VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

AUTONOMNÍ SYSTÉM PRO PODPORU PĚSTOVÁNÍ ROSTLIN

AUTONOMOUS SMART PLANT SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **RADIM BURÁŇ**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. ZDENĚK VAŠÍČEK, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

147738

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Buráň Radim**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Autonomní systém pro podporu pěstování rostlin**
Kategorie:	Vestavěné systémy
Akademický rok:	2022/23

Zadání:

1. Seznamte se s platformou ESP32, existující softwarovou podporou a možnostmi využití cloudových technologií pro potřeby obousměrné komunikace se senzorickými uzly. Seznamte se s problematikou tzv. chytrého pěstování rostlin, tj. veličinami používanými pro monitorování růstu a akcemi pro podporu růstu (automatická závlaha, modulace osvětlení, apod.).
2. Navrhněte vestavěný systém pro podporu růstu rostlin, tzv. chytrý květináč, který bude monitorovat vstupy z různých senzorů (např. vlhkost, vodivost, impedance, pH, intenzita osvětlení v různých spektrech, apod.) a na základě uživatelem zvoleného scénáře ovládat aktuátory tak, aby se co nejvíce zefektivnil proces růstu. Vestavěný systém navrhněte tak, aby jeho činnost byla autonomní a současně bylo možné ze strany uživatele komunikovat skrze cloud.
3. Navržený systém implementujte formou prototypu využívajícího např. dostupné vývojové platformy.
4. Vyhodnoťte funkčnost implementace a parametry. Diskutujte možná rozšíření.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodu 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Vašíček Zdeněk, doc. Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Cílem této práce bylo navrhnout a implementovat chytrý květináč pro podporu růstu rostlin, který bude možné ovládat skrze cloud. Pro tyto potřeby bylo nutno seznámit se s platformu ESP32, cloudovou komunikací, veličinami ovlivňující rostliny a dostupnými senzory použitelnými k monitorování těchto vlastností. Z důvodu ovládání skrze internet je část této práce věnována i implementaci webové aplikace, která klade důraz na jednoduchost a obsahuje přednastavení pro různé rostliny, které mohou být vytvářeny i komunitou.

Abstract

The aim of this thesis was to design and implement a smart pot for plant growth support that can be controlled through the cloud. For this purpose, it was necessary to acquaint with the ESP32 platform, cloud communication, factors affecting the plants and available sensors usable to monitor these properties. Because of the control through the internet, part of this thesis is also dedicated to the implementation of a web application that emphasizes simplicity and includes presets for different plants that can be also created by the community.

Klíčová slova

chytrý květináč, ESP32, MQTT, REST, InfluxDB, webová aplikace, klient-server, vlnová délka světla, PAR, PPFD

Keywords

smart pot, ESP32, MQTT, REST, InfluxDB, web application, client-server, light wavelength, PAR, PPFD

Citace

BURÁŇ, Radim. *Autonomní systém pro podporu pěstování rostlin.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Zdeněk Vašíček, Ph.D.

Autonomní systém pro podporu pěstování rostlin

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Zdeňka Vašíčka, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Radim Buráň
9. května 2023

Poděkování

Rád bych poděkoval vedoucímu této práce – panu doc. Ing. Zdeňku Vašíčkovi, Ph.D. Dále bych poděkoval Tomášovi Mikulcovi za poskytnutí 3D tiskárny pro vytištění konstrukce osvětlení květináče.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Již nějakou dobu je na vzestupu internet věcí (Internet of Things – IoT)[1]. Jedná se o síť fyzických věcí („things“), které komunikují spolu navzájem či s jinými systémy pomocí internetu. Většinou obsahují řadu senzorů, kterými získávají data z reálného světa a následně je využívájí pro různé akce – ať už se jedná o automatizaci či jejich reprezentaci v uživatelském prostředí. Do této kategorie spadají různé věci – od teploměru v akváriu, přes chytré náramky či domácí spotřebiče, až po auta. Tento velice rychlý rozvoj je způsobem mimo jiné dostupností levných a energeticky málo náročných senzorů, jednoduchostí připojením do cloudu a obecně rozvojem cloudových technologií[2].

Zároveň jsem chtěl prozkoumat chytré pěstování rostlin v domácím prostředí. Existující řešení většinou spoléhají pouze na automatické zalívání, či i světlo podporující růst. Ovšem ne každé rostlině vyhovuje stejné světlo a navíc spektrum nelze ovlivnit. Různé vlnové délky mají na rostliny jiný vliv [15]. Proto by mohla být konkurenční výhoda mít možnost měnit intenzitu světla a i v jaké vlnové délce. Tímto by bylo možné ovlivnit jakým způsobem rostlina roste – jestli spíše do šířky či výšky nebo její kvetení. Žádné dostupné řešení neumí ani analyzovat spektrum světla, přitom již existují senzory právě pro tento účel a jejich cena není astronomická

Cílem této práce je navrhnout autonomní květináč, který bude analyzovat data ze senzorů a dle přednastavení uživatele používat aktuátory pro zefektivnění růstu. A tento systém následně implementovat pomocí dostupných vývojových platforem (v tomto případě ESP32).

[1] https://explodingtopics.com/blog/iot-stats
[2] https://www.oracle.com/internet-of-things/what-is-iot/

Kapitola 2

Existující řešení

Mnohé řešení na crowfundingových platformách, jako třeba jedno hezky graficky zpracované[1] jsou ve skutečnosti podvody, kdy po kampani firma vyhlásí bankrot a lidi nevidí produkt, ani své peníze.

Existují i květináče využívající hydroponii. Hydroponie je způsob pěstování rostlin mimo půdu, jen v roztoku se živinami. Toto řešení přináší výhody v tom, že lze přímo řídit dávkování živin, je zde menší riziko plísní a škůdců a zaručuje vyšší efektivitu růstu[2]. Nevýhodou je ovšem vyšší pořizovací cena[3]. Komerčně prodávaná řešení většinou mají tyto živiny v kapsli spolu s rostlinou, které cenově opět nevychází úplně nejlevněji.

Nepovedlo se mi tedy najít řešení, které by obsahovalo všechno zmíněné zároveň. Tedy knihovnu přednastavení pro rostliny, osvětlení, které není stálé, ale lze nastavit v určitých spektrech, automatické zavlažování pumpou a dostupnost skrze cloud s dostupnými informacemi v grafech.

[1] https://www.indiegogo.com/projects/lua-the-smart-planter-with-feelings
[2] https://www.higarden.cz/blog/hyroponie-co-obnasi-a-jak-zacit/
[3] https://www.conserve-energy-future.com/advantages-disadvantages-hydroponics.php

Kapitola 3

Platforma ESP32

Série ESP32 zahrnuje čipy zaměřené na malou spotřebu, nízkou cenu a bezdrátovou konektivitu vyráběné firmou Espressif sídlící v Číně. Ve světě má další 4 pobočky. Jedna z poboček je dokonce v naší republice, v Brně. Informace jsem čerpal z oficálních webových stránek [21].

Pro domácí kutilství a prototypování existují tzv. „Devkity“, které již implementují mimo ESP32 čipu i UART pro USB komunikaci, vývody pinů a většinou alespoň jedno tlačítko a LED diodu. Některé dokonce obsahují i zabudovaný držák na baterie[1] nebo OLED displej[2]

Na rozdíl od konkurenčního Arduina má již zabudovanou WiFi a Bluetooth, není tedy nutné řešit rozšiřující desky. Ale pracuje na 3.3V místo 5V, je tedy nutné dát si pozor (některé senzory potřebují 5V).

Jedná se o nástupce ESP8266, hlavní rozdíly jsou v tabulce:

Funkce	ESP8266	ESP32
CPU	1 jádro Xtensa LX6 @ 160MHz	1-2 jádra Xtensa LX6 @ 240MHz
WiFI	HT20 (130 Mbit)	HT40 (300 Mbit)
Bluetooth	ne	4.2 + BLE
GPIO	17	34
SPI/I^2C/I^2S/UART	2/1/2/2	4/2/2/2
PWM	8 kanálů	16 kanálů
ADC	10 bit	12 bit
Dotykové piny	0	10

Ačkoliv může mít ESP32 dvě jádra, tak jedno je používáno pro WiFi komunikaci, aby neblokovala aplikační jádro. Uživatel je ale schopen spouštět kód na libovolném jádře.

Existují i novější typy ESP32, které obsahují RISC-V instrukční sady, novější WiFi 6 a Bluetooth 5 Low Energy či standard IEEE 802.15.4 pro komunikace na blízko mezi zařízení:

- `ESP32-S2` – LX7 @ 240MHz, jen WiFi
- `ESP32-S3` – 2x LX7 @ 240MHz, WiFi + Bluetooth 5, podpora vektorových instrukcí

[1] https://wiki.geekworm.com/WEMOS_ESP32_Board_with_18650_Battery_Holder

[2] https://www.lilygo.cc/products/lilygo%C2%AE-ttgo-t-display-1-14-inch-lcd-esp32-control-board

- `ESP32-C2` – RISC-V @ 120MHz, WiFi + BL 5
- `ESP32-C3` – RISC-V @ 160MHz, WiFi + BL 5
- `ESP32-C6` – RISC-V @ 160MHz, WiFi 6 + BL 5 + IEEE 802.15.4
- `ESP32-H2` – RISC-V @ 96MHz, BL5 + IEEE 802.15.4

3.1 Komunikace se senzory a aktuátory

Platforma ESP32 má bohaté možnosti propojení s jinými prvky díky podpoře různých sběrnic a jejich počtu. V podkapitolách níže uvedu jejich příklady. Pro specifikaci ohledně počtu sběrnic na ESP32 jsem čerpal z oficiálních stránek espressif [21], pro popis I^{2}C a SPI sběrnic z látky předmětu IMP (Mikroprocesorové a vestavěné systémy)[3].

3.1.1 Analogová komunikace

Funguje pomocí převodu mezi analogovým signálem na digitální nebo obráceně. Jinými slovy mapování napětí na číselné hodnoty. ESP32 obsahuje dva 8-bit digitálně analogové převodníky (DAC) a dva 12-bit analogově digitálně převodníky (ADC).

ESP32 samozřejmě umí generovat i pulzně šírkovou modulaci (PWM) na až 16-ti kanálech zároveň.

3.1.2 SPI

Je zkratka pro Serial Peripheral Interface. Jedná se o sériovou plně duplexní komunikaci po sběrnici, tedy přenos probíhá oběma směry. ESP32 obsahuje čtyři SPI sběrnice, ovšem dvě z nich se používají pro komunikaci s interní flash pamětí, proto se je nedoporučuje používat. Používá tyto vodiče:

- MOSI – Master Out Slave In – data z masteru do slave
- MISO – Master In Slave Out – data ze slave do masteru
- SPSCK – SPI Serial Clock – hodinový signál generovaný master zařízením
- SS – Slave Select – pro každý slave zvlášť, vybírá a aktivuje jednotku pro přenos dat v log 0

3.1.3 I^{2}C

Znamená Inter-Integrated Circuit. Stejně jako SPI se jedná o synchronní sériovou komunikace mezi master a slave komponentami, ovšem I^{2}C je pouze half-duplex, tedy data mohou jít jedním směrem v jeden moment. Původní specifikace byla navržena firmou Philips Semiconductors v roce 1982, později proběhlo pár aktualizací. ESP32 obsahuje dvě I^{2}C sběrnice. Používá tyto vodiče:

- SDA – Serial Data – přenos dat, start/stop podmínka, potvrzení příjmu
- SCL – Serial Clock – synchronizační signál

[3] https://www.fit.vut.cz/study/course/IMP/.cs

K jedné sběrnici lze připojit až 127 jiných zařízení, což je velká výhoda oproti SPI, kde každý slave potřebuje vlastní SS propojení. Je toho docíleno pomocí adresování jednotlivých slave komponent. Ovšem zde nastává problém, že adresy musí být unikátní. Pokud by byly shodné, tak je nutné použít multiplexor, další I^2C sběrnici nebo změnit adresu na slave (některé devkity se senzory tuto funkci podporují, většinou stačí spájet určité plošky na desce). Tento limit je dán tím, že adresování je 8bitové, kde 1 bit je pro výběr zápisu či čtení a zbylých 7 pro adresu.

3.2 Softwarová podpora

Oficiální vývojovou platformou je ESP-IDF, která používá jazyk C/C++. Existují ovšem i alternativy, například Arduino IDE, které je vhodné pro rychlé prototypování či menší projekty (takzvané `sketch`e), ale hlavní výhoda je v přenositelnosti mezi různými platformami mikrokontrolerů a větší množství knihoven. Jako vývojové prostředí lze použít i `PlatformIO`, které podporuje jak Arduino, tak ESP-IDF a necelých 200 ESP32 desek. Pokud se chceme vyvarovat C/C++, tak je možnost zvolit `MicroPython` – což je Python určený pro mikrokontrolery.

Samotné ESP32 implementuje FreeRTOS (Real-Time Operating System) v upravené podobě pod názvem `ESP-IDF FreeRTOS`, primárně z důvodu podpory více CPU jader[4]. Jedná se o jednoduchý systém vyvinutý pro mikrokontrolery, který mimo jiné umožňuje za běhu vytvářet úkoly (`tasks`), které s využitím čekání a přerušení umožňují multitasking.

Mimo programování pro mikrokontroler lze psát kód i v assembleru pro ULP (Ultra Low Power) koprocesor, který je aktivní i v hlubokém spánku, kdy nejsou aktivní jádra procesoru. ULP slouží pro převody pomocí ADC, měření pomocí teplotního čidla či externích I2C senzorů a obsluhu RTC GPIO pinů. Je taky možné pomocí něj probudit ESP32 ze spánku[5].

[4] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html

[5] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ulp.html

Kapitola 4

Cloudová komunikace s ESP32

Pro komunikaci je možné využít více protokolů, ovšem každý má své výhody a nevýhody a proto se hodí na rozdílné věci.

Taktéž je možné využít P2P (Peer-to-peer) komunikaci mezi jednotlivými mikrokontrolery v místech, kde není WiFi signál. Pokud bych uvažoval téma mé práce, tak třeba velký skleník, kde je mnoho květináčů, ale není zde 100% pokrytí WiFi signálem. Pak by byla data posílána mezi jednotlivými senzorickými uzly a odeslána tím, který by měl dostupný WiFi signál z routeru. Jedná se o **ESP-NOW** a **ESP-WIFI-MESH**.

4.1 ESP-NOW

Redukuje 5 vrstev OSI pouze na 1, navíc nepoužívá hlavičky jako typická komunikace po internetu, je tedy méně náročný. Odezva je maximálně 150ms. Nevýhodou je nutnost znát MAC adresu. Je zde taktéž limit 20 spárovaných zařízení, 6 pokud je zapnuté šifrování. Jedná se o limit na zařízení, lze tedy vytvořit síť s více zařízeními, kdy nejsou všechny spojeny se všemi. Je možné využít broadcast, který žádný limit nemá, ale posílá úplně všem, i nespárovaným, tedy nutno vyřešit zabezpečení. Limit velikosti přenášených dat je 250 bajtů[1].

4.2 ESP-WIFI-MESH

Na rozdíl od ESP-NOW má výhodu, že podporuje automatické formování a uzdravování meshe při odebrání/přidání nového zařízení. Rychlost přenosu je až 10Mbps a dosah 200 metrů. Limit velikosti přenášených dat je 1456 bajtů. Nevýhodou je nutno použít vývojový framework **ESP-MDF** místo **ESP-IDF** (ten je již obsažen)[2].

4.3 REST

Jedná se o akronym – `REpresentative State Transfer`, který popisuje architekturu komunikace přes protokol (většinou HTTP). Umožňuje práci s daty na serveru dle dotazů klienta. Rozhraní (API) splňující podmínky RESTu se označuje jako „RESTful“ [13]. Definuje tzv. `endpointy` (koncové body) na které se klient dotazuje a je mu odeslána odpověd.

