



STŘEDNÍ PRŮMYSLOVÁ ŠKOLA  
OBCHODNÍ AKADEMIE  
JAZYKOVÁ ŠKOLA  
FRÝDEK-MÍSTEK

Příspěvková organizace  
Moravskoslezského kraje



## MATURITNÍ ZKOUŠKA

***PRAKTICKÁ ZKOUŠKA Z ODBORNÝCH PŘEDMĚTŮ***

**Zavlažovací systém PlantHub**

**Téma č. 4**

Obor vzdělání: **18 – 20 – M/01 Informační technologie**

**Informační technologie**

Třída: **4. IT** Autor práce: **Jakub Vantuch, Filip Sikora**

Školní rok: **2021/22** Vedoucí učitel práce: **Ing. Jiří Sumbal**

Oponent práce: **Ing. Beňová Jiřina**



## **Prohlášení autora**

„Prohlašujeme, že jsme tuto práci vypracovali samostatně a použili jsme literárních pramenů a informací, které citujeme a uvádíme v seznamu použité literatury a dalších zdrojů informací.“

Ve Frýdku-Místku, dne: ..... podpis

Ve Frýdku-Místku, dne: ..... podpis

## **Zadání práce**

### **Jakub Vantuch**

- Hardwarové řešení - stanovení cílů, volba hardware (řídící jednotka, senzory, akční členy)
- Návrh obvodu a plošného spoje
- Fyzická realizace
- Naprogramování řídící jednotky
- Interaktivní ovládání PlantHubu z webového UI
- Odladění, ověření funkčnosti

### **Filip Sikora**

- Softwarové řešení - stanovení cílů, volba sw platformy a konkrétního software
- Konfigurace RPi jako webového serveru
- Vytvoření databáze
- Vytvoření front end
- Vytvoření back end
- Nastavení bezpečnosti
- Interaktivní ovládání PlantHubu z webového UI
- Odladění, ověření funkčnosti

## Anotace

PlantHub je zavlažovací systém, s plánovaným a automatickým způsobem zavlažování. Nastavení celého systému je možno upravit přes webové uživatelské rozhraní (WUI), které navíc nabízí i dashboard a prostředí pro manuální ovládání čerpadla. Jádrem našeho systému je mikropočítač Raspberry Pi (RPi), který slouží jako hostovací zařízení pro webový server, databázi, a zároveň jako řídící jednotka celého našeho systému. Systém PlantHub dále získává informace o vlhkosti půdy, teplotě, vlhkosti vzduchu, hladině vody a vykresluje je ve svém WUI. Ve stejné chvíli naměřená data ukládá do databáze v periodě 4 hodin. Jelikož voda časem v nádrži dojde, systém PlantHub snímá stav hladiny vody v nádrži a včas upozorní, že je třeba ji doplnit.

## Klíčová slova

zavlažování; automatizace; statistika; mikropočítač; uživatelské rozhraní

# **Obsah**

<b>1</b>	<b>Úvod</b>	<b>7</b>
<b>2</b>	<b>Seznam zkratek a pojmu</b>	<b>8</b>
<b>3</b>	<b>Hardware</b>	<b>10</b>
3.1	Senzory . . . . .	10
3.1.1	Senzor teploty a vlhkosti vzduchu DHT11 . . . . .	10
3.1.2	Senzor vlhkosti půdy . . . . .	10
3.1.3	Analogově digitální převodník MCP3008 . . . . .	10
3.1.4	Ultrasonický senzor HC-SR04 . . . . .	12
3.1.5	Čerpadlo . . . . .	12
3.1.6	Tranzistor 2N2222 . . . . .	13
3.2	Raspberry Pi . . . . .	13
3.2.1	Architektura ARM . . . . .	13
3.3	Router . . . . .	13
<b>4</b>	<b>Návrh obvodu a plošného spoje</b>	<b>14</b>
4.1	Testovací verze . . . . .	14
4.2	Finální verze . . . . .	15
<b>5</b>	<b>Fyzická realizace</b>	<b>16</b>
5.1	Pouzdro . . . . .	16
5.2	Hub . . . . .	17
5.3	Nádrž . . . . .	18
5.4	Celý systém . . . . .	19
<b>6</b>	<b>Hlavní program</b>	<b>20</b>
6.1	Postup práce . . . . .	20
6.2	Fáze programu . . . . .	20
6.2.1	Měření . . . . .	21
6.2.2	Periodické ukládání dat . . . . .	21

6.2.3	Controller . . . . .	21
6.2.4	Inicializace . . . . .	21
6.2.5	Zavlažování . . . . .	22
6.2.6	Kontrola . . . . .	22
6.2.7	Ukládání dat . . . . .	22
6.3	Programovací jazyk Go . . . . .	22
6.3.1	Paralelní programování v Go . . . . .	23
6.3.2	Typový systém . . . . .	23
<b>7</b>	<b>WUI</b>	<b>24</b>
7.1	Programování WUI . . . . .	24
7.2	Inicializační formulář . . . . .	24
7.3	Dashboard . . . . .	25
7.4	Manuální ovládání . . . . .	26
7.5	Nastavení . . . . .	27
7.6	Notifikace . . . . .	28
<b>8</b>	<b>Databáze</b>	<b>29</b>
8.1	SQL . . . . .	29
8.2	GraphQL . . . . .	29
<b>9</b>	<b>Webový server</b>	<b>30</b>
<b>10</b>	<b>Docker a kontejnerizace</b>	<b>31</b>
10.1	Výhody kontejnerizace . . . . .	31
10.2	Kontejnery vs. virtuální počítače . . . . .	32
<b>11</b>	<b>Bezpečnost</b>	<b>33</b>
<b>12</b>	<b>Závěr</b>	<b>34</b>
<b>13</b>	<b>Seznam obrázků</b>	<b>35</b>
<b>14</b>	<b>Reference</b>	<b>36</b>

<b>15 Seznam použitého softwaru</b>	<b>37</b>
<b>16 Seznam příloh</b>	<b>38</b>

## 1 Úvod

Naším cílem je návrh, sestavení a naprogramování automatického zavlažovacího systému s názvem PlantHub. Systém se skládá z řídící jednotky (stanice), senzorů, čerpadla, nádrže a WUI. Tato stanice pravidelně snímá data ze senzorů měřících teplotu a vlhkost vzduchu, vlhkost půdy a stav hladiny v nádrži, ze které stanice přečerpává vodu. Naměřená data se živě odesílají a zobrazují ve WUI, ve stejné chvíli se samostatně, v periodě čtyř hodin ukládají do databáze. Přístup k naměřeným datům a WUI má pouze uživatel lokální sítě, do které je PlantHub připojen.

