # minutes
sensor:
    properties:
        id:
            type: integer
        name:
            type: string
        dimension:
            type: string
sensordata:
    properties:
        time:
            type: string
            format: date-time
        value:
            type: string
            format: double
sensorlist:
    type: array
    items:
        type: object
        allOf:
            - $ref: '#/components/schemas/sensor'
sensordatalist:
    type: array
    items:
        type: object
        allOf:
            - $ref: '#/components/schemas/sensordata'
systemstate:
    properties:
```

```

    activeconfiguration:
      $ref: '#/components/schemas/configuration'
    sensorstate:
      type: array
      items:
        type: object
        properties:
          sensor:
            $ref: '#/components/schemas/sensor'
          data:
            $ref: '#/components/schemas/sensordata'
    pumpState:
      type: boolean
    lightState:
      type: boolean
  userinfo:
    properties:
      name:
        type: string
      password:
        type: string
  paths:
    /dashboard:
      get:
        tags:
          - "dashboard"
        summary: "Sends the present state of the system"
        description: "Returns if the system is functional and the
value of sensors, actuators, active configuration, etc."
        operationId: "sendDashboardData"
        responses:
          "200":
            description: "The present state of the system"
            content:
              application/json:
                schema:
                  $ref: '#/components/schemas/systemstate'
          "404":
            description: "Error: Cannot get system status"
    /sensor/data/{sensorid}:
      get:

```



```

tags:
- "sensor"
summary: "Returns the historical data measured by a sensor"
description: "Returns the measurements taken on the given
sensor from the given date and time"
operationId: "sendSensorData"
parameters:
- in: "path"
name: "sensorid"
required: true
schema:
type: integer
description: "ID of the sensor we should return data
about"
- in: "query"
name: "from"
schema:
type: string
format: date-time
description: "The historical data will be between this
point in time and the present time. Should be a point in time which
happened already. Default value is returning all of the
measurements."
responses:
"200":
description: "The requested historical data on a single
sensor"
content:
application/json:
schema:
$ref: '#/components/schemas/sensordatalist'
"404":
description: "Sensor not found"
/sensor/list/:
get:
tags:
- "sensor"
summary: "Returns the sensors of the system (name and id)"
operationId: "sendSensorList"
responses:
"200":

```

```

        description: "The list of sensors (names with ids)"
        content:
            application/json:
                schema:
                    $ref: '#/components/schemas/sensorlist'
/active-configuration:
    get:
        tags:
            - "configuration"
        summary: "Returns the active configuration"
        description: "Return all the configuration information of the
active configuration"
        operationId: "activeConfiguration"
        responses:
            "200":
                description: "The active configuration of the system"
                content:
                    application/json:
                        schema:
                            $ref: '#/components/schemas/configuration'
            "404":
                description: "Error: No active configuration in system"
/active-configuration/{id}:
    put:
        tags:
            - "configuration"
        summary: "Updates the current configuration based on the given
id"
        parameters:
            - in: "path"
              name: "id"
              required: true
              schema:
                  type: integer
                  description: "configuration id"
        responses:
            "200":
                description: "The (updated) active configuration of the
system"
                content:
                    application/json:

```

```

        schema:
          $ref: '#/components/schemas/configuration'
      "403":
        description: "Forbidden"
      "404":
        description: "Configuration not found"
      "500":
        description: "Internal error: server could not set active
configuration"
  /configuration/{id}:
    get:
      tags:
        - "configuration"
      summary: "Returns the details of the configuration with the
given id"
      description: "Return all the configuration information of the
configuration with the given id"
      parameters:
        - in: "path"
          name: "id"
          required: true
          schema:
            type: integer
            description: "configuration id"
      responses:
        "200":
          description: "The requested configuration of the system"
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/configuration'
        "404":
          description: "Configuration not found"
    put:
      tags:
        - "configuration"
      summary: "Update existing configurations"
      parameters:
        - in: "path"
          name: "id"
          required: true

```

```

    schema:
      type: integer
      description: "configuration id"
  responses:
    "200":
      description: "Ok"
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/configuration'
    "404":
      description: "Configuration not found"
    "500":
      description: "Internal error: Could not update
configuration"
  /configuration:
    post:
      tags:
        - "configuration"
      summary: "Add a new configuration"
      description: "Adds a new configuration and returns the newly
added configuration in the response"
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/configuration"
      responses:
        "200":
          description: "Ok"
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/configuration'
        "500":
          description: "Internal error: Could not add configuration"
  /configuration/list:
    get:
      tags:
        - "configuration"

```

```
summary: "Returns a list of all the configurations"
description: "Returns a list of all the configurations"
responses:
  "200":
    description: "The active configuration of the system"
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/configuration'
```

/users/sign-up:

```
post:
  tags:
    - "user"
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/userinfo"
  responses:
    "200":
      description: "Ok"
    "400":
      description: "Username exists"
```

/users/login:

```
post:
  tags:
    - "user"
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/userinfo"
  responses:
    "200":
      description: "Ok"
    "401":
      description: "Unauthorized (bad credentials)"
```