[1] https://docs.espressif.com/projects/espressif-esp-faq/en/latest/application-solution/esp-now.html

[2] https://www.espressif.com/en/products/sdks/esp-wifi-mesh/overview

Používá 4 základní operace – **CRUD** a jejich příklady:

- Create (POST) – `HTTP POST /devices`
- Read (GET) – `HTTP GET /devices/dev1`
- Update (PUT) – `HTTP PUT /devices/dev1`
- Delete (DELETE) – `HTTP DELETE /devices/dev1`

Díky této architektuře je možné vytvořit aplikaci formou klient – server, kdy tyto 2 celky mohou být na odlišných strojích a komunikovat spolu pomocí REST API. Klient nemusí být ani frontend webové aplikace, ale třeba přímo nativní aplikace na telefonu či jiné plaformě, ale stále může používat stejné API serveru pomocí HTTP dotazů. To znamená i jednodušší vývoj a škálovatelnost do budoucna, jen je nutno zachovat kompatibilitu API mezi klienty a serverem. Což lze docílit po velké změně API pomocí úpravy cesty k endpointu, např. přidat „ver1“ do URI a tak zachovat více verzí API zároveň.

4.4 MQTT

Dlouze `Message Queuing Telemetry Transport`, ovšem tento název se příliš nepoužívá, neboť protokol zpracovává více věcí, než je z názvu patrné. Jedná se o protokol vhodný pro IoT zařízení, neboť díky své jednoduchosti není příliš náročný na výpočetní výkon a objem přenesených dat.

Architektura MQTT potřebuje mezičlen, tzv. „MQTT Broker“. Ten obstarává klienty tak, že zpracovává přijaté zprávy tříděním dle témat, tzv „topiců“ a ty následně odesílá všem klientům, kteří mají o ně zájem. Témata tvoří strukturu podobnou adresářové oddělenou lomítky, např. `domov/pokoj1/teplota`.

Komunikace probíhá pomocí pub/sub modelu. Odesílání zpráv je za pomoci „publish“ na dané téma, broker následně přepošle danou zprávu všem klientům, kteří požádali o „subscribe“ na zmíněné téma. Aby mohl klient odesílat zprávy, tak není nutné, aby byl přihlášen k danému tématu.

Zprávy mohou mít 3 úrovně QoS (Quality of Service), které určují garanci doručení:

- `QoS 0` – „Maximálně jednou“ – zpráva se může ztratit
- `QoS 1` – „Minimálně jednou“ – zpráva může být doručena vícekrát
- `QoS 2` – „Pouze jednou“ – zpráva je doručena přesně jednou

Je důležité je správně použít v implementaci, protože využitím nesprávné se mohou ztratit důležitá data nebo naopak bude zbytečná zátěž co se týče datových přenosů v řádu desítek procent – rozdíl mezi QoS 0 a 1 je zhruba 40% a mezi QoS 1 a 2 zhruba 50%[3].

4.4.1 MQTT vs HTTP přenos

Pokud porovnáme tyto 2 typy přenosů, tak vyjde najevo, že již při 2 zprávách je výhodnější použít MQTT než zasílat data přes HTTP. Tabulka převzata od HiveMQ[4].

[3] https://medium.com/@dheeptuck/how-to-optimize-data-usage-over-mqtt-792abebd2cd1
[4] https://www.hivemq.com/blog/mqtt-vs-http-protocols-in-iot-iiot/

Akce	MQTT	HTTP
Připojení	5572	2261
Odpojení	376 (opt)	0
Odeslání zprávy	388	3285
Součet pro 1 zprávu	6336	5546
Součet pro 10 zpráv	9829	55460
Součet pro 100 zpráv	44748	546000

Důvodem velkého rozdílu je to, že primární overhead u MQTT je připojení na broker, ale u HTTP je overhead u každého requestu.

Kapitola 5

Chytré pěstování rostlin

V dnešní době lze zefektivnit pěstování rostlin díky využití IoT zařízení, které monitorují prostředí v reálném čase díky různým senzorům a následně provádí akce pro optimalizaci – ať už se jedná o zrychlení růstu či o úsporu zdrojů. Z pravidla se jedná o světelné, teplotní, vlhkostní a živiny upravující akce.

5.1 Půdní vlastnosti

Půda ovlivňuje mnoho faktorů pro růst rostliny a i ostatní veličiny. Je pro rostlinu zdrojem živin. Pro následující teorii jsem čerpal z [8] a [23]. Z druhého zmíněného zdroje jsou i informace ohledně půdní pH.

5.1.1 Obsah vody

Pro měření obsahu vody v půdě jsou nejběžnější 2 způsoby: Gravimetrický obsah vody v půdě (GWC – **Gravimetric Water Content**) měří hmotnost vody v půdě. Tato hodnota se získává měřením rozdílu váhy vysušené půdy

$$\text{GWC } (\%) = \frac{\text{hmotnost mokré půdy (g)} - \text{hmotnost vysušené půdy (g)}}{\text{hmotnost vysušené půdy (g)}} * 100$$

Jelikož se půda suší v troubě při teplotě okolo 105°C, tak je tato metoda považována za destruktivní, jelikož je půda nepoužitelná na další analýzu.

Dalším způsobem je objemový obsah vody (VWC – **Volumetric Water Content**), který se zabývá poměrem objemu vody a půdy. Na tomto způsobu pracují i senzory půdní vlhkosti. Rovnice je následující:

$$\text{VWC } (\%) = \frac{\text{objem vody (ml)}}{\text{objem půdy (ml)}} * 100$$

5.1.2 Kyselost a zásaditost

Koncentrace vodíkového kationtu (H^+) určuje kyselost, z tohoto vychází logaritmická stupnice pH s rozsahem od 0 po 14 s tím, že interval mezi 6,5 až 7,4 definuje neutrální pH. Půdy mají většinou pH mezi 4 a 8.

Rostliny preferují rozmezí podobné neutrálnímu, ale mírně ke kyselosti, tedy mezi 6 a 7 pH, ale třeba borůvky mají nejraději pH 4[1]. Důvod proč je vhodné udržovat ideální pH je ten, že rostliny jsou pak schopny nejlépe získávat živiny z půdy.

5.2 Světelné vlastnosti

V této kapitole jsem čerpal primárně z [15].

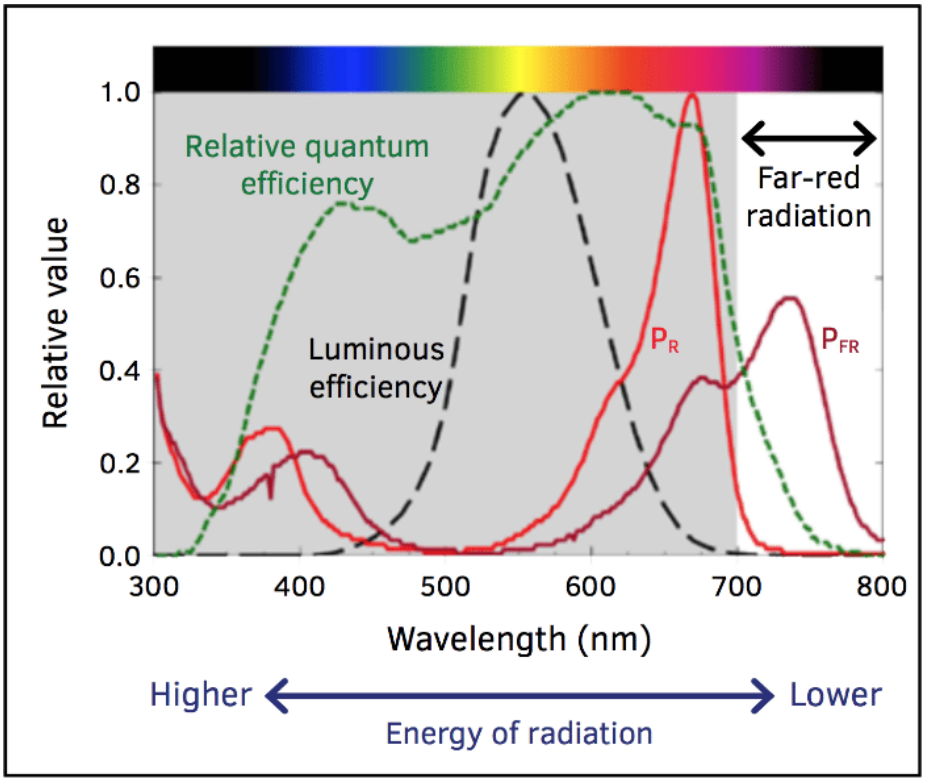
Světlo je pro rostliny jeden z nejdůležitějších prvků, který ovlivňuje jak rostlina roste. Záleží na vlnové délce (kvalitě), množství (kvantitě), úhlu a trvání (fotoperiodě).

Rostliny reagují na světlo fotosyntézou v rozmezí fotosynteticky aktivního záření (Photosynthetically Active Radiation – **PAR**) mezi 400 a 700nm. S tím, že nejvíce v části s modrým a červeným světlem. Rozpětí mezi 500 a 600nm má velmi malou absorbci kvůli rozptylu světla mezi molekulami vody v rostlinných buňkách listů a vzduchem ve vnitrobuněčných prostorách.

Relativně novým prvkem pozorování je ovšem i spektrum mezi 700 a 800nm označované jako „daleká červená“ (Far Red, občas taky Deep Red). Světlo o této vlnové délce je sotva viditelné lidským okem, pouze jako velmi tmavá červená o malém jasu. U rostlin toto záření podporuje velmi rychlý růst, neboť si rostlina myslí, že je ve stínu pod jinou vyšší rostlinou, která blokuje světlo ze slunce. Tento jev je označován jako **shade-avoidance response**, tedy reakce na vyhýbání se stínu [19]. Cílem tohoto růstu je získat co nejvíce světla ze slunce pro sebe. Princip tohoto chování tkví v tom, že rostliny mají 2 fytochromy (fotoreceptory pro červenou barvu[2]), jeden pro červenou P_R a druhý pro dalece červenou P_{FR}. Pokud klesne podíl červené a stoupne podíl daleké červené, rostlina na to zareaguje výše popsaným zrychlením růstu. Důvod, proč je podíl dalece červené vyšší pod rostlinami je ten, že listy absorbují více červenou a méně dalece červenou.

[1] https://www.almanac.com/plant-ph

[2] https://www.studysmarter.co.uk/explanations/biology/plant-biology/phytochromes/



Obrázek 5.1: Spektrum s označeným P_R a P_{FR}[19]

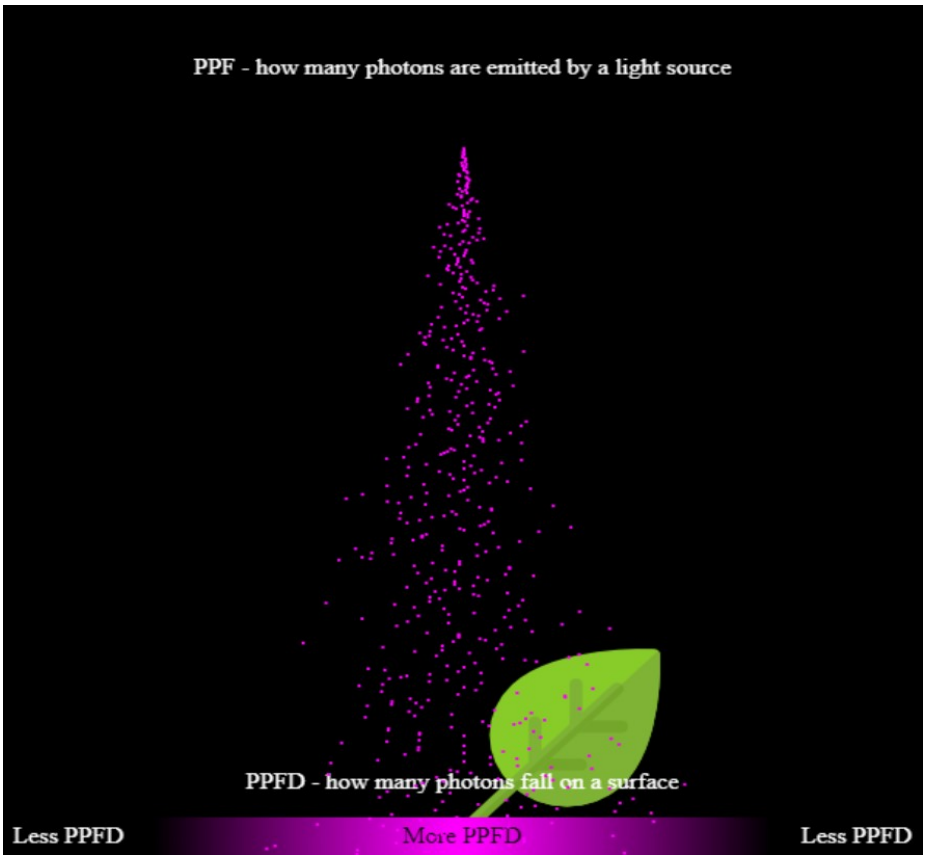
Další veličinou je **PPF** (Photosynthetic Photon Flux), která určuje množství vyzářených fotonů v rozmezí PAR. Její jednotkou je µmol/s. Tato veličina nepočítá s plochou, jedná se pouze o počet vyzářených fotonů. Na tento účel se používá následující veličina.

Tím je **PPFD** (Photosynthetic Photon Flux Density), definuje kolik fotonů dopadne na jeden metr čtvereční za sekundu. Jednotkou je µmol/m2/s. Důležitá je i vzdálenost od zdroje světla. Čím blíže znamená větší koncentrace fotonů. Tento efekt by tedy šlo přirovnat k brokovnici[3]. Pokles koncentrace je i vidět na následujícím obrázku 5.2.

V komerční sféře mají skleníkové rostliny mezi 100 a 200 µmol/m2/s po dobu 16-ti až 20-ti hodin denně.

[3] https://www.growmarket.cz/a/metriky-pro-mereni-pestebniho-svetla-ppf-ppfd-dli

[4] Převzato z: https://www.waveformlighting.com/horticulture/what-is-the-difference-between-ppfd-and-ppf

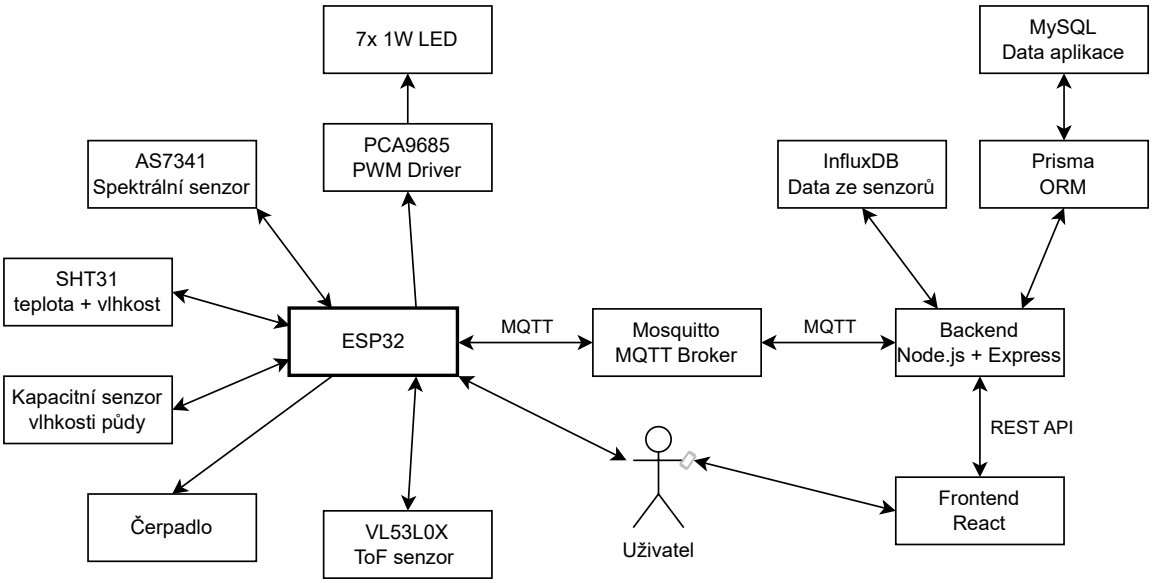


Obrázek 5.2: Rozdíl mezi PPF a PPFD[4]

Kapitola 6

Návrh vestavěného systému

Samotný návrh lze rozdělit do 2 celků – senzorický uzel s aktuátory ovládaný ESP32 a webovou aplikaci běžící v cloudu.