WUI je hostované na naší stanici. Uživatel má možnost zobrazení statistik jak živě naměřených dat, tak dat historických. WUI také nabízí manuální kontrolu nad čerpadlem, nařízením WUI a samotné řídící jednotky.

Jednotlivé součásti a moduly stanice jsme vybrali podle finanční dostupnosti a adekvátních technických požadavků na přesnost měření. Obvod jsme v testovací verzi postavili na nepájivém poli a ve finální verzi navrhli a objednali vlastní plošný spoj (PCB). Pro ochranu naší stanice jsme navrhli kryt, který jsme následně vytiskli na 3D tiskárně.

Hlavní program jsme napsali v programovacím jazyce Go, jehož hlavní využití je vytváření backendů webových aplikací. Tento jazyk se nám zalíbil natolik, že jsme se v něm rozhodli vytvořit jak application programming interface (API), databázové funkce, tak i hlavní program. Program je rozdělen na několik sekvencí, z toho hlavní jsou měřící sekvence a sekvence controlleru, která pomocí snímání dat z databáze určuje, zda se jedná o první spuštění PlantHubu. Na základě toho určí, jestli se spustí incializační sekvence nebo sekvence samotného zavlažování.

- Kapitoly 1, 2, 12, 13, 14, 15, 16 jsme zpracovali společně.
- Kapitoly 3, 4, 5, 6 zpracoval Jakub Vantuch.
- Kapitoly 7, 8, 9, 10, 11 zpracoval Filip Sikora.

## 2 Seznam zkratek a pojmu

**WUI** webové uživatelské rozhraní

Vysvětleno v kapitole [7.](#)

**API** application programming interface

Spojení pro přenos dat mezi zařízeními nebo programy, v našem případě, pomocí http protokolu.

**REST** representaion programming interface

Definuje strukturu [API](#) tak, že zobrazuje všechna data a vytváří k nim jednoduchý přístup. Protože, ale nenačítá pouze vybraná data, není tento proces vždy vhodný, jelikož může v mnoha případech zpomalovat celou aplikaci.

**GraphQL** graph query language

Vysvětleno v kapitole [8.2.](#)

**PCB** plošný spoj

Předem připravené pájecí pole, vytvořené tak aby přesně odpovídalo danému obvodu.

**RPi** Raspberry Pi

Mikropočítač s podporou IoT. Více v kapitole [3.2.](#)

**ADC** analogově digitální převodník

Vysvětleno v kapitole [3.1.3.](#)

**DHT11** digital humidity temperature v.11

Senzor pro čtení hodnot teploty a vlhkosti vzduchu.

**HC-SR04** Ultrasonický senzor vzdálenosti

Vysvětleno v kapitole [3.1.4.](#)

**LED** light emitting diode

Polovodičová součástka vyzařující světlo.

**DHCP** dynamic host configuration protocol

Protokol pro automatické přiřazování IP adres v dané síti.

**SAR** successive-approximation

Metoda převodu analogového vstupu na digitální. Více v kapitole [3.1.3.](#)

**MSB** most significant bit

Vysvětleno v kapitole [3.1.3.](#)

**LSB** least significant bit

Vysvětleno v kapitole [3.1.3.](#)

$V_{ref}$  Referenční napětí

Vysvětleno v kapitole [3.1.3.](#)

$V_{in}$  Vstupní napětí

Vysvětleno v kapitole [3.1.3.](#)

$V_{dac}$  Napětí převedené z digitální do analogové hodnoty

Vysvětleno v kapitole [3.1.3.](#)

**IoT** Internet of things

Vysvětleno v kapitole [11.](#)

## 3 Hardware

### 3.1 Senzory

#### 3.1.1 Senzor teploty a vlhkosti vzduchu DHT11

Senzor digital humidity temperature v.11 (DHT11) se skládá z jednotky pro měření teploty, jednotky pro měření vlhkosti vzduchu a převodníku. Teplotu měří senzor termistorem. Termistor je keramický polovodič, který zmenšuje svou rezistivitu, když se okolní teplota zvýší. Vlhkost měří senzor na základě rezistivity substrátu, umístěného mezi dvěma elektrodami. Tento substrát zachytává vlhkost a vytváří tak vodivé prostředí.

#### 3.1.2 Senzor vlhkosti půdy

Jedná se o kapacitní senzor, který se skládá ze dvou vodivých desek a převodníku. Čidlo funguje na způsob kapacitoru avšak jeho kapacita je ovlivněna vlhkostí, která ovlivňuje dielektrikum mezi dvěma deskami.

#### 3.1.3 Analogově digitální převodník MCP3008

RPi má v rámci general-purpose input/output (GPIO) pinů pouze digitální vstupy, protože je ale senzor vlhkosti půdy analogový, museli jsme použít analogově digitální převodník (ADC).

Pomocí ADC v integrovaném obvodu s 10-bitovým rozlišením tedy přemapujeme analogový signál do deseti různých digitálních hodnot, kterým se také říká bitové rozlišení. ADC měří hodnoty od 0 až po 1023 a následně je pomocí Serial Peripheral Interface (SPI) komunikace posílá do RPi. RPi metodou successive-approximation (SAR) definuje adresu v binární formě, srozumitelné pro digitální vstup. V první iteraci převodu se na SAR registru nastaví první bit na logickou hodnotu 1, stane se z něj most significant bit (MSB), ten určuje pozici výstupu srovnávání dané iterace. Je nutné mít správné referenční napětí, to totiž určuje nejvyšší možnou binární adresu. Adresa v první iteraci teda bude polovinou nejvyšší možné binární adresy + 1. V další iteraci je první bit least significant bit (LSB), to je ten bit, který se v dané iteraci používá pro výstup srovnávání a druhý bit se stává MSB. Srovnává se hodnota zpětné konverze binární adresy s hodnotou analogového vstupu, pokud je větší, na MSB se

nastaví logická hodnota 1, pokud menší tak logická hodnota 0. Podle počtu bitů ADC se určují iterace a přesnost převedené hodnoty.