Obrázek 6.1: Obecný návrh systému

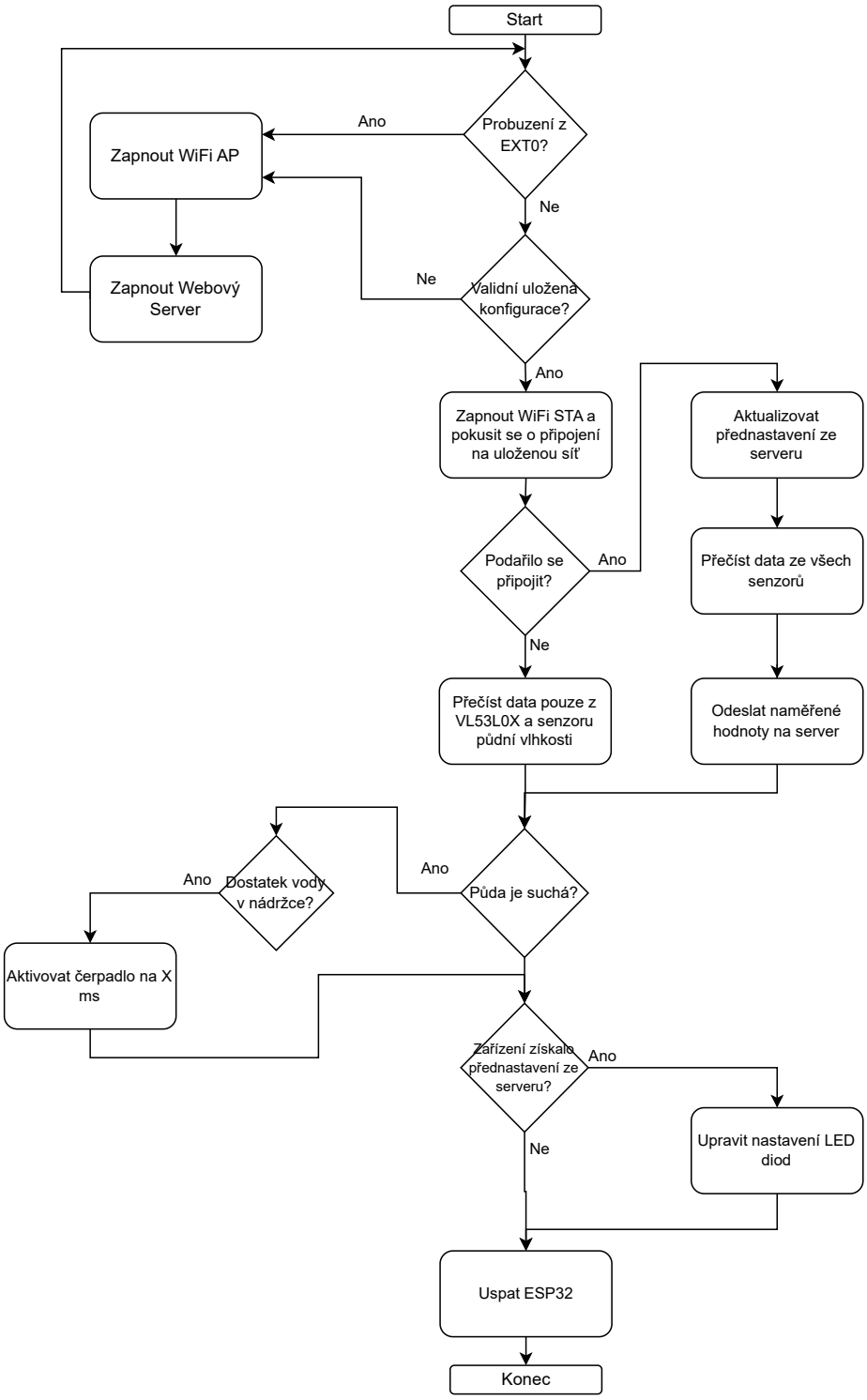
6.1 Senzorický uzel

Jádrem celé jednotky je mikrokontroler obsluhující celkově 4 senzory a 2 aktuátory. S tím, že aktuátor PCA9685 PWM Driver ovládá 7 různých 1W LED diod s rozdílnou vlnovou délkou pro pokrytí rozsahu spektra PAR. Jelikož se jedná o vysoce svítivé diody, tak potřebují chladič a externí napájení, proto bylo nutné zvolit dostatečně výkonný zdroj.

Informace a specifikace použitých senzorů jsou zmíněny v následující kapitole s implementací hardwaru, viz 7.1.

Po stránce softwaru musí být možné skrze mikrokontroler zadat údaje k připojení se na WiFi a možnost unikátní identifikace zařízení. Pro tyto účely se hodí možnost zapnout ESP32 v režimu WiFi přístupového bodu a zobrazit uživateli formulář pro zadání těchto

údajů. Pro identifikaci zařízení je použito UUID[1], které zároveň slouží jako primární klíč v databázi.



Obrázek 6.2: Schéma chování senzorového uzlu

[1]UUID – Universally unique identifier

6.2 Webová aplikace

Jako frontend pro prezentaci dat a změnu nastavení chování senzorického uzlu jsem zvolil jednoduchou webovou aplikaci, neboť je dostupná ze všech typů zařízení a ne jen z jedné platformy, pokud bych se vydal cestou nativní aplikace.

Jelikož celkový návrh počítá s více možnostmi nastavení, tak aplikace obsahuje knihovnu přednastavení, kde jsou jak oficiální (vytvořené admin účtem), tak od jiných uživatelů. U všech lze psát komentáře jako forma zpětné vazby a informace pro jiné uživatele.

Webová aplikace je navržena formou klient–server, kde klientem je webový frontend využívající javascriptovou knihovnu React[2] – pro tvorbu uživatelského prostředí za pomoci komponent, které se následně spojují pro vytvoření celých stránek. Serverem je backend využívající framework Express pod frameworkem Node.js[3] – ten slouží pro spouštění javascript kódu mimo webový prohlížeč. Tím pádem je oddělena logika práce s daty na serveru a zobrazování na klientu. Díky tomu je systém lepší na údržbu. Tyto 2 celky spolu komunikují pomocí REST API.

6.3 Databáze

Z důvodu velkého množství dat ze senzorů není vhodné použít klasickou relační databázi, ale **Time Series DataBase** (TSDB). Tyto databáze jsou pro tyto účely optimalizovány. Jeden záznam obsahuje časový údaj (time stamp) a datový pár měření-hodnota, většinou ale i nějaké identifikační tagy (např. kde se senzor nachází a kdo jej vlastní).

6.3.1 InfluxDB

Nejpoužívanější TSDB je InfluxDB[4], kterou jsem se rozhodl v této práci taktéž použít. Její datový záznam vypadá následovně: `<measurement name>,<tag set> <field set> <timestamp>` [11].

Nejvyšším prvkem datového modelu v této databázi je **Organizace**. Ta obsahuje skupinu uživatelů, kbelíky a úkoly.

Data jsou v databázi rozdělena do kbelíků (**Bucket**). Tento objekt uchovává jednotlivá měření – i více než jeden druh. Navíc je možné na něm nastavit časový údaj, po který se mají uložená data uchovat. Tím pádem lze mít automatické čištění již nepotřebných dat.

Samotné měření (**Measurement**) je vlastně kontejner pro množinu dat. Jedná se o páry typu `field - value`, `tag key - tag value` a časem měření. Jeho identifikace je název ve formě řetězce. Tento název je následně v databázi ve sloupci `_measurement`.

Množina **Field**ů většinou obsahuje samotná data, která se mají uchovat. Jedná se o dvojice kdy klíč a hodnota jsou ve 2 různých sloupcích – `_field` a `_value`.

Množina štítků (**Tag**) je volitelná, ale hodí se pro odlišení dat (slouží jako metadata) a použití v dotazech do databáze. A to z důvodu, že jsou na rozdíl od fieldů indexované a tedy daleko rychlejší. Stejně jako fieldy se jedná o dvojici klíč – hodnota, ale v tomto případě je klíč název sloupce a jednotlivé hodnoty jsou jako řádky.

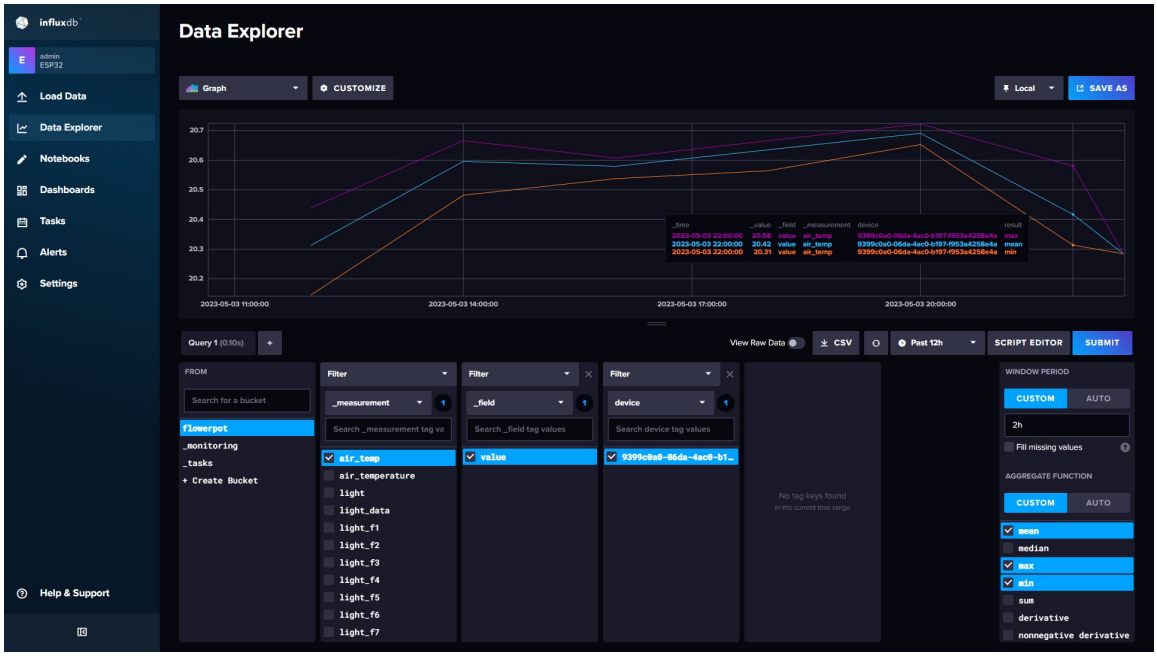
Dalším důležitým pojmem jsou série (**Series**). Ty tvoří množiny bodů, které mají stejné měření, štítky a klíč od fieldu.

Jednotlivé body (**Points**) jsou definovány sérií v určitém čase a hodnotou fieldů.

[2] https://react.dev/
[3] https://nodejs.org/en
[4] https://www.influxdata.com/time-series-database/#ranking

Další výhodou InfluxDB jsou úkoly (**Tasks**), což jsou uživatelem definované skripty, které se spouští v určitém čase[5]. Mohou sloužit pro agregování dat z celého dne (například průměr za hodinu místo velkého množství dat generovaných každých 5 minut) z kbelíku A a následné vložení do kbelíku B, který má nastavený delší limit vymazání dat, neboť je v něm méně záznamů vůči předchozímu kbelíku.

Pro účely této práce se zapisují data obsahující název měření, štítek s UUID zařízení pro identifikaci a následně samotný pár field–value. Tedy výchozí, pouze s jedním identifikačním štítkem.



Obrázek 6.3: Přehled dat v administraci Influx DB

6.3.2 Relační DB

Pro uchování ostatních dat webové aplikace jsem použil relační databázi MySQL.

S oběmi databázemi komunikuje pouze backend webové aplikace. Klient tedy musí zasílat dotazy na server přes REST API a senzorický uzel zasílá data přes MQTT na server.

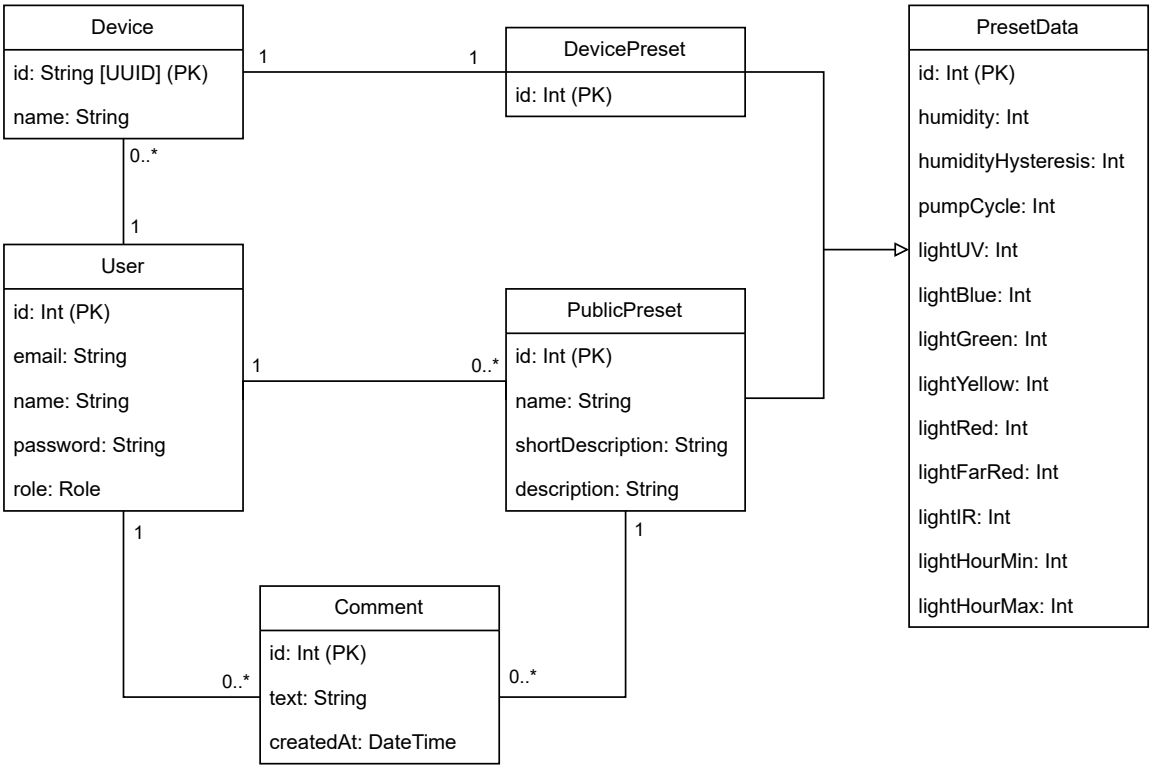
6.4 Komunikace mezi komponentami

Z důvodu použití různých technologí je vhodné mít i různé typy komunikace vyhovující pro daný účel. Pro tuto práci používám přenos přes WiFi pomocí MQTT a HTTP dotazů na API.