Příklad funkcionality, pro jednoduchost se 4-bitovým ADC.

Referenční napětí ( $V_{ref}$ ) = 16V

Vstupní napětí ( $V_{in}$ ) = 11V

### 1. iterace

Napětí převedené z digitální do analogové hodnoty ( $V_{dac}$ ) =  $\frac{1}{2}V_{ref}$

(MSB)			
1	0	0	0

### 2. iterace

$V_{dac} = 8$

$V_{in} > V_{dac} \Rightarrow \underline{\text{LSB}} = 1; \underline{\text{MSB}} = 1$

(LSB)	(MSB)		
1	1	0	0

### 3. iterace

$V_{dac} = 12$

$V_{in} < V_{dac} \Rightarrow \underline{\text{LSB}} = 0; \underline{\text{MSB}} = 1$

	(LSB)	(MSB)	
1	0	1	0

### 4. iterace

$V_{dac} = 10$

$V_{in} > V_{dac} \Rightarrow \underline{\text{LSB}} = 1; \underline{\text{MSB}} = 1$

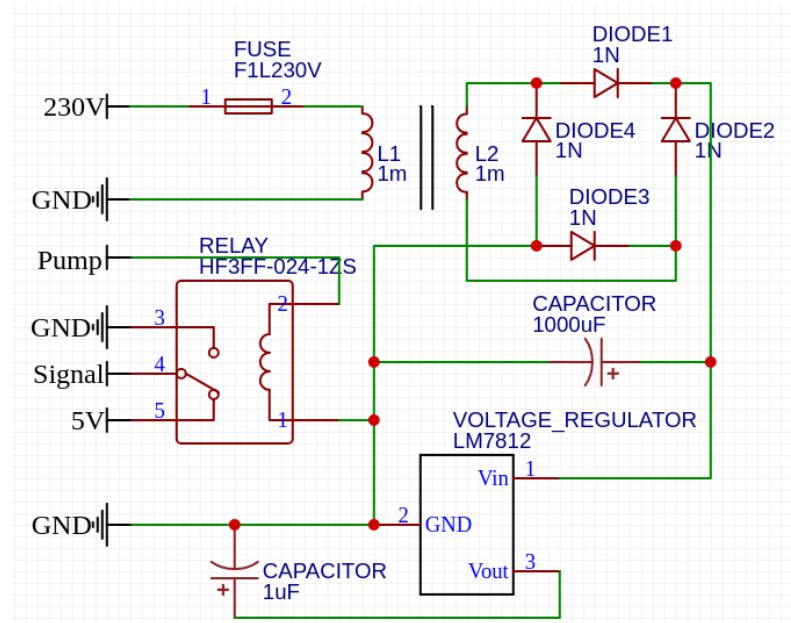
		(LSB)	(MSB)
1	0	1	1

### 3.1.4 Ultrasonický senzor HC-SR04

Ultrasonický senzor vzdálenosti (HC-SR04) vydává zvukové vibrace na vysoké frekvenci, neslyšitelné pro lidské ucho. Poté čeká, až se zvuk odrazí zpět od překážky a vypočítá vzdálenost, na základě času měřeného od vysílání zvukové vlny k zpětnému přijmutí. Z důvodu špatného a nétolik přesného převodu hodnot při použití kombinace RPi a Go jsme byli nuceni k tomuto senzoru připojit Arduino Nano jako prostředníka pro správný převod hodnot. To jsme poté pomocí USB portu připojili k RPi.

### 3.1.5 Čerpadlo

Námi původně zvolené ponorné mini čerpadlo se skládá z DC motoru, na němž je upevněna centrifuga pro čerpání vody a vlastního pouzdra, z kterého vede otvor pro napojení odtokové hadičky. Čerpadlo je připojeno na zdroj napětí a jeho kostra je uzemněna. Bohužel v procesu testování se nám pokazilo čerpadlo, takže jsme museli vymyslet alternativní cestu, která se nakonec ukázala tou lepší variantou. Koupili jsme nové silnější čerpadlo, fungující na stejném principu, připojili k němu samostatný zdroj a přidali relé na jeho spouštění, jako je ukázáno v schématu (Obr. 1), které jsme vytvořili v nástroji EasyEDA [11]. Zdroj RPi jsme použili jako signální pin pro spínání relé a vše jsme poskládali do samostatného krytu.



Obr. 1: Schéma obvodu čerpadla

### **3.1.6 Tranzistor 2N2222**

Protože samotný signální pin neposkytuje dostatečné napětí pro chod čerpadla, ovládáme jej tranzistorem 2N2222. Při výměně čerpadla za jiné, které je spouštěno samostatným relé, jde napětí stále přes tranzistor. Tento tranzistor je bipolární Negative-Positive-Negative tranzistor, to znamená, že jeho polarita je nastavená tak, aby na kolektoru přijímal pozitivní napětí. Díky našemu vyměnitelnému připojení modulů, je možné čerpadlo vyměnit za jiné a původní kabel používat jako spouštěč čerpadla jakéhokoliv výkonu a nároků na zdroj. Při používání čerpadla na 5V stačí dva vstupní kontakty + a -, při použití čerpadla se samostatným relé stačí pouze vyměnit kabel za ten se třemi vstupy +, -, a Signal, v tomto případě bude Signal na stejném pinu jako +, pouze přidáme původní napětí.

## **3.2 Raspberry Pi**

Vybrali jsme si jej, protože kombinuje malou velikost a vyšší výpočetní sílu, než Arduino. Musí totiž zvládnout řídit všechny senzory, ukládat data do databáze a zároveň hostuje i samotnou webovou aplikaci.

### **3.2.1 Architektura ARM**

Advanced RISC Machines (ARM) je založen na Reduced Instruction Set Computing (RISC) a je to nejnověji používaná CPU architektura. Tyto procesory jsou designované na všechny moderní chytré telefony, zařízení s operačním systémem Android i Apple produkty.

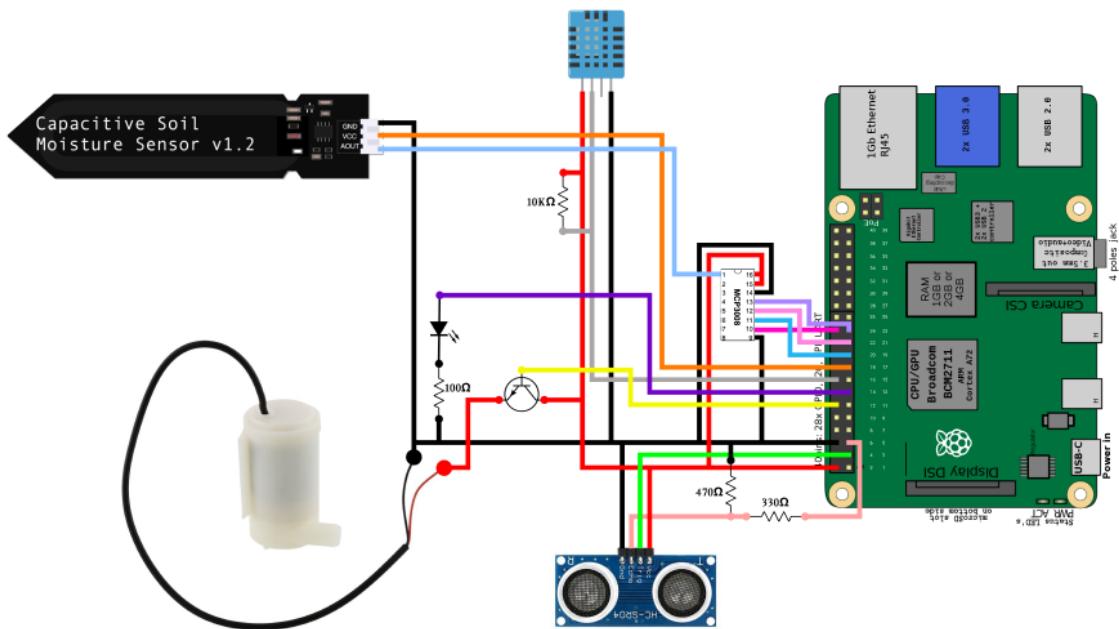
## **3.3 Router**

Připojení RPi do lokální sítě je provedeno pomocí ethernet kabelu do patřičného routeru s dynamic host configuration protocol (DHCP) mechanismem. Klientský počítač připojený do stejné sítě, si poté může jednoduše zobrazit WUI hostované na samotném RPi

## 4 Návrh obvodu a plošného spoje

### 4.1 Testovací verze

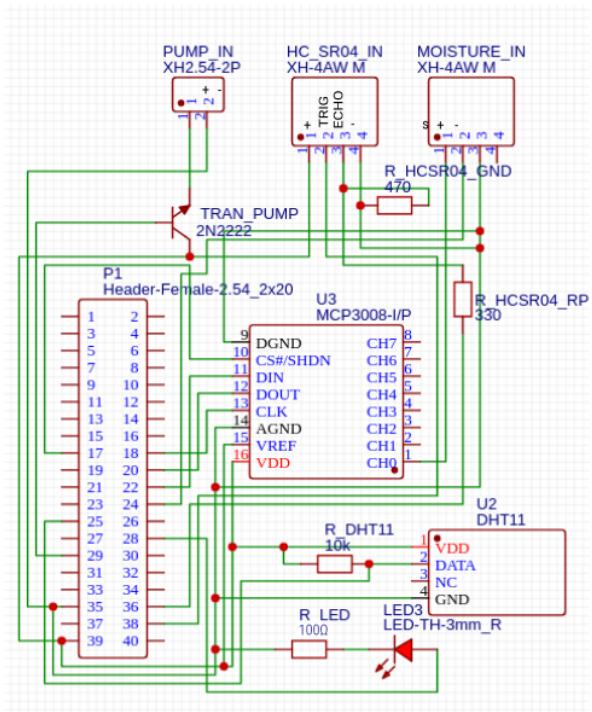
V prvním kroku jsme si vytvořili grafické znázornění celého obvodu (Obr. 2) v nástroji figma [9], Testovací verzi našeho obvodu jsme postavili na nepájivém kontaktním poli. Po úspěšném otestování a ověření funkčnosti všech senzorů jsme přešli na profesionálnější řešení.



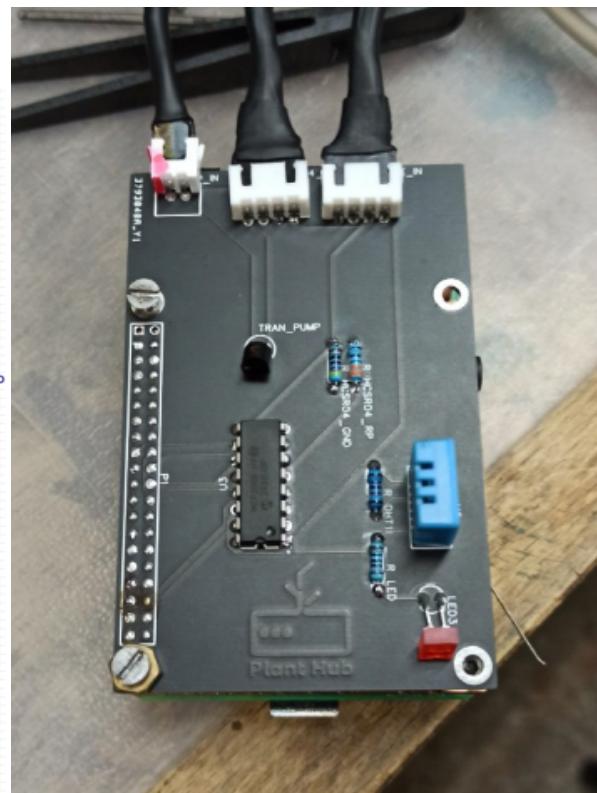
Obr. 2: Grafické znázornění obvodu

## 4.2 Finální verze

Ve finální verzi jsme navrhli naše vlastní schéma hlavního obvodu (Obr. 3). Toto schéma jsme si nechali vytisknout společností JLCPCB a po doručení jsme na něj napájeli patřičné komponenty (Obr. 4).