6.4.1 Mikrokontroler a backend webové aplikace

Z důvodu ideálně co nejmenší spotřeby zdrojů na ESP32 je pro tuto komunikaci vybrán protokol MQTT, který je detailně popsaný v podkapitole 4.4. V tomto případě MQTT

[5] https://docs.influxdata.com/influxdb/v2.7/process-data/get-started/



Obrázek 6.4: ER diagram tříd v MySQL databázi

přenáší zprávy ve formátování JSON. Z důvodu obousměrné komunikace je mikrokontroler a backend připojen do různých témat.

Prvním tématem je `esp32/alive`. Zde publikuje zprávy mikrokontroler, neboť dává o sobě vědět serveru, že je aktivní a chtěl by získat aktuální informace o přednastavení definované uživatelem. Obsahem zprávy je pouze UUID zařízení. Jelikož je tato zpráva důležitá, tak se používá QoS 1 (tedy garanci doručení alespoň jednou).

Dalším tématem, spíše tématy, je `esp32/devices/{UUID zařízení}`. Zde probíhá komunikace ze strany serveru, který zasílá přednastavení do mikrokontroleru. Jedná se o jediné téma, které je odebíráno mikrokontrolerem. Zprávy zde opět používájí QoS 1.

Pro zasílání dat na server z mikrokontroleru slouží téma `esp32/sensors`. Zde publikované zprávy obsahují UUID zařízení pro identifikaci a jeho aktuální naměřené hodnoty ze senzorů. Jelikož se jedná pouze o data pro vizualizaci na frontendu webu v grafech, tak je zde použit QoS 0 (zpráva je doručena maximálě jednou, ale může se stát, že nebude doručena vůbec).

6.4.2 Frontend a backend webové aplikace

Pro účely komunikace mezi těmito komponentami by nebylo příliš vhodné komunikovat přes broker jak u MQTT, proto se zde používá `REST API`. Jak funguje tento způsob komunikace je již popsáno v podkapitole 4.3. Data jsou přenášena taktéž ve formě JSON objektů.

Aplikace používá koncové body odpovídající datovému modelu v relační databázi a jeden pro data ze senzorů. Jedná se o následující body:

- `/users` – primárně pro čtení uživatelů z databáze, jelikož tvorba uživatelů probíhá skrze koncové body určené pro autentizaci
- `/records` – slouží pro čtení záznamů ze senzorů v InfluxDB
- `/devices` – pro práci s jednotlivými zařízeními, plná podpora CRUD
- `/presets` – slouží pro editaci přednastavení, jak privátních pro zařízení, tak veřejných. Obsahuje ještě koncový bod `/data`, který vrací pouze data o přednastavení, plná podpora CRUD
- `/comments` – pro CRUD operace s komentáři

Pro autentizaci uživatele slouží ještě další samostatné koncové body, které jsou definované v `authRoutes.js`, a to: `/login`, `/register`, `/logout`. A pro změny údajů uživatele `/changePassword` a `/changeEmail`.

6.4.3 Mikrokontroler a klient (webový prohlížeč)

Tato komunikace probíhá pouze pokud je zařízení v režimu WiFi přístupového bodu při konfiguraci. Jedná se podobně jako u 6.4.2 komunikaci pomocí HTTP API, ale obsahuje pouze 2 koncové body se 3 možnostmi.

- `/ (GET)` – Jako odpověď se odešle stránka s konfiguračním formulářem do webového prohlížeče uživatele.
- `/ (POST)` – Zaslání dat od uživatele na mikrokontroler.
- `/restart` – Bez přenosu dat, pouze pro restartování zařízení.

Kapitola 7

Implementace

V této kapitole se budu věnovat samotné realizaci návrhu, výběru konkrétních senzorů a překonání strastí, které vypluly na povrch.

Samotná kapitola je rozdělena na více sekcí. První se týká použitých součástek a elektronického zapojení 7.1. Poté tisknutí květináče na 3D tiskárně 7.3. Následující pasáže se týkají již samotné implementace v softwaru, jak na ESP32 7.4, tak část s webovou aplikací 7.6.

Prvně jsem chtěl tuto práci implementovat v prostředí Arduino IDE – zde jsem i napsal demo implementaci obsahující většinu návrhu bez senzorů (tedy primárně komunikaci pomocí MQTT, konfiguraci přes WiFi AP a experimenty se spánkovými režimy ESP32). Důvod byl, že je zde více dostupných knihoven, ale pak jsem vyzkoušel ESP-IDF pod PlatformIO a práce mi ve výsledku přišla příjemnější, ba dokonce připojování na domácí WiFi síť a MQTT probíhalo rychleji. Navíc možnost eliminovat blokování celého programu kvůli `delay()` metodám, které jsou i v Arduino knihovnách, pomocí FreeRTOS tasků se taky hodí.

7.1 Hardware

Pro účely této práce jsem jako mikrokontroler použil **FireBeetle 2 ESP32–E**. Informace jsem čerpal z wiki stránek produktu [10], které byly v březnu roku 2023 upraveny s lepším přehledem pinů a jejich vlastností. Jedná se o vylepšenou verzi FireBeetle ESP32, jak již název napovídá, obsahuje modul ESP32–E (opravuje primárně chyby na HW úrovni[1]).

Výhody tohoto devkitou jsou, že obsahuje PH2.0 konektor pro Li–ion baterie a i zabudovaný nabíjecí obvod. Dále pak GDI konektor pro připojení displeje. Pro zpětnou vazbu jsou přímo na desce 2 uživatelsky ovládané LED diody. Jedna zelená LED dioda na pinu D9 a poté WS2812 RGB LED dioda na pinu D8. Důležitá vlastnost je i ta, že tato deska má velmi nízkou spotřebu ve spánkovém režimu, již její předchůdce v tomto vynikal[2]. Pro dosažení ještě větší úspory energie je možné přeříznout tenkou spojnici na desce, která odpojí zabudovanou RGB diodu při napájení z baterie pro pokles odběru o 500 µA. Spotřeba ve spánku je pak 13 µA. Menší nevýhoda vůči jiným ESP32 devkitům je interní flash pamět o velikosti 4MB (předchůdce měl 16MB), ale i tak je dostatečná.

Co se týče senzorů, tak na okolní teplotu a vlhkost byl použit **SHT31** jelikož je přesnější a i levnější než často používaný **DHT22** [20]. Senzor zvládá měřit teplotu v rozsahu -40 až

[1] https://www.dfrobot.com/product-2195.html
[2] https://diyi0t.com/reduce-the-esp32-power-consumption/

120°C s přesností 0,3°C a vlhkost v rozsahu 0 až 100% RH s přesností 2%[3]. Tento senzor je zapojen v nepájivém poli vedle samotného devkitu. Jeho I^{2}C adresa je 0x44.

Pro monitoring hladiny vody v nádržce byl použit ToF (Time of Flight) senzor **VL53L0X**. Jedná se spíše o nevšední řešení pro hobby projekt (invazivní plováky jsou častější), ale šlo dohledat použití konkrétního senzoru i pro tento účel, který skončil úspěchem[4]. Princip tohoto senzoru spočívá v tom, že vyšle infračervený laserový paprsek (není vidět lidským okem) a následně měří dobu jeho letu než se odrazí od nějakého objektu zpátky k senzoru[5]. Nevýhoda tohoto řešení je ta, že je zde minimální vzdálenost pod kterou nezvládne měřit – v případě VL53L0X to jsou zhruba 3cm. Je tedy nutné si dát pozor na to, aby voda nevystoupala nad tuto úroveň, což je vyřešeno umístěním senzoru na podstavu. Ta je tvořena distančním stojánkem určeným pro krabici pizzy nalepeným na víku nádržky s vodou. I^{2}C adresa tohoto senzoru je 0x29.

Nejexotičtější senzor celého projektu je spektrální senzor světla **DFRobot AS7341**, který je i dostatečně přesný pro měření PPFD [7]. Informace jsem čerpal z wiki stránek produktu [9]. Měří v 11-ti kanálech. Z těchto kanálu je 8 (pojmenovány „F1“ až „F8“) dle vlnové délky s rozsahem začínajícím na 405nm a končícím u 690nm – což zhruba odpovídá rozsahu PAR na který reagují rostliny (popsáno v kapitole 5.2). Dalším kanálem s filtrem dle vlnové délky je „NIR“ (Near InfraRed), který zvládá měřit v blízkosti IR záření okolo 900nm. Předposlední kanál je bez filtru na vlnovou délku nazýván „Clear“, který měří všechno dopadající světlo. Poslední kanál slouží pro detekci blikání ve frekvenci 50 nebo 60Hz. Výhodou tohoto senzoru je to, že měří světlo dopadající z okolí a nefunguje pouze na principu poznání barvy nasvíceného předmětu pomocí zdroje světla, což by pro tuto práci nebylo vyhovující. Ovšem i tuto funkci může zastoupit díky 2 LED diodám určným pro tento účel. Senzor obsahuje 6 nezávislých 16-ti bitových ADC převodníků, proto je nutné měřit ve 2 krocích pro získání všech kanálů. Jejich citlivost lze nastavit v 11-ti stupních změnou parametru `AGAIN` v rozmezí 0 až 10. To odpovídá hodnotám 0,5 a poté druhé mocnině od 1 po 512[3]. Reálně ovšem hodnoty nejsou násobeny přesně touto hodnotou, ale jsou zde mírné odchylky, které jsou uvedeny v dokumentaci AS7341. Jeho I^{2}C adresa je 0x39.

Až na senzor půdní vlhkosti, který je analogový, komunikují senzory s ESP32 pomocí I^{2}C sběrnice, což znamená pouze propojit SDA a SCL piny a samozřejmě napájení 3V3 a GND. Jediný požadavek tedy je, aby měly všechny senzory unikátní I^{2}C adresu a nekomunikovali po stejné sběrnici zároveň. Použité senzory tento požadavek splňují, ovšem po I^{2}C komunikuje i PWM driver, ale jeho adresa je taktéž unikátní – 0x40. V případě, kdyby nebyla, tak je možné jeho adresu změnit spájením plošek na jeho desce.

7.1.1 Senzor vlhkosti půdy

Následující informace jsem čerpal ze srovnání více typů senzorů na měření vlhkosti z [18].

Pro měření vlhkosti v půdě lze použít několik typů senzorů. Odporové fungují na principu měření vodivosti mezi 2 elektrodami zanořenými do půdy, zvýšení obsahu vody znamená snížení odporu. Odezva na změnu obsahu vody je zhruba 9 minut. Nevýhoda těchto čidel je ale ta, že mají obnažené elektrody a tedy zde probíhá elektrolýza a tím pádem

[3] https://www.laskakit.cz/senzor-teploty-a-vlhkosti-vzduchu-sht31/

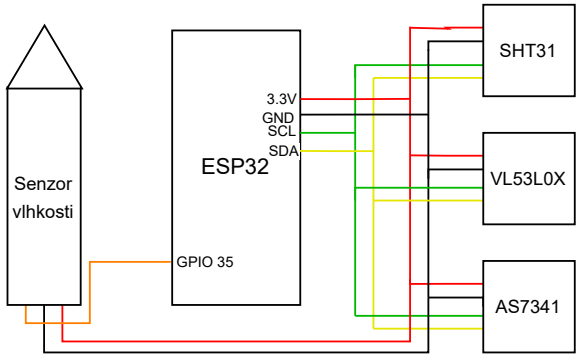
[4] https://www.hackster.io/team-protocentral/liquid-level-sensing-using-a-laser-tof-sensor-d04232

[5] https://www.makeuseof.com/what-is-time-of-flight-sensor-how-does-tof-work/

degradují při měření. Proto je nutné do tohoto typu senzoru pouštět proud po co nejmenší časový úsek.

Druhým typem jsou kapacitní čidla. Na rozdíl od odporových mají jen jednu sondu místo dvou, navíc není obnažená, proto tento typ senzorů nepodléhá degradaci při měření. Další výhodou je rychlejší reakce na změnu obsahu vody v půdě, jen okolo jedné minuty. Fungují na principu změny dielektrické konstanty, čím více vody, tím je vyšší. Nevýhodou je trochu vyšší cena, ale hlavně to, že konkrétní použité čidlo[6] nemá voděodolnou horní část sondy, neboť jsou zde elektronické součástky. Tento problém byl vyřešen zalitím obvodu do lepidla z tavné pistole.

Senzor půdní vlhkosti je zapojen do GPIO 35, což je pin 1. ADC převodníku na 7. kanálu. Pro tento účel se musel použít ADC1 a ne ADC2 z důvodu, že ten je používán WiFi, která je aktivní. Rozlišení převodníku je nastaveno na plných 12 bitů a attentuation (útlum) je nastaven na 11dB, což je maximální možný pro největší rozsah napětí[7].



Obrázek 7.1: Schéma zapojení senzorů

7.1.2 LED osvětelní

Ovládání LED diod obstarává 16-ti kanálový PWM driver **PCA9685**. Jedná se o verzi z LaskaKit, ale specifikace je stejná jako deska od Adafruit. Odtud jsem tedy čerpal informace [12]. Má oddělené napájení desky (VCC) a napájení servo motorů (V+) s limitem 10 A a 6 V. Což plně dostačuje pro LED diody. Zároveň každý datový PWM pin má v sérii odpor o hodnotě 220 Ω, což limituje maximální proud na 25 mA. Tohoto limitu se ovšem nikdy nedosáhne, neboť deska je v případě této práce napájena z 3V3 pinu, tedy maximální proud je 15 mA.

Důvod, proč jsem použil tento driver a ne PWM přímo z ESP32 je ten, že diod je 7 a jsou vzdálené od ESP32, tedy je rozdíl natahovat mnoho dlouhých kabelů pro každou diodu zvlášť a nebo 2 kabely pro I^2C propojení a 2 pro napájení. Zároveň se tím uspoří mnoho GPIO pinů.

Samotné LED diody jsou běžné 1W SMD s chladičem. Jejich vlnové délky jsou následující: 400, 450, 525, 595, 625, 690 a 730nm. Jejich úhel svitu je 120° (díky čočce z výroby), nebylo tedy potřeba používat difuzor či jiné čočky.

Jelikož mají LED diody vysoký odběr proudu na to, aby je nápajelo ESP32, bylo nutno rozdělit napájení MCU a diod. Použil jsem USB zdroj (5V @ 3.1A) s 2 výstupy – jeden USB-

[6] https://www.laskakit.cz/kapacitni-cidlo-pro-mereni-vlhkosti-pudy/

[7] https://docs.espressif.com/projects/esp-idf/en/v4.4.4/esp32/api-reference/peripherals/adc.html#adc-attenuation

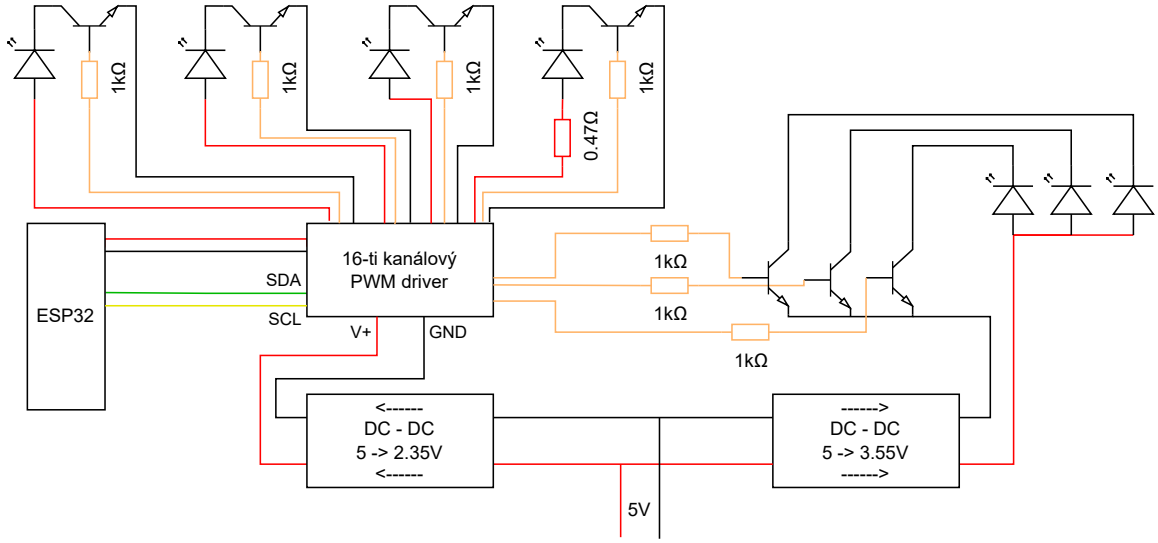
C kabel pro napájení ESP32 a USB-A zdířka pro zapojení dalšího kabelu, který vede energii k osvětlení. Jelikož mají použité LED diody různé požadavky na napětí, tak byly rozděleny do 2 shluků. Pro snížení napětí se používají dva DC – DC stepdown měniče **LM2596**, které mají maximální proud 2A, chvilkově zvládnou až 3A. První shluk o 4 diodách má napětí 2.35V, IR dioda má ovšem limit 2.1V, proto je zde použit odpor o hodnotě 0.47Ω. Druhý shluk napájí zbylé 3 diody 3.55V. Detailnější spotřebě energie je věnována následující sekce 7.2.

Pro rozsvícení diod se používají NPN tranzistory zapojené jako přepínač. Jedná se o **BC338-40**, které mají maximální proud 0.8A a dostatečný koeficient h_{FE}. Jejich báze jsou napojeny na jednotlivé datové piny PWM driveru, ty zvládají maximální proud 25mA, ale mají sériově zapojen 220Ω odpor, což limituje proud při použití desky na 3V3 na 15mA, ale to by i tak povolovalo velký proud do LED diod. Pro proud 300mA je dle specifikace h_{FE} minimálně 170, typicky 320. Z tohoto důvodu jsem zde připojil ještě odpor o hodnotě 1kΩ, tedy celkově 1220Ω před bází tranzistoru:

$$I_B = \frac{U_B - U_{BE}}{R_B}$$

$$I_B = \frac{3,3 - 0,7}{1220} = 2,13mA$$

Pokud uvažujeme h_{FE} 170, tak to odpovídá proudu zhruba 362mA mezi kolektorem a emitorem. Což je dostatečné.



Obrázek 7.2: Schéma zapojení LED diod

7.1.3 Čerpadlo

Pro zalévání bylo použito mini čerpadlo s odběrem pouhých 220mA[8], tedy je možné jej zapojit přímo na Vcc pin mikrokontroleru (zvládnul by to i 3V3 pin, dle specifikace je maximální odběr proudu 600 mA[9]). Motor je aktivován pomocí NPN tranzistoru **BC338-40** (stejný jako u LED diod), zde jsem použil odpor o hodnotě 2kΩ:

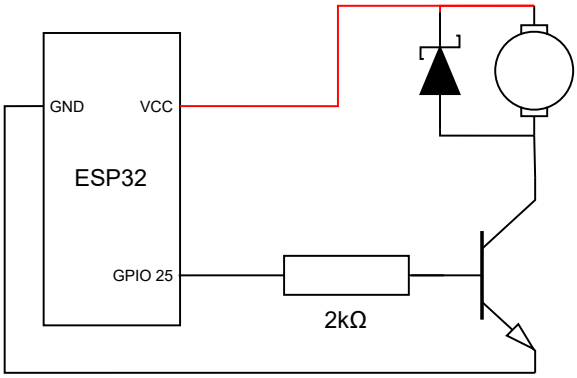
[8] https://navody.dratek.cz/navody-k-produktum/mini-dc-cerpadlo.html
[9] https://www.dfrobot.com/product-2195.html

$$I_{BE} = \frac{I_C}{h_{FE}}$$
$$I_{BE} = 0.22/170 = 1,294mA$$

$$R_B = \frac{U_B - U_{BE}}{I_B}$$
$$R_B = \frac{3,3 - 0,7}{0,001294} = 2009,27\Omega$$

Pro jistotu jsem použil i paralelně zapojenou Schottkyho diodu jako flyback, pro zamezení nežádoucí špičky napětí při odpojení DC motoru čerpadla. Tento jev nastane v momentě odpojení přívodu energie do elektromotoru, který pak začne fungovat jako generátor, neboť se stále ještě točí. Reverzně zapojená Schottkyho dioda tuto energii spotřebuje a tedy není uvolněna do systému[10].

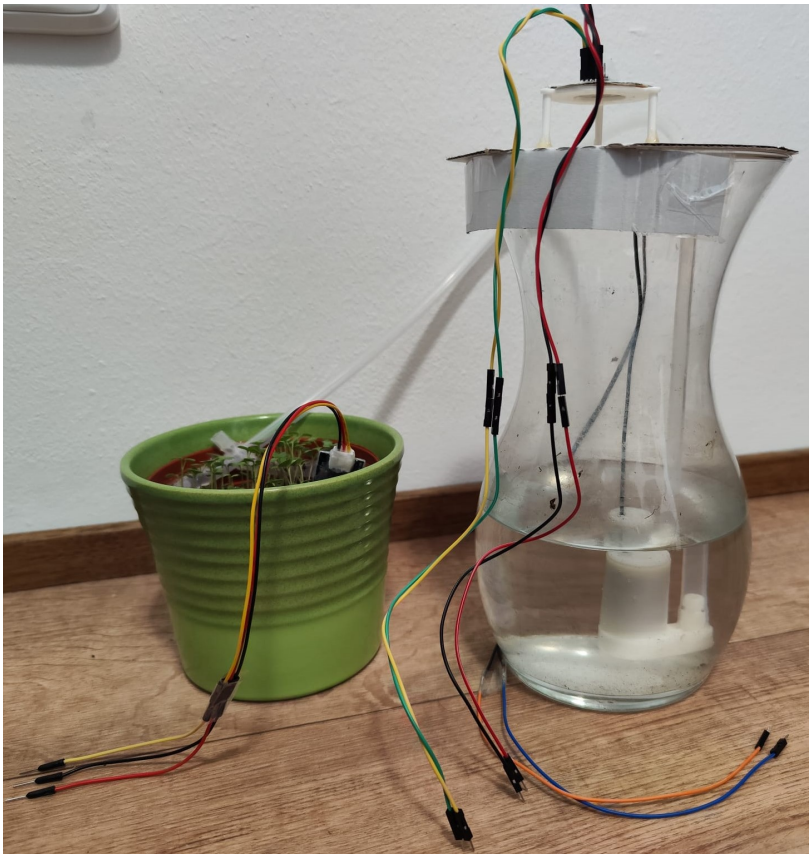
Při tomto zapojení pumpa zvládnula přečerpat 100ml vody zhruba za 4 sekundy, tedy 25ml/s. Toto bude využito při nastavení objemu vody na jedno zalívání z webu.



Obrázek 7.3: Schéma zapojení čerpadla

Aby byla voda při zalívání rozprostřena po celém květináči, tak okolo horní hrany je přilepena hadička do tvaru kruhu ve které jsou dírky. Voda tak teče rovnoměrně ze všech stran shora na hlínu a ne do jednoho místa či pouze do květináče zespod.

[10] https://www.circuitbread.com/ee-faq/how-does-a-flyback-diode-work



Obrázek 7.4: Sestavená část pro zalívání

7.2 Odběr energie

Jak již bylo zmíněno v podsekci 7.1.2, pro napájení celého květináče slouží 5V napájecí zdroj o maximálním odběru proudu 3,1A.

Největší spotřebu mají LED diody. Ovšem ty jsou za DC–DC stepdown měniči napětí. Při výpočtech vycházím z části podsekce 7.1.2, konkrétně z popisu spotřeby LED diod.

V kalkulacích se používá vzorec na výkon $P = U * I$.

$$\text{První skupina} = 3,55 * 0,35 * 3 = 3,7275W$$

$$\text{Druhá skupina} = 2,35 * (0,7 + 3 * 0,4) = 4,465W$$

$$\text{Celkem} = 3,7275 + 4,465 = 8,1925W$$

Je ovšem nutné započítat ztrátu z měničů LM2596. Dle specifikace je maximální účinnost 92%. Ovšem v tomto konkrétním použití lze očekávat účinnost jen okolo 70%[11].

$$\text{Odběr LED modulu vč. ztrát} = 8,1925/0,7 = 11,7W$$

Maximální odběry proudu pro ostatní použité komponenty jsou následující:

[11] https://lygte-info.dk/review/Power%20Adjustable%20buck%20converter%20LM2596%20UK.html

- `AS7341` – 5mA (bez používání přisvětlovacích LED – případ této práce. Při jejich zapnutí až +42mA) [3];
- `SHT31` – 1.5mA [4]
- `VL53L0X` – 40mA [5]
- `Senzor vlhkosti půdy` – 5mA[12]
- `PCA9685` – 10mA + odběr z PWM pinů (7 * 2,13mA) = 14,91mA [2]
- `Pumpa` – 220mA[13]

Jelikož senzory a PCA9685 berou proud z 3V3 pinu devkitu, tak je zde limit 600mA daný použitým LDO[14]. Tento limit je ale dostatečný, neboť je využito maximálně $5 + 1.5 + 40 + 5 + 10 + 14,91 = 76,41mA$ z jeho kapacity senzory/aktuátory.

$$\text{Maximální celkový odběr senzorů, PCA9685 a desky} = 5 * 0,6 = 3W$$

$$\text{Odběr pumpy} = 5 * 0,22 = 1,1W$$

Maximální výkon použitého zdroje je $P = 5 * 3.1 = 15,5W$. Při součtu spotřeby senzorů, aktuátorů a samotného devkitu s ESP32 ovšem vychází $11,7 + 3 + 1,1 = 15.8W$. Avšak této hodnoty není dosaženo díky dočasnému vypnutí osvětlení pokud je zapnuté čerpadlo.

Reálně by se ovšem nemělo nic stát ani kdyby osvětlení bylo stále zapnuté, neboť odběr senzorického uzlu (bez osvětlení) je nižší než limit 600mA z LDO a senzory již neměří při pumpování vody.

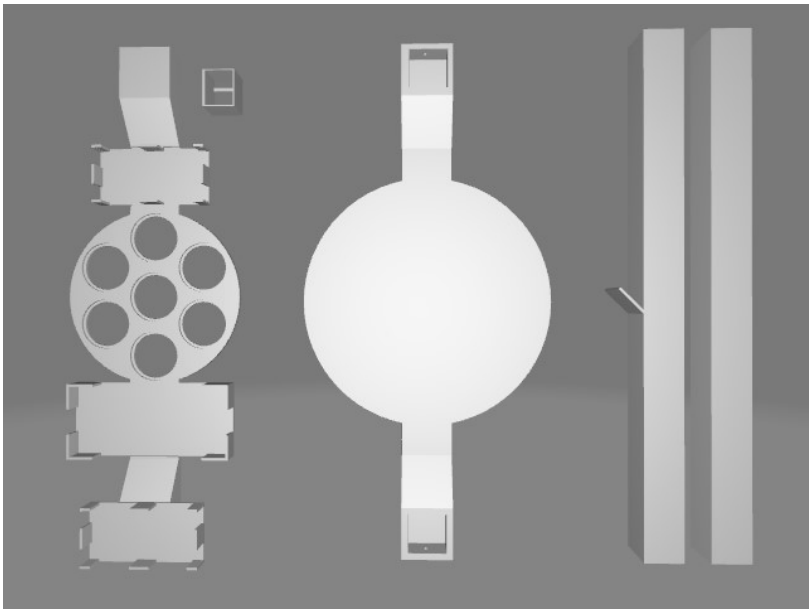
7.3 3D model a tisk

Jelikož bylo nutné upevnit LED diody nad květináč a zároveň zaručit, že se celá konstrukce nezhroutí, tak jsem se vydal cestou 3D tisku. Model (Obrázek 7.5) obsahuje otvory pro diody a místa pro uchycení obou měničů napětí a PWM driveru. Stability je docíleno zatížením podstavy květináčem.

Jako materiál pro tisk byl zvolen PETG[15]. Neboť na rozdíl PLA[16] má vyšší tepelnou odolnost – začne měknout až okolo 80°C vůči 60°C[17]. Tento parametr je zde důležitý z důvodu, že 1W LED diody se i s chladičem velmi zahřívají.

Bohužel jsem měl udělat otvory pro LED o něco málo širší, neboť přesná šířka 20mm není ve skutečnosti dostatečná, zřejmě kvůli hranám chladičů SMD diod. Proto diody hezky nesedí a bylo nutno improvizovat s dodatečným upevněním.

[12] https://www.laskakit.cz/kapacitni-cidlo-pro-mereni-vlhkosti-pudy/
[13] https://navody.dratek.cz/navody-k-produktum/mini-dc-cerpadlo.html
[14] https://www.dfrobot.com/product-2195.html
[15] PETG – Polyethylene Terephthalate Glycol
[16] PLA – Polylactic Acid
[17] https://3dsolved.com/3d-filament-glass-transition-temperatures/



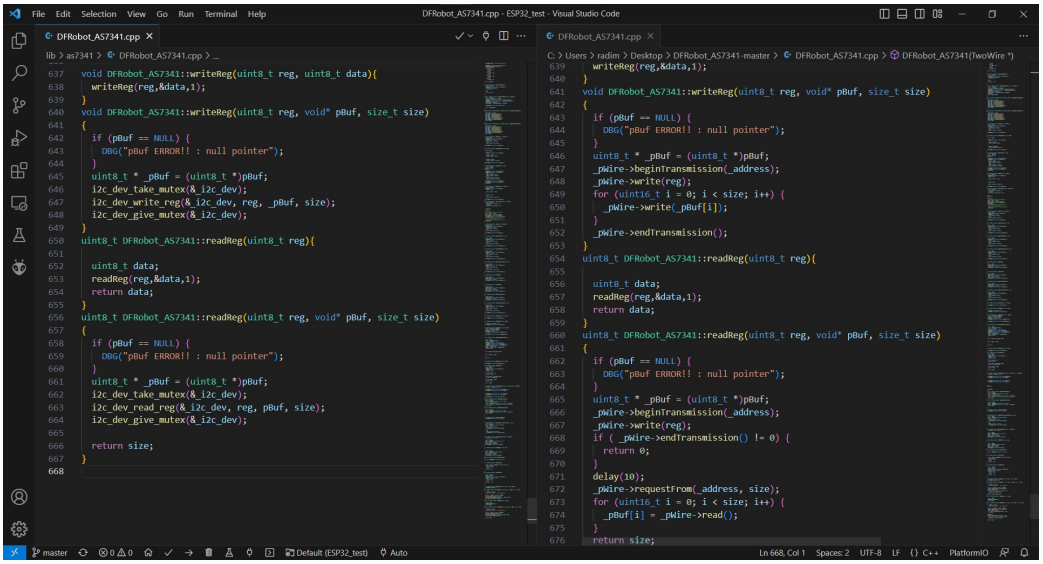
Obrázek 7.5: 3D model konstrukce

7.4 Knihovny v ESP-IDF

Menší komplikací při použití ESP-IDF byl problém v podání neexistujících knihoven pro spektrální a ToF senzor. Pro ostatní jsem našel knihovny v `ESP-IDF Components library` [22]. Proto jsem se rozhodl, že zkonvertuji knihovny určené pro Arduino IDE na ESP-IDF. Jedná se o DFRobot AS7341[18] a DFRobot VL53L0X[19].