Obr. 3: Schéma obvodu



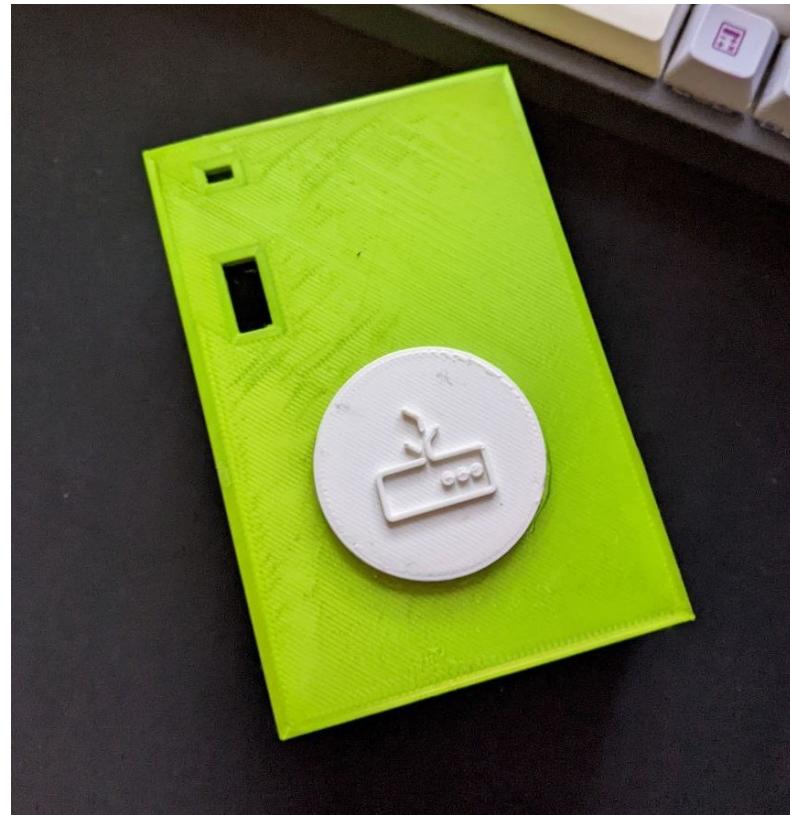
Obr. 4: Finální verze PCB se senzory

## 5 Fyzická realizace

### 5.1 Pouzdro

Navrhli jsme vhodné pouzdro (Obr. 5) pro naše RPi, abychom ochránili citlivé elektronické součástky a zároveň měli možnost jednoduše vyjmout stanici z krytu. Pouzdro jsme vytiskli na školní 3D tiskárně pevným Polyethylene terephthalate glycol (PETG) filamentem.

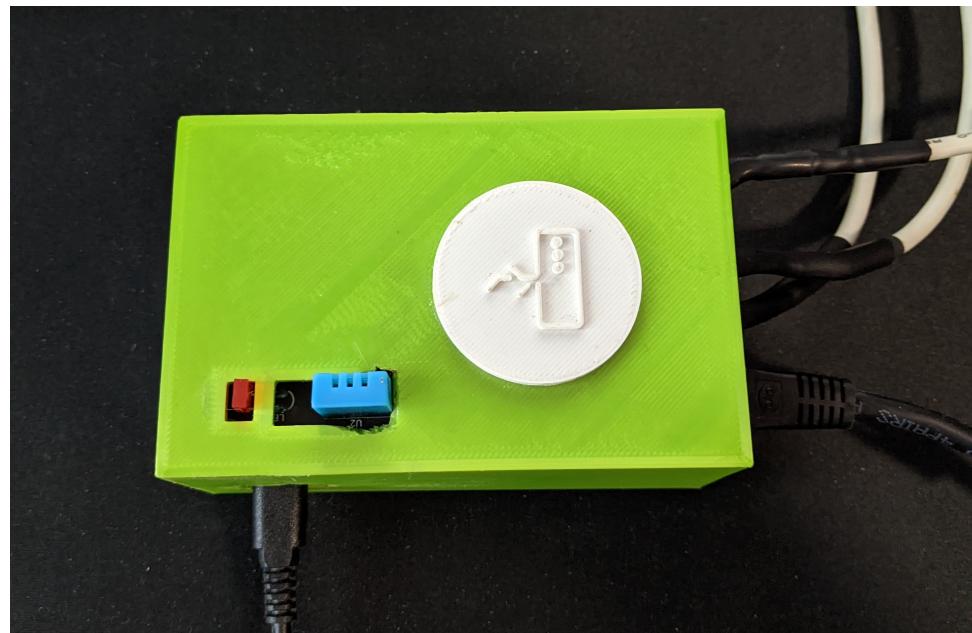
Při první verzi našeho pouzdra jsme nepřidali dostatečnou toleranci pro SD kartu, která přesahuje hranice PCB o několik milimetrů. Omámeni nadšením z první hmatatelné verze našeho pouzdra, jsme se neopatrнě a rychle pokusili, napasovat pouzdro na RPi. Naší neopatrnosti se nám podařilo zmiňovanou SD kartu zlomit. Za odměnu jsme si tedy znova procvičili instalaci a konfiguraci operačního systému RPi.



Obr. 5: Pouzdro

## 5.2 Hub

Zkompletovaný hub se tedy skládá z pouzdra, RPi, PCB a připojených senzorů, ethernet kabelu a zdrojového kabelu. Jednoduchým způsobem je možno hub rozebrat pro případnou opravu.