Jelikož ostatní použité knihovny pro senzory využívají **i2cdev**[20] knihovnu pro správu I^2C sběrnice, tak jsem ji využil taktéž, abych mohl používat všechny I^2C senzory zároveň a nemusel řešit kompatibilitu knihoven mezi nimi. Jedná se o jednoduchou pomocnou knihovnu využívající semafory pro zablokování zápisu či čtení ze sběrnice jiným zařízením, než aktuálně používaným. Primárně bylo nutné přepsat zápisy a čtení do/z registrů přes sběrnici (detail na obrázku 7.6) a nahradit `delay()` metody z Arduina metodou `vTaskDelay()` z FreeRTOSu.

[18] https://github.com/DFRobot/DFRobot_AS7341
[19] https://github.com/DFRobot/DFRobot_VL53L0X
[20] https://esp-idf-lib.readthedocs.io/en/latest/groups/i2cdev.html



Obrázek 7.6: Část ESP-IDF konverze knihovny pro AS7341

7.5 ESP32

Implementace softwaru na mikrokontroleru je samozřejmě rozdělena do více souborů, jejich funkcionalita je následující:

- `main.cpp` – Zde začíná celý program v metodě `app_main()` (definováno ESP-IDF), nastavuje se GPIO, přerušení pro konfiguraci a zapíná WiFi STA. Zároveň je zde jednoduchá implementace debounceru tlačítka pro přepínání WiFi režimů. Debounce je metoda určena pro ošetření zákmitů tlačítka při jeho zmáčknutí/pouštění, aby nebylo zaregistrováno několik aktivací místo jedné.
- `actuators .cpp/.hpp` – Ovládání aktuátorů, tedy pumpy a LED osvětlení
- `mqtt .cpp/.hpp` – Zprostředkovává komunikaci přes MQTT broker s backendem webové aplikace. Tedy přijímá přednastavení a zasílá naměřené hodnoty ze senzorů.
- `sensors .cpp/.hpp` – Provádí jednotlivá měření a ukládá je do datové struktury. Jakmile je vše hotovo, inicializuje aktuátory.
- `utility .cpp/.hpp` – Obsahuje metody použitelné z více částí kódu. Hlavičkový soubor definuje většinu konstant použitých v programu.
- `webserver .cpp/.hpp` – Stará se o zobrazení a zisk dat z webového formuláře určeného ke konfiguraci WiFi a ID zařízení.
- `wifiswitcher .cpp/.hpp` – Slouží k přepínání mezi režimy WiFi AP a WiFi STA.

Implementace používá globální „Event Group“ pojmenovanou `STATUS_GROUP` (definováno v `main.cpp`). Jedná se o skupinu určenou k synchronizaci jednotlivých úkolů, aby se následná akce spustila až po dokončení všech/konkrétních předešlých [1]. V případě této práce slouží k synchronizaci úkolů pro propřipojení na WiFi, MQTT, status senzorů a následně pro spouštění aktuátorů.

7.5.1 Nastavení Wifi a ID

Pro konfiguraci WiFi a ID zařízení je použit WebServer, který je na IP adrese `192.168.1.1` po připojení se na WiFi AP vytvořený mikrokontrolerem.

Tento WiFi přístupový bod se zapíná automaticky pokud nejsou WiFi údaje a ID zařízení uloženy v NVS paměti zařízení. Případně manuálně po stisku tlačítka. Jeho název je „ESP32“ s heslem „1234567890“. Jako indikace, že je zapnut tento režim slouží svítící zelená LED dioda na desce.

Nastal zde ovšem problém, že hlavičky požadavků byly větší než defaultní nastavení HTTP serveru a s každým odesláním formuláře z webového UI se mikrokontroler zhroutil. Jako oprava posloužilo zvýšit limit `CONFIG_HTTPD_MAX_REQ_HDR_LEN` z 512 na 1024 v sdkconfig souboru (pro použitou desku `sdkconfig.dfrobot_firebeetle2_esp32e`).

Vycházel jsem z oficiální dokumentace HTTP Serveru pod ESP-IDF[21].

Samotný WebServer obsahuje tři URI handlery:

- `uriGet` – Slouží pro odeslání webové stránky klientovi. Zasílá stránku po částech s využitím `httpd_resp_send_chunk` z důvodu zobrazení názvu WiFi a ID zařízení (pokud jsou uloženy) bez jejich konkatenace ke konstantě obsahující celý HTML + CSS kód. Pro zjednodušení se zobrazují aktuální hodnoty pomocí JavaScriptu (díky využití getElementById se vybere prvek formuláře a nastaví se mu hodnota uložená v zařízení), abych mohl data připojit na konec a nemusel je měnit ve středu dlouhého řetězce – obsahu celé stránky, který je v jedné statické proměnné.
- `uriPost` – Prohledá zaslaná data z konfiguračního formuláře pro údaje a uloží je do interní paměti zařízení.
- `uriRestart` – Restartuje zařízení. Tím pádem se zařízení následně zapne v normálním režimu a využívá nakonfigurované údaje.

7.5.2 Senzory

Každý senzor má definovanou bitovou masku (Event Bit), která se po dokončení nastaví do `STATUS_GROUP`.

Postup každého měření probíhá následovně:

1. Inicializace senzoru.
2. Samotné měření.
3. Zapsání do datové struktury a JSONu
4. Nastavení bitové masky v `STATUS_GROUP`.

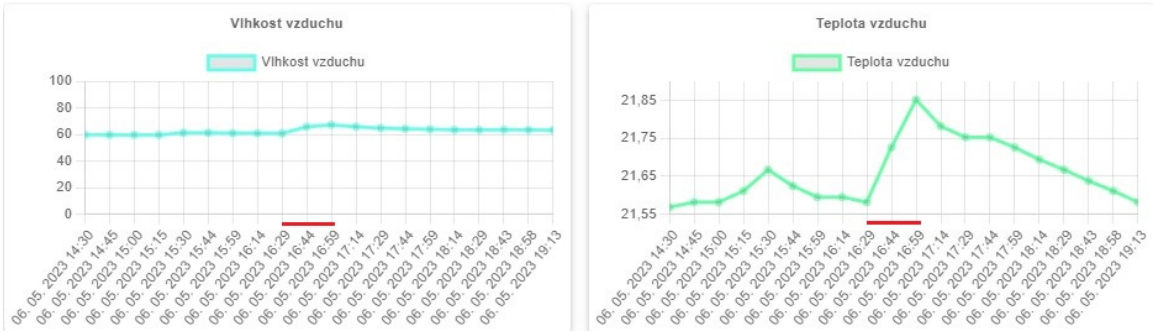
Jak již bylo zmíněno v sekci 7.1, čidla komunikují pomocí I^2C s vyjímkou senzoru půdní vlhkosti. Ten je analogový a proto používá zabudovaný AD převodník v ESP32. Jeho návratová hodnota je menší, čím vyšší je detekovaná vlhkost. Pro namapování hodnot se používá metoda **map()** z utility.cpp/.hpp. Hodnota odpovídající 100% vlhkosti byla změřena při saturaci hlíny vodou. Pro zisk hodnoty pro 0% vlhkosti by byl ideální způsob kalibrace změřit hlínu vysušenou v peci/troubě, ale při cenách energií v době řešení této

[21] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp_http_server.html

práce jsem se uchýlil k méně přesnému řešení – použít hodnotu naměřenou senzorem, když je vytažený z hlíny. Neboť proces vysoušení trvá alespoň 1 den.

Kalibrace ToF senzoru **VL53L0X** pro snímání hladiny vody v nádržce byla jednoduchá – stačilo pouze zaznamenat naměřenou vzdálenost při plné nádržce a když byla úroveň vody menší než zvládlo čerpadlo ještě nasát. Tyto hodnoty jsou opět mapovány metodou `map()` na rozmezí 0 až 100%, opět reverzním způsobem – větší číslo (vzdálenost) odpovídá méně vody v nádržce. Senzor se projevil jako přesný (porovnáváno vůči pravítku), ale velmi náchylný na rušení – při použití jiného nepájivého pole stačilo mírně pohnout s kontakty (i mimo aktivní měření) a hodnoty byly jiné, či se senzor ani neinicializoval.

Senzor teploty a vlhkosti vzduchu **SHT31** jsem porovnával s pokojovým termostatem (ovšem pouze s teploměrem). Naměřené hodnoty se zdály pokaždé mírně vyšší – většinou o necelý 1°C, ale to by mohlo být ovlivněno i umístěním senzoru. Co se týče otestování vlhkoměru, tak již sestavený květináč byl položen v místnosti vedle koupelny (propojeno chodbou, ne přímo), která byla zavřená počas používání sprchy. Po otevření dveří bylo vidět, že vlhkost stoupla zhruba o 7% a teplota o 0,3°C za půl hodiny a následně klesala k původním hodnotám. Rozdíl teploty se zdá být malý, ale na grafech 7.7 je změna hezky vidět.



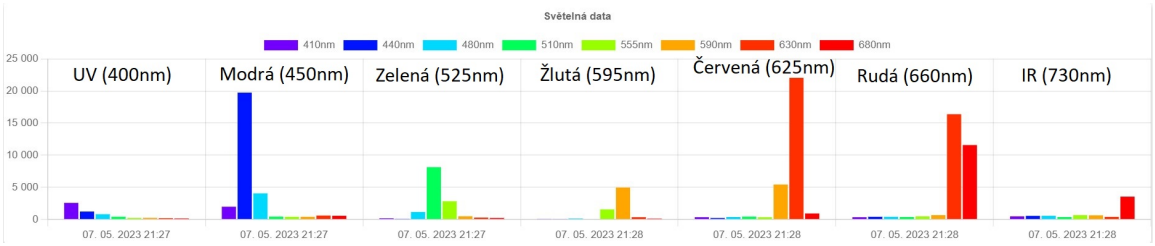
Obrázek 7.7: Test SHT31

Implementace spektrálního senzoru **AS7341** je zajímavější než ostatní v tom, že navrácené hodnoty jsou v rozmezí 0 až 1000. Což ovšem nestačí pro pokrytí všech intenzit. Dle dokumentace je výchozí hodnota parametru `AGAIN` nastavena na 9 [9], tedy zesílení 256x. Dle mého pozorování to je dostatečné pro běžné testování světla v domě, ale na přímém slunci či poblíž silného zdroje světla, což použité LED diody jsou, nastane problém se saturováním senzoru a tedy ztrátě podstatné části informace. Z tohoto důvodu je měření implementováno v jednoduchém algoritmu, který mění zmíněný parametr `AGAIN` pro úpravu zesílení převodníků na senzoru v případě, že jsou hodnoty příliš malé či se blíží 1000 tak, aby ke ztrátě nedocházelo. Pro vyzkoušení jsem postupně zapínal LED diody na 100% svítivosti v tmavé místnosti a zaznamenával hodnoty naměřené senzorem drženým ve stejné vzdálenosti od aktivní diody. Na obrázku 7.8 je patrné, že pokud je vlnová délka diody mimo rozsah senzoru, tak je její hodnota nižší. Kalibrace pro získání PPFD hodnoty by byla v domácích podmínkách bez příslušného referenčního zařízení pro měření PPFD ze světla poněkud nereálná, tak jsem se rozhodl převzít výsledné hodnoty modelu z práce zaměřující se využít tento senzor jako měřič PPFD [7]. Tento model je lineární – stačí tedy pronásobit získané údaje ze sezoru koeficienty:

b0	b1	b2	b3	b4
-9,8853	0,0046	0,0136	0,0243	0,0459
b5	b6	b7	b8	
-0,0471	0,0195	0,0178	-0,0026	

$$y_i = b_0 + \sum_{i=1}^{n} (b_i x_i) + \epsilon$$

Pro ověření výsledné hodnoty PPFD jsem vyzkoušel aplikaci `PPFD Meter`[22] využívající senzory pro okolní osvětlení v telefonu. Naměřené hodnoty z AS7341 byly zhruba dvojnásobné – ve jmenované práci použili difuzor, ale senzor by se měl dát používat i bez něj. Proto jsem je následně dělil dvěmi.



Obrázek 7.8: Zaznamenané hodnoty z jednotlivých LED

7.5.3 Aktuátory

Jakmile senzory dokončí svá měření, tak se zapnou aktuátory. To je řešeno pomocí čekání na bity nastavené ze senzorů v `STATUS_GROUP` za využití metody `xEventGroupWaitBits`.

Prvně se aktivuje pumpa za pár podmínek:

- Vlhkost půdy je menší než přednastavená s připočítanou hysterezí
- V nádržce na vodu je dostatek vody
- Doba od posledního zalití je více než 1 hodina

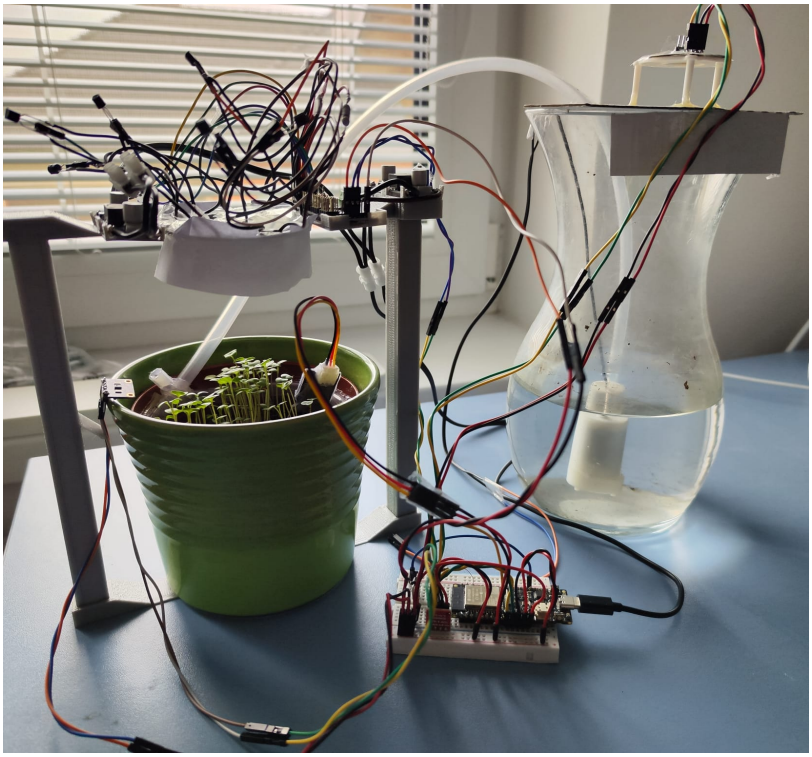
Pokud jsou všechny splněny, tak se pumpa zapne po dobu nutnou k přečerpání přednastaveného množství vody. Zároveň je po tuto dobu vypnuté LED osvětlení z důvodu limitů zdroje, více v kapitole o spotřebě 7.2. Dle měření jsem vypozoroval, že rychlost toku vody je zhruba 25ml/s, což je zmíněno v podkapitole 7.1.3. To odpovídá 40ms na 1ml vody.

Poté se nastavuje svit LED diod. Driver PCA9685 se inicializuje na maximální frekvenci PWM, tedy 1.5kHz.

7.5.4 Zakončení jednoho běhu

Po změření hodnot ze senzorů se odešlou naměřené hodnoty přes MQTT na backend webové aplikace. Tento proces je více popsán v sekci týkající se komunikace 6.4.1. Následně se mikrokontroler uspí na 15 minut. Ovšem pokud by se použila pevná hodnota, tak by docházelo k mírnému posunu s každým během. Proto je od tohoto času odečtena doba běhu mikrokontroleru získána metodou `esp_timer_get_time()`. Před samotným uspáním se ještě nakonfiguruje probuzení pomocí tlačítka určeného k nastavení WiFi a ID zařízení a nastaví se výše zmíněný časovač.