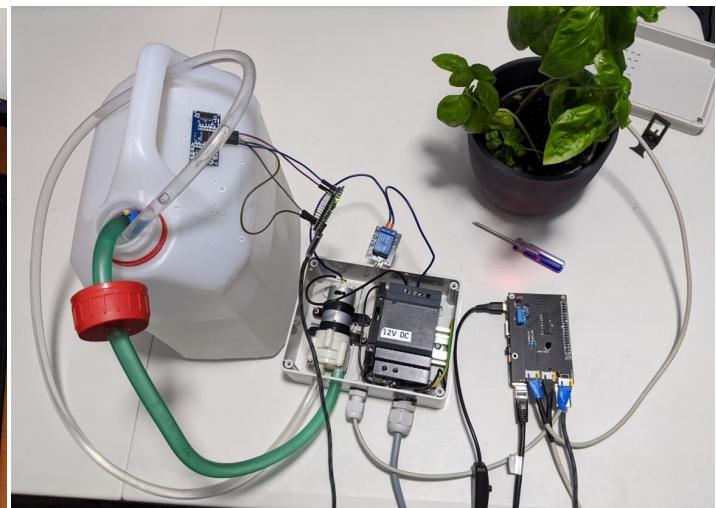
Obr. 6: Hub

### 5.3 Nádrž

K zavlažování bylo potřeba postavit nádrž na vodu, ze které bude naše čerpadlo přečerpávat vodu k zavlažování. Nejvhodnějším řešením je nádoba s co největším objemem, která bude jednoduše přenosná, odolná a nebude složité připevnit na ní senzory. Naše první verze nádoby z upraveného 5L pivního soudku, nebyla totik úspěšná. I přesto, že jsme ji natřeli antikorozní barvou tak začala na několika místech rezivět a z důvodu špatného upevnění čerpadla i protékat. Poté co se nám pokazilo původní čerpadlo jsme byli plně rozhodnuti pro změnu nádrže, kterou používáme doted'. Nádrž má stejný objem jako ta původní, tedy 5L, ale je lépe přenosná a odolnější, což přispívá její životnosti. Přívodová hadička je do ní vedena víčkem a vystužena polyvinyl chloride (PVC) trubkou aby se nekroutila a dosáhla až na dno.



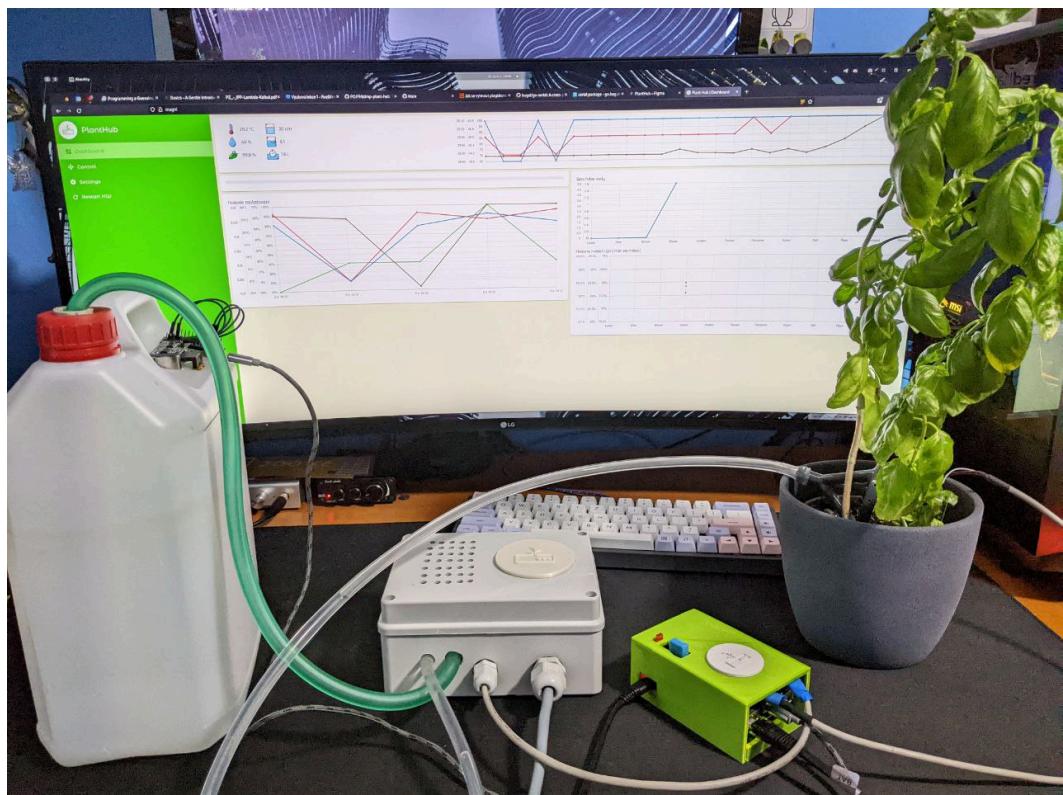
Obr. 7: První verze nádrže



Obr. 8: Finální verze nádrže

## 5.4 Celý systém

Celý systém se tedy skládá z Hubu, nádrže, rostliny, routeru a klientského zařízení. Se-stavení trvá asi 5 minut, protože musíme počkat, než se načte OS RPi a následně zapojit všechny modulární části našeho systému.

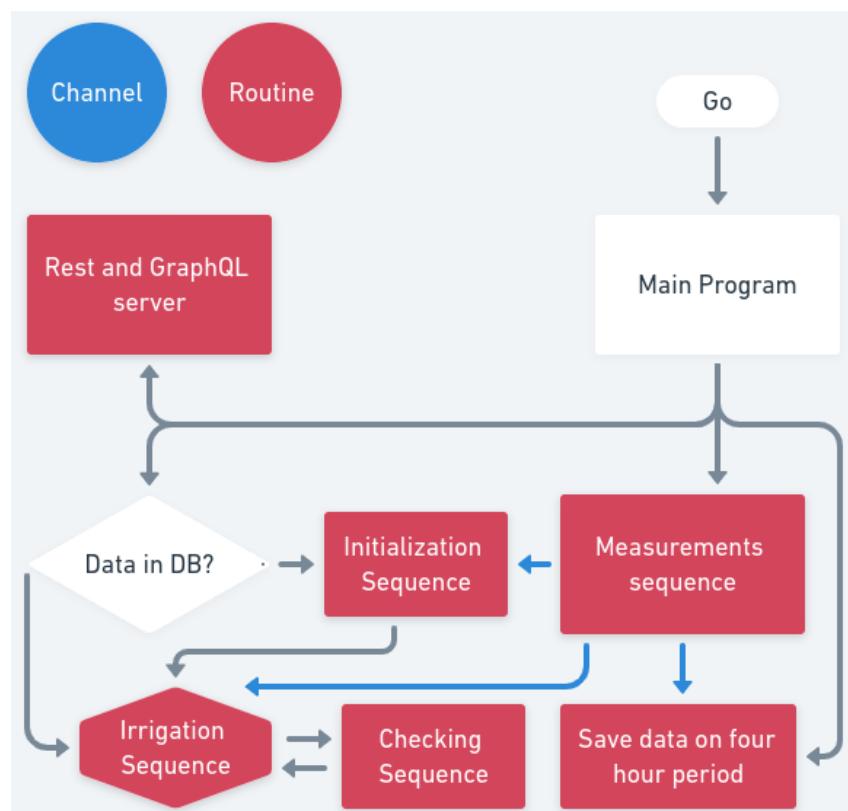


Obr. 9: Celý systém

## 6 Hlavní program

### 6.1 Postup práce

Náš hlavní program pro zavlažování, komunikaci s databází a WUI jsme začali psát ve vysokoúrovňovém programovacím jazyce Python. Od toho jsme nakonec upustili, kvůli horšímu výkonu. Program jsme přepsali v programovacím jazyku Go.