[22] https://play.google.com/store/apps/details?id=com.homestudio.ppfdmeter



Obrázek 7.9: Sestavený květináč

7.6 Webová aplikace

Jak již bylo popsáno v návrhu řešení, webová část používá model klient–server.

Server se skládá z frameworku Node.js a webového frameworku Express, který slouží pro cesty ke koncovým bodům API a obecně práci s požadavky a odpovědmi.

Jak klient, tak i server mají konfigurační soubor `.env`, kde se nachází nastavení tokenů, URL adres databází a komponent (klient, server). Tento soubor je specifický pro každý stroj a z důvodu bezpečnosti je nutné mít pokaždé jiné tokeny.

Pro psaní webové části jsem použil IDE IntelliJ IDEA.

7.6.1 Autentizace

Jelikož se jedná o veřejně dostupnou centrální webovou aplikaci, tak je nutné určit, o jakého uživatele se jedná. Autentizace využívá JSON Web Tokeny (JWT). Jedná se o standard určený pro podepsání datové struktury v JSONu [6]. S implementací jsem se inspiroval dle návodu publikovaného na TowardsDev[23].

Proces probíhá tak, že klient zašle serveru pokus o přihlášení, pokud údaje odpovídají, server vygeneruje token pomocí tajného klíče v `.env` s userID v obsahu a zašle jej klientovi, který si jej uloží v cookie a/nebo zasílá v hlavičce. Tato cookie je podepsána dalším tajným klíčem definovaným v `.env`. Zároveň si klient uloží údaje o uživateli do react kontextu pro použití na jiných místech, třeba v podmínkách na zobrazení obsahu.

Poté musí klient zasílat v hlavičce tento token serveru nebo mít povolené cookies a využít token uložený tímto způsobem, aby se ním mohl prokázat. Pokud je nutno použít

[23] https://towardsdev.com/jwt-authentication-with-node-js-and-react-dc41ef0e6136

hlavičky, tak abych jej nemusel manuálně přidávat do každého požadavku, tak je použita možnost definovat defaultní hlavičku v axiosu jako autorizační Bearer token.

Server poté verifikuje v `authMiddleware` přijatý token pomocí tajného klíče. Pokud je vše validní, tak se do requestu přidá objekt s daty uživatele, které se pak používají na dalších místech. Třeba jako podmínka pro použití API na serveru.

Pro tento účel jsou na klientu komponenty `AuthMiddleware`, což je výše zmíněný react kontext starající se o získání dat uživatele ze serveru. Proto obaluje v `App.js` veškerý obsah. Jeho doplněk je komponenta `AuthRequired`, která zobrazí obsah jen pokud je uživatel přihlášen. Pokud ne, je přesměrován na přihlašovací obrazovku. Stránka, kterou chtěl uživatel vidět je zapamatována a po úspěšném přihlášení je na ni přesměrován.

Na serveru je kromě výše zmíněného authMiddlewaru důležitý `authController`, který se stará o registraci a přihlašování včetně generování a zasílání tokenu. Zde jsou tedy všechny podmínky na hesla, jestli uživatel již neexistuje v databázi a jiné.

7.6.2 Frontend

Frontent je to, co uživatel vidí, proto je nutné zobrazit co nejvíce dat, ale zároveň přehledně, aby nebylo uživatelské prostředí chaotické. Navíc se musí počítat s responzivním designem, aby byl web použitelný i na menším displeji. Neboť mobilní telefony nyní tvoří 60% internetového provozu[24].

Jelikož je React knihovna zaměřená na opětovné použití kódu, je vhodné vytvářet menší komponenty, které pak skládat do větších.

Webová aplikace je jednoduchá, má 4 hlavní obsahové pohledy:

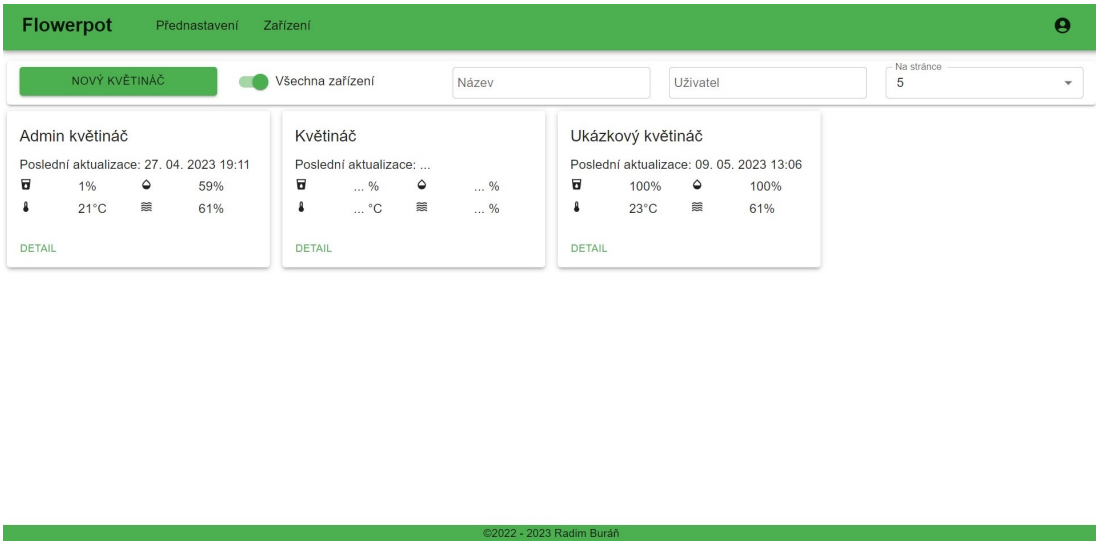
- Přehled přednastavení
- Přehled zařízení
- Detail přednastavení
- Detail zařízení
- Stránku s profilem uživatele

V přehledu přednastavení se zobrazují jen názvy a krátký popis. Výchozí chování je, že se zobrazí přednastavení od všech uživatelů – to lze vypnout přepínačem.

V přehledu zařízení vidí běžný uživatel jen svoje zařízení, ale admin uživatel si může přepnout pohled na všechna zařízení a filtrovat dle jména uživatele (tedy jako u přehledu přednastavení). V náhledu zařízení se zobrazují poslední naměřené hodnoty (Obrázek 7.10).

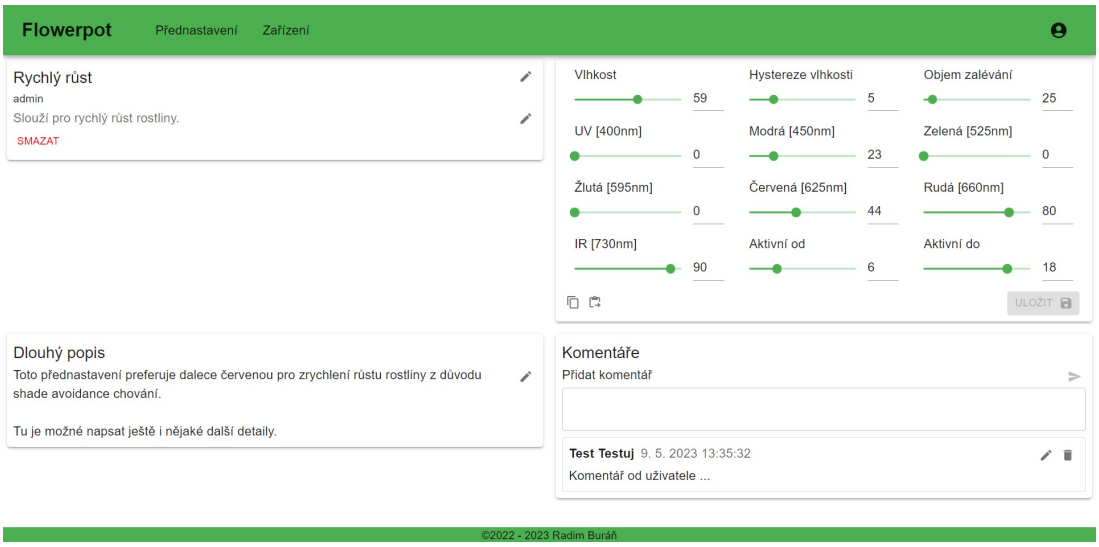
Detail veřejného přednastavení rozdělí stránku do 4 částí – obecné informace, dlouhý popis, přednastavení a komentáře (Obrázek 7.11). V obecných informacích je možnost editovat název a krátký popis, či smazat toto přednastavení. Je zde implementován limit počtu znaků – 50 u nadpisu a 200 u popisu, při překročení je textové pole podbarveno červeně a nelze více psát. Detailní popis má limit 800 znaků. Samotné přednastavení lze měnit manuálně, či využít tlačítka pro zkopírování/vložení. Kopírování funguje globálně na celé webové aplikaci, pro tyto účely se používá cookie `copiedPreset`, do které se uloží všechny proměnné přednastavení až na objem pumpované vody – neboť ta se liší od velikosti květináče. Možnost editovat a ukládat může pouze vlastník přednastavení či admin uživatel. Komentáře může psát kdokoliv. Komentář je zvýrazněn zeleně v případě, že aktuální uživatel jej

[24] https://www.oberlo.com/statistics/mobile-internet-traffic



Obrázek 7.10: Přehled zařízení – pohled administrátora

napsal, či pokud jej napsal admin, tak červeně. Možnost smazat již napsaný komentář může pouze admin uživatel.



Obrázek 7.11: Detail veřejného přednastavení – pohled administrátora

Detail zařízení rozdělí pohled na 3 části – přehled s obecnými informacemi, konfigurovatelné přednastavení a grafy s daty ze senzorů v čase. V přehledu informací je nejdůležitější údaj ID zařízení ve formátu UUID, který se dá zkopírovat a následně se vkládá do formuláře při konfiguraci senzorického uzlu. Dále jsou zde poslední naměřené hodnoty a možnost změnit název zařízení či jej smazat. Přednastavení květináče se chová stejně jako nastavení pod veřejným přednastavením. Pro zobrazení grafů je použita knihovna **react-chartjs-2**[25] se zásuvným modulem pro přibližování. Ovšem nevýhoda této knihovny je, že u dotykového

[25] https://react-chartjs-2.js.org/

displeje se mi nepovedlo implementovat omezovací obdélník pro přibližování/posun grafu jako při práci s myší. Proto je nutné na telefonu scrollovat po krajích displeje.

Na stránce s profilem uživatele je pouze přehled s údaji a možná změna hesla či emailu.

Komponenty, které podporují zobrazení více objektů z databáze (tedy přehled přednastavení či zařízení a komentáře) používají stránkování, aby se nezobrazilo všechno naráz.

7.6.3 Backend

Backend obstarává všechnu práci s daty, jak se zápisy, tak se čtením z databází. Používá komunikaci jak přes REST API s klientem, tak pomocí MQTT se senzorickým uzlem. Toto téma již bylo řešeno v návrhu, konkrétně v kapitole 6.4 věnující se komunikaci v celém systému.

Struktura serveru je ve stylu Models/Routes/Controllers/Services pro přehlednost a rozdělení, co se kde v kódu děje. Datové modely objektů jsou ovšem definovány již pomocí knihovny `Prisma` [16], která komunikuje s MySQL databází. Jedná se o ORM („Object Relation Mapping“), tedy objektově-relační mapování. Slouží pro propojení databázových tabulek s objekty použitými v kódu. Prisma tohle řeší trochu jinak, neboť se definují databázové tabulky i objekty společně v jednom souboru – `schema.prisma`[26]. Proto v projektu není složka „Models“. Zbylé části tohoto typu návrhu jsou:

- `Routes` – mapují endpointy na controllery
- `Controllers` – parsují parametry z dotazů a volají metody v services
- `Services` – vytváří dotazy do databáze a vrací data. Většinou obsahují metodu `getByID()` sloužící k získání jednoho konkrétního objektu, `update()` sloužící ke změně objektu, `deleteByID()` pro smazání, `getByFilter()` pro navrácení více objektů a `createForUser()` určené pro vyvoření pro daného uživatele.

Aplikace začíná v souboru `index.js`, kde se nastavují jednotlivé cesty ke koncovým bodům definovaným v `routes` složce. Zároveň je zde definován `CORS`, což znamená `Cross-Origin Resource Sharing`. To omezuje přístup na API pouze z klienta. Je použita knihovna `cors`[27].

Důležitým prvkem je `authMiddleware`, tento soubor definuje middleware pro zisk JWT tokenu od uživatele. Tento token je pak ověřen, jestli je validní. Pokud ano, tak je přidán datový objekt uživatele do požadavku jdoucího od klienta. Proto pak mohou controllery zjistit, o jakého uživatele se jedná (třeba výpis vlastních zařízení).

Některé „service“ nejsou napojeny na controllers a routes, neboť slouží pro jiný účel. Například `mqttService.js` je vlastně MQTT klientem, který obstarává všechny instance senzorických uzlů. Detailnější popis je v návrhu věnujícímu se komunikaci 6.4.1. Tento klient využívá knihovnu `mqtt`[28]. Zde je tedy implementována i logika zisku dat ze senzorů ze zprávy a jejich následné zapsání do InfluxDB – to je implementováno v `recordsService.js`. Data jsou zapsána pouze v případě, že v relační databázi existuje zařízení s ID odpovídajícímu UUID v přijaté zprávě. Jiným příkladem může být `dbService.js`, která inicializuje připojení do databází a poskytuje metody pro práci s nimi na dalších místech v databázi.

[26] https://www.prisma.io/docs/concepts/overview/prisma-in-your-stack/is-prisma-an-orm
[27] https://expressjs.com/en/resources/middleware/cors.html
[28] https://www.npmjs.com/package/mqtt

Backend implementuje i jednoduché odesílání emailů uživateli (např. při registraci či změně hesla). Toho je docíleno použitím knihovny `Nodemailer` [17], která je nakonfigurována pro použití příkazu `sendmail` – chováním tedy podobně jako funkce `mail()` v PHP[29].

Jak již bylo zmíněno v implementaci frontendu 7.6.2, tak některé jeho komponenty používají stránkování. Pro jejich funkčnost je tedy potřeba podpora na backendu. Této podpory je docíleno vracením ne jen seznamu dat, ale i informací o jejich celkovém počtu dle zadaného filtru. Implementováno je to ve všech metodách `getByFilter()` v souborech v services.

[29] https://nodemailer.com/transports/sendmail/

Kapitola 8

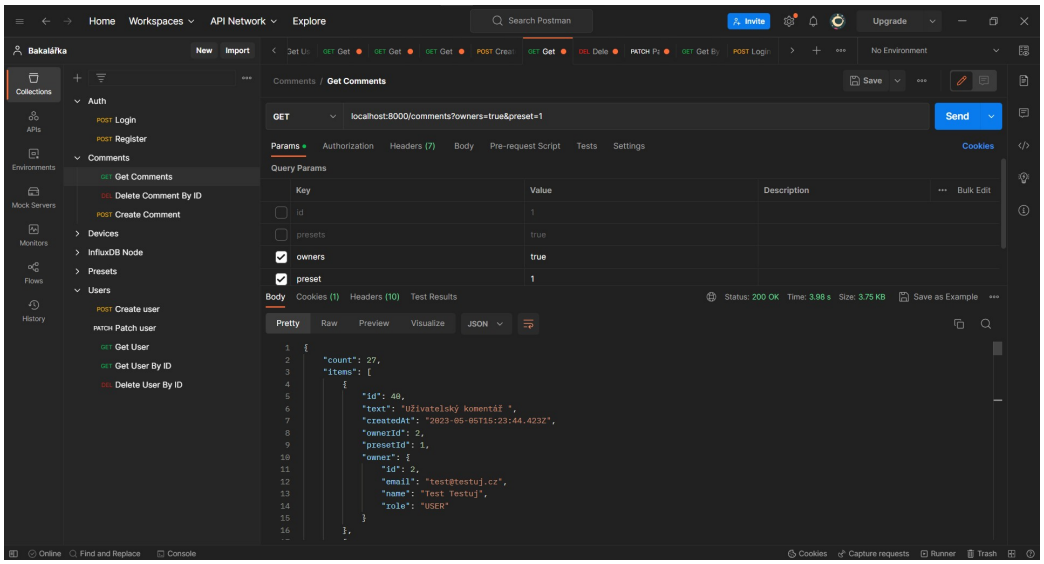
Nasazení

Jelikož je systém navrhnut a implementován tím způsobem, že je nutná webová aplikace, která je ideálně dostupná veřejně na internetu, tak jsem ke zprovoznení využil VPS[1].