Obr. 10: Vývojový diagram programu

### 6.2 Fáze programu

Fáze programu se spouštějí buďto v samostatné go rutině a komunikují spolu pomocí kanálů, nebo na základě podmínek, kde se po splnění požadavků ukončí.

### **6.2.1 Měření**

Senzor vlhkosti půdy a DHT11 průběžně posílají naměřená data do WUI. Hlavní program ve stejné chvíli čte data ze senzorů a kontroluje jestli naměřené hodnoty nepřekračují limitní hodnoty vlhkosti půdy pro spuštění zavlažování, a ve stejné chvíli kontroluje jestli není čas spustit čerpadlo podle plánovaného zavlažování. Pokud ano, RPi pošle signál pro otevření tranzistoru, což spustí čerpadlo.

### **6.2.2 Periodické ukládání dat**

V samostatné rutině běží funkce pro ukládání naměřených dat v periodě 4 hodin. Tato rutina, stejně jako ostatní, čte hodnoty senzorů z měřící sekvence. Naměřená data jsou následně statisticky zobrazena ve WUI.

### **6.2.3 Controller**

Po spuštění měřící sekvence a sekvence periodického ukládání dat, se spustí buďto inicializační sekvence, nebo zavlažovací sekvence. Podle toho, jestli se v databázi nachází předchozí nastavení. Pokud předchozí nastavení nejsou nalezena, program čeká na uložení dat z WUI a light emitting diode (LED) bliká dvakrát po sobě, dokud data nejsou dostupná. Pokud jsou data k dispozici, spustí se zavlažovací sekvence a načtou se limitní data z databáze. V periodě jedné sekundy se budou číst data o vlhkosti půdy.

### **6.2.4 Inicializace**

Půda musí být ze začátku suchá. Senzor vlhkosti půdy zasuneme co nejhлouběji do půdy. RPi bude chvíli sbírat data a pak je zprůměruje do hodnoty, která bude sloužit jako limit pro spuštění čerpadla.

Ve WUI jde navíc ještě manuálně nastavit hranice vlhkosti půdy pro spuštění čerpadla.

Nastavit se dá také množství vody, které bude přečerpáno při jednom spuštění a jaká je hranice pro přijatelnou výšku hladiny vody v nádrži. Pokud nejsou tyto hodnoty uvedeny, čerpadlo bude vodu přečerpávat, dokud se nezmění hodnota kapacitního čidla pro měření vlhkosti půdy a HC-SR04 použije výchozí nastavení.

### **6.2.5 Zavlažování**

Čerpadlo začne čerpat vodu a zavlažovat rostlinu. Voda se čerpá tak dlouho, dokud senzor vlhkosti půdy nezmění svou hodnotu, nebo dokud není vyčerpán limit přečerpané vody na jedno spuštění.

### **6.2.6 Kontrola**

Po ukončení přečerpávání se spustí HC-SR04 a změří výšku hladiny vody. Naměřená data poté odešle do RPi, kde se uloží do databáze. Pokud bude naměřená hodnota nižší, než je limitní hodnota, začne blikat LED a RPi odešle upozornění o doplnění nádrže do WUI. Jakmile bude hladina vody doplněna, signalizace se vypne.

### **6.2.7 Ukládání dat**

Náš systém ukládá zvlášť periodicky naměřená data a data naměřená před zavlažováním. Dále ukládá nastavení jak pro limity k zavlažování, tak pro WUI.

## **6.3 Programovací jazyk Go**

Go je open-source programovací jazyk, který byl vytvořen společností Google v roce 2009. Je to stále relativně nový programovací jazyk, který se nevyvinul z jiných jazyků jako C# a Java. Go ignoruje teorii o programovacích jazycích. Místo toho, aby se zaměřoval na akademické teorie, zaměřuje se na reálné praktiky, používané pro vývoj next-gen v clouдовých, distribuovaných a souběžných aplikacích.

Jazyk Go je staticky typovaný a využívá garbage collector (automatický správce paměti). Jedná se o komplikovaný programovací jazyk, který své binární soubory kompiluje pro každou platformu. Dá se zařadit do skupiny C jazyků, podle jeho základní syntaxe. Go poskytuje expresivní syntax s jednoduchým typovým systémem a má vestavěné nástroje pro paralelní programování. Výkon Go je srovnatelný s jazyky C a C++, ale zároveň nabízí rychlý a jednoduchý vývoj aplikací.

Stejně jako C a C++ se Go kompiluje do nativního strojového kódu, takže nepotřebujeme běhové prostředí jako Common Language Runtime (CLR) a Java Virtual Machine (JVM). To má řadu výhod, třeba při distribuci aplikace v aplikačních kontejnerech, jako je Docker.

### **6.3.1 Paralelní programování v Go**

Vývoj počítačů se v průběhu posledního desetiletí značně posunul. V minulosti aplikace běžely na počítačích pouze s jediným procesorovým jádrem. Dnes jsou standartně k vidění čtyřjádrové procesory v uživatelských počítačích, dokonce i naše [RPi](#) má dvě jádra. Stále používáme programovací jazyky a technologie navržené v éře, kdy byly dostupné pouze jednojádrové procesory.

Většina programovacích jazyků dnes nabízí knihovny, nebo frameworky pro paralelní programování, ale nemají tuto vlastnost zabudovanou přímo v jádře jazyka. V Go je paralelní programování součástí jazyka už od samotného počátku. Používá takzvané Gorutiny, které umožňují spouštět funkce souběžně. Souběžné funkce pak mezi sebou mohou komunikovat a předávat data pomocí kanálů. Dokonce i některé standardní knihovny jazyka mají zabudovanou souběžnost. Například standardní knihovna ‘net/http’ pro programování HTTP, zpracovává přicházející požadavky souběžně pomocí Gorutin.