Pro spouštění lokálně na svém PC jsem použil program **XAMPP**, který distribuuje mimo jiné Apache včetně MySQL databáze[2] – tyto komponenenty jsem používal při vývoji.

8.1 Testování

Průběh testování lze rozdělit na více částí. Primárně z počátku při psaní API na serverové části webové implementace jsem používal aplikaci **Postman**. Zde lze psát dotazy na API endpointy a analyzovat odpovědi. Jako příklad na obrázku 8.1 je odpověď s daty komentářů pro přednastavení a taky jak velká je odpověď a kolik času trvalo, než přišla.



Obrázek 8.1: Příklad dotazu na komentáře v Postman

[1] Virtual Private Server

[2] https://www.apachefriends.org/

Pro testování klientské části webové aplikace na kritických místech jsem používal javascript debugger a rozšíření do prohlížeče pro react[3]. Díky těmto nástrojům je možné vidět aktuální rozložení react komponent, jejich vlastnosti a proměnné.

Pro vyzkoušení a analýzu MQTT komunikace jsem použil aplikaci **MQTTX**. Jedná se o klienta, který podporuje přihlášení do více témat a taky možnost publikovat zprávy s různými QoS. Všechny zprávy jsou vidět dle času pod sebou.

Jelikož použitá deska nemá přímo v sobě zabudovaný JTAG adaptér pro debugging a ani jsem nepoužíval externí, tak jsem využil debug výpisy do serial monitoru do PC. Pro analýzu pádů z důvodu chyby v kódu zase `exception decoder` v PlatformIO, stačí jej povolit v konfiguračním souboru projektu pro danou desku.

8.2 Zprovoznění

Pro zprovoznění ESP32 je nutné na něj nahrát program pomocí PlatformIO (šlo by i v ESP-IDF, neboť jej využívám, ale PlatformIO se stará o konfiguraci). Použité knihovny jsou již obsaženy ve zdrojových souborech ve složce „lib“. Po nahrání na ESP32 a po zapnutí se spustí WiFi AP s názvem „ESP32“ a heslem „1234567890“, zde se musí nastavit ID zařízení a údaje domácí sítě WiFi k připojení.

Pro zprovoznění webové aplikace je použita jedna instance VPS na které běží vše potřebné, tedy databáze (MySQL a InfluxDB), backend i frontend webové aplikace a pomocné programy. Primárně Apache pro proxyPass, aby se nemusel do URL psát i port frontendu a sendmail pro odesílání emailů.

Jelikož jsem pro účely demonstrace nezakoupil vlastní doménu, tak emaily používají „@example.com“, což je doména určená pro testovací účely[4] – z tohoto důvodu se může stát, že email nepřijde do hlavní složky s poštou, ale do spamu.

Obě instance node.js jsou spravovány pomocí **pm2** – což je manažer procesů k tomuto určený[5]. Jeho obsluha je velmi jednoduchá – pro pro přidání frontendu posloužil příkaz `pm2 start index.js --name "node-api"` (vykonán ve složce „client“) a pro backend `pm2 start --name "node-react" npm -- start` (vykonán ve složce „server“). Následně stačilo povolit automatické spouštění pm2 po restartu VPS a to bylo vše.

```
root@ubuntu:~# pm2 l
```

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user	watching
0	node-api	default	1.0.0	fork	352436	4h	5	online	0%	95.6mb	root	disabled
1	node-react	default	N/A	fork	352451	4h	6	online	0%	70.7mb	root	disabled

Obrázek 8.2: Přehled node aplikací pod pm2

8.3 Pozorování

V této kapitole je probráno pozorování zpětné vazby systému na změny okolního prostředí a následně samotný experiment s rostlinou.

Na následujícím obrázku 8.3 je možné vidět krátký časový úsek po západu slunce s vypnutým svícením LED diodami. Lze vypozorovat postupné snižování teploty, ze světelných

[3] https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi

[4] https://example.com/

[5] https://pm2.keymetrics.io/

dat úbytek světla ve všech spektrech (krátké zvýšení kvůli otočení žaluzií) a následné rozsvícení velkého světla v místnosti. Zároveň je vidět špatná konfigurace ze strany uživatele, neboť bylo nastaveno příliš velké množství vody při jednom zalévání – velký úbytek v nádržce s vodou a přílišné zalití květináče. Následné zvýšení vodní hladiny v nádržce je způsobeno tím, že v případě použité vázy jako nádržky je nutné, aby byla hadička dost dlouhá. Proto po nějaké době z ní voda steče zpátky do nádoby a to se projeví jako zvýšení hladiny.

Pozorování věnovaná reakci čidla na teplotu a vlhkost byla již probrána v implementaci, konkrétně na obrázku 7.7. Za zmínku v této kapitole stojí i testování spektrálního senzoru AS7341 na jednotlivé LED diody na obrázku 7.8.

Obrázek 8.3: Grafy v detailu květináče

8.3.1 Experiment s rostlinou

Pro vyzkoušení, jak funguje implementovaný květináč, jsem použil rukolu setou (Eruca Sativa). Experiment probíhal tak, že byl porovnáván růst bez osvětlení a s osvětlením.

Ve všech případech je osvětlení zapnuto maximálně 10 hodin denně. Neboť při více než 12-ti hodinách by se mohlo stát, že rostlina přejde do stádia pro tvorbu semínek a konce své životnosti [14].

Prvotní nastavení bylo (dle výkonu LED) 50% modrá, 10% červená, 45% rudá a 65% dalece červená. Bohužel AS7341 nezvládne zachytit vlnovou délku 730nm, tak nebylo možné porovnat její intenzitu.

Již 4. dnem byl spozorován náskok rostliny pod umělým osvětlením. Rostliny měly 5cm, mimo osvětlení zhruba 4cm. Proto jsem upravil nastavení LED na 35% modrá, 45% červená, 50% rudá a 15% dalece červená.

Následně bylo spozorováno zvětšení plochy listů 6. dnem růstu vůči rostlině bez LED osvětlení 8.6.

Po 9–ti dnech měla rostlina pod LED větší listy a podobnou délku, jako rostlina, která rostla bez LED osvětlení po dobu jednoho měsíce 8.7.

(a) Bez LED (b) S LED

Obrázek 8.4: 3. den

(a) Bez LED (b) S LED

Obrázek 8.5: 4. den

(a) Bez LED (b) S LED

Obrázek 8.6: 6. den

Obrázek 8.7: Levá: 30 dní bez LED | Pravá: 9 dní s LED

Kapitola 9

Možná rozšíření

Co se týče rozšíření, tak jedno z nejzásadnějších (když pominu přehlednější kabeláž u LED diod) je rozdělení senzorického uzlu na moduly. Díky tomu by mohl mikrokontroler fungovat pouze na LiPo baterii a obsluhovat senzory s vodní pumpou. Osvětlení by byl další modul, který by již vyžadoval externí napájení, které nyní stejně používá. Takto by šlo pokrýt dočasné výpadky proudu a rozdělit produkt do dvou částí v případě komerční výroby na verzi se základní funkcionalitou nebo rozšířenou s osvětlením.

Další možností je implementovat ESP-NOW / ESP-WiFi-Mesh. Díky tomu by mohly jednotlivé květináče komunikovat mezi sebou mimo dosah sítě WiFi. Toto by mělo využití například ve velkých sklenících. V domech je většinou pokrytí sítí WiFi dostatečné, tak zde by byl benefit marginální nebo žádný.

Kapitola 10

Závěr

Cílem této práce bylo implementovat květináč optimalizující růst rostliny. Což dle pozorování, si myslím, že se povedlo, neboť proces růstu rukoly byl znatelně rychlejší. Zároveň byla vytvořena webová aplikace, která umožňuje měnit parametry z cloudu a obsahuje knihovnu přednastavení. Detailní vyhodnocení výsledků je probráno již v sekci 8.3.

Jako přínos práce bych označil celkové řešení obsluhy chytrého květináče, kdy na webové aplikaci jsou definovány přednastavení pro různé rostliny – i od jiných uživatelů, ne jen administrátorů – a možnost modulovat osvětlení co se týče intenzity i vlnové délky podle potřeby rostliny.

Práci samozřejmě provázely i menší problémy – blokující delay() v Arduino IDE, kvůli kterému jsem se rozhodl na přechod pod ESP-IDF, kde byla ovšem absence knihoven pro dva použité senzory, nutnost zvětšení limitu hlavičky pro WebServer, jinak ESP32 padalo nebo rušení senzoru způsobené nepájivým polem. Instalace a zprovoznění InfluxDB na VPS dělalo taky problémy, že bylo nutno několikrát opakovat tento proces.

Možná rozšíření byla již prodiskutována v kapitole 9.

Tato práce mi taktéž dovolila se seznámit s věcmi, ke kterým jsem se dříve moc nedostal – ať už šlo o komunikaci skrze MQTT, hlubší seznámení se s programováním pro mikrokontrolery v ESP-IDF, tvorba modelu pro 3D tiskárnu a obecně informace o růstu rostlin.

Literatura

[1] *FreeRTOS event bits, event groups and event flags* [online]. FreeRTOS [cit. 2023-4-21]. Dostupné z: https://www.freertos.org/FreeRTOS-Event-Groups.html.

[2] *PCA9685 16-channel, 12-bit PWM Fm+ I2C-bus LED controller*. High Tech Campus 60, Eindhoven, Netherlands: NXP Semiconductors, duben 2015 [cit. 2023-4-20]. Dostupné z: https://www.laskakit.cz/user/related_files/pca9685.pdf.

[3] *AS7341 11-Channel Multi-Spectral Digital Sensor* [online]. Tobelbader Strasse 30, Premstaetten, Austria: ams-OSRAM AG, červen 2020, v3.00 [cit. 2023-4-20]. Dostupné z: https://ams.com/as7341.

[4] *Datasheet SHT3x-DIS*. Laubisruetistrasse 50, Stäfa, Switzerland: Sensirion AG, prosinec 2022 [cit. 2023-4-20]. Dostupné z: https://sensirion.com/products/catalog/SHT31-DIS-B/.

[5] *VL53L0X Time-of-Flight ranging sensor*. 39 Chemin du Champ-des-Filles, Geneva, Switzerland: STMicroelectronics, prosinec 2022 [cit. 2023-4-20]. Dostupné z: https://www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html.

[6] *JSON Web Tokens - jwt.io* [online]. Okta, Inc., 2023 [cit. 2023-2-21]. Dostupné z: https://jwt.io/.

[7] Bäumker, E., Zimmermann, D., Schierle, S. a Woias, P. A Novel Approach to Obtain PAR with a Multi-Channel Spectral Microsensor, Suitable for Sensor Node Integration. *Sensors*. Květen 2021, sv. 21, s. 3390, [cit. 2023-1-20]. DOI: 10.3390/s21103390.

[8] Cherlinka, V. *Soil Moisture: How To Measure & Monitor Its Level*. Září 2022 [cit. 2023-1-20]. Dostupné z: https://eos.com/blog/soil-moisture/.

[9] DFRobot. *AS7341 Visible Light Sensor 11 Channels Breakout Wiki - DFRobot* [online]. 2023 [cit. 2023-1-23]. Dostupné z: https://wiki.dfrobot.com/AS7341_Visible_Light_Sensor_SKU_SEN0365.

[10] DFRobot. *FireBeetle Board ESP32E SKU DFR0654 DFRobot* [online]. Březen 2023 [cit. 2022-9-28]. Dostupné z: https://wiki.dfrobot.com/FireBeetle_Board_ESP32_E_SKU_DFR0654.

[11] Dix, P. *Why Time Series Matters for Metrics, Real-Time Analytics and Sensor Data* [online]. 5. vyd. červenec 2021 [cit. 2023-4-21]. Dostupné z: https://www.influxdata.com/time-series-database/#download.

[12] Earl, B. *Adafruit PCA9685 16-Channel Servo Driver* [online]. Říjen 2012 [cit. 2023-2-10]. Dostupné z: https://learn.adafruit.com/16-channel-pwm-servo-driver.

[13] Gupta, L. *What is REST - REST API Tutorial* [online]. Duben 2022 [cit. 2023-1-10]. Dostupné z: https://restfulapi.net/.

[14] Littlewood, N. *How To Grow Arugula Indoors* [online]. Červen 2021 [cit. 2023-2-9]. Dostupné z: https://www.geturbanleaf.com/blogs/leafy-greens/growing-arugula-indoors.

[15] Ouzounis, T., Rosenqvist, E. a Ottosen, C.-O. Spectral Effects of Artificial Light on Plant Physiology and Secondary Metabolism: A Review. *HortScience horts*. Washington, DC: American Society for Horticultural Science. 2015, sv. 50, č. 8, s. 1128 – 1135, [cit. 2022-11-30]. DOI: 10.21273/HORTSCI.50.8.1128. Dostupné z: https://journals.ashs.org/hortsci/view/journals/hortsci/50/8/article-p1128.xml.

[16] Prisma Data, I. *Prisma | Next-generation Node.js and TypeScript ORM* [online]. 2022 [cit. 2022-9-27]. Dostupné z: https://www.prisma.io/.

[17] Reinman, A. *Nodemailer* [online]. Duben 2023 [cit. 2023-4-25]. Dostupné z: https://nodemailer.com/about/.

[18] Ruiz, A. *Comparing soil moisture sensors for smart irrigation systems*. Březen 2021 [cit. 2023-1-20]. Dostupné z: https://www.hackster.io/antonio-ruiz/comparing-soil-moisture-sensors-for-smart-irrigation-systems-caa7aa.

[19] Runkle, E. A closer look at far-red radiation. [online]. Greenhouse Product News. Květen 2016, s. 50, [cit. 2023-4-20]. Dostupné z: https://gpnmag.com/article/a-closer-look-at-far-red-radiation/.

[20] Smith, R. *Compare DHT22, AM2302, SHT31, SHT71, Si7021, BME280.* [online]. Kimberly & Robert's Home Page, únor 2018 [cit. 2023-2-11]. Dostupné z: https://www.kandrsmith.org/RJS/Misc/Hygrometers/calib_many_addsht31.html.

[21] Systems, E. *WiFi & Bluetooth MCUs and AIoT Solutions | Espressif Systems* [online]. 2015 [cit. 2022-11-23]. Dostupné z: https://www.espressif.com/en.

[22] Uss, R. V. et al. *ESP-IDF Components library* [online]. 2019 [cit. 2023-2-9]. Dostupné z: https://esp-idf-lib.readthedocs.io/en/latest/.

[23] Voroney, P. Soils for Horse Pasture Management. In:. Leden 2019, s. 65–79. DOI: 10.1016/B978-0-12-812919-7.00004-4. ISBN 9780128129197.