### **6.3.2 Typový systém**

Pragmatický design Go neobsahuje klíčové slovo pro třídu a jeho objektová orientace je odlišná od tradičních, objektově orientovaných programovacích jazyků. V Go nahrazuje funkci tradiční třídy typ struct. Dědění není v go podporováno, ale podporuje kompozici typů (Příloha 1).

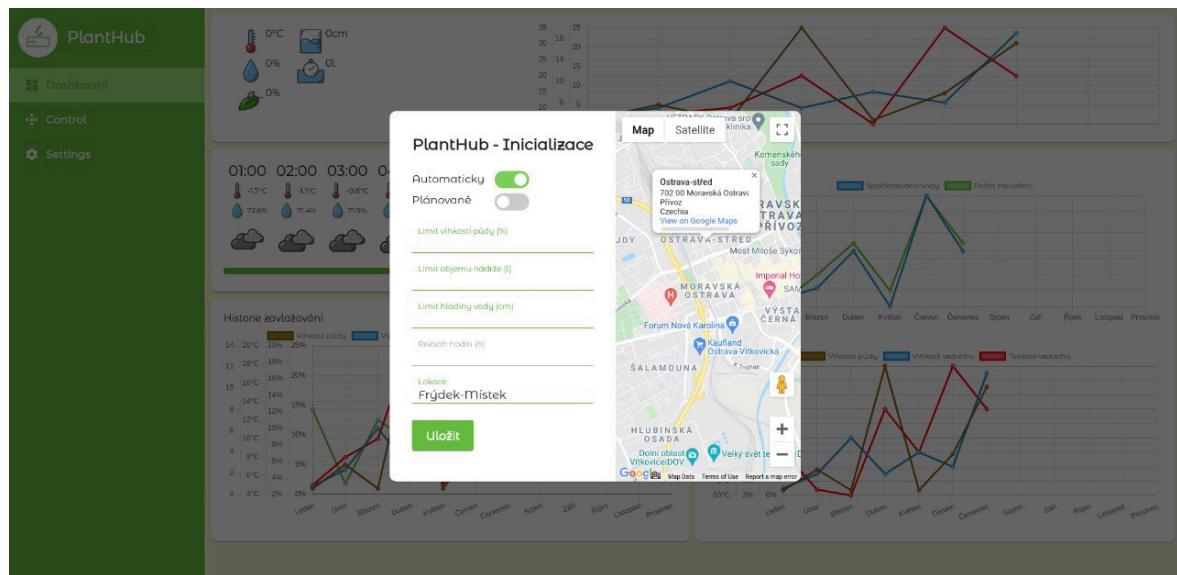
## 7 WUI

### 7.1 Programování WUI

Webovou aplikaci jsme napsali pomocí javascriptového frameworku React.js, CSS frameworku Tailwind a programovacího jazyku Typescript, který je dialektem javascriptu, podporující volitelné statické typování. JavaScript je dynamicky typovaný jazyk. Statické typování je výhodné hlavně u větších aplikací, kde zjednodušuje odchyťávání chyb v oblasti datových typů, nebo jim předchází úplně.

Ve webovém rozhraní je možné zobrazit si statisky, jak živě naměřených dat, tak dat uložených v databázi. Z OpenWeather API získáváme data o předpovědi počasí a do dashboardu zobrazujeme předpovědi na dalších 15h.

### 7.2 Inicializační formulář



Obr. 11: Inicializační formulář

Okno prvotního nastavení obsahuje uživatelsky přívětivé rozhraní, které uživatele provede nastavením systému. Nastaví se mód zavlažování, limitní hodnoty a lokace, ze které má PlantHub ukazovat předpověď počasí.

### 7.3 Dashboard

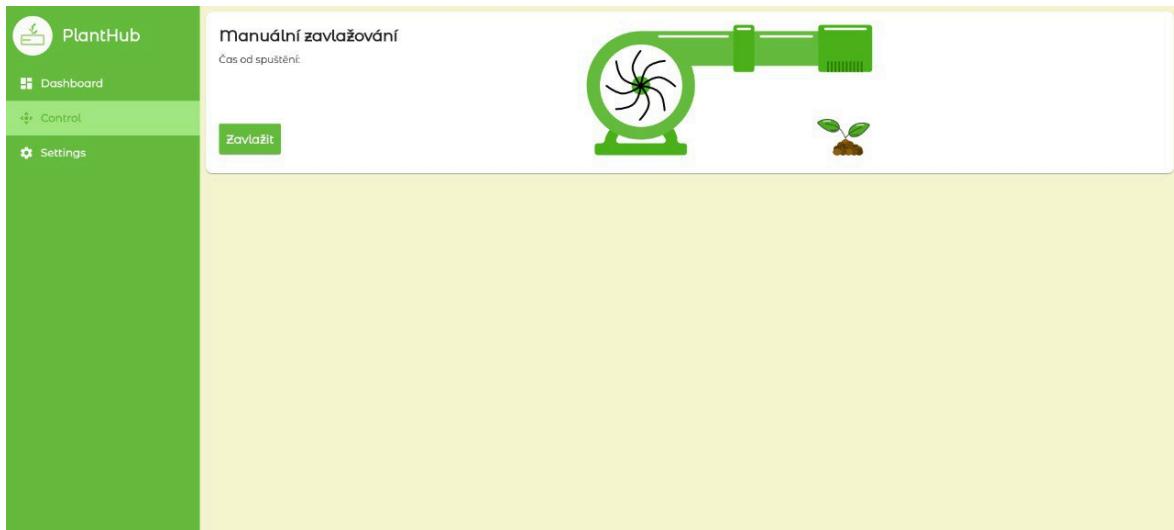


Obr. 12: Dashboard

Prostředí dashboardu (nástěnky) je hlavní částí webové aplikace. Uživatel zde může sledovat aktuální dění v podobně real-time dat ze senzorů a aktuálního stavu nádrže. K tomu jsme mu připravili praktickou kartu s předpovědí počasí a karty s grafy historických dat.

Prvním takovýmto grafem je historie zavlažování. Tento graf zobrazuje statisticky jaká byla teplota, vlhkost vzduchu a vlhkost půdy, těsně před zavlažováním. Na pravé straně jsou umístěny další dva grafy, první zobrazuje spotřebu vody při zavlažování za jednotlivé měsíce. Druhý potom vykresluje historické průměry naměřených dat v jednotlivých měsících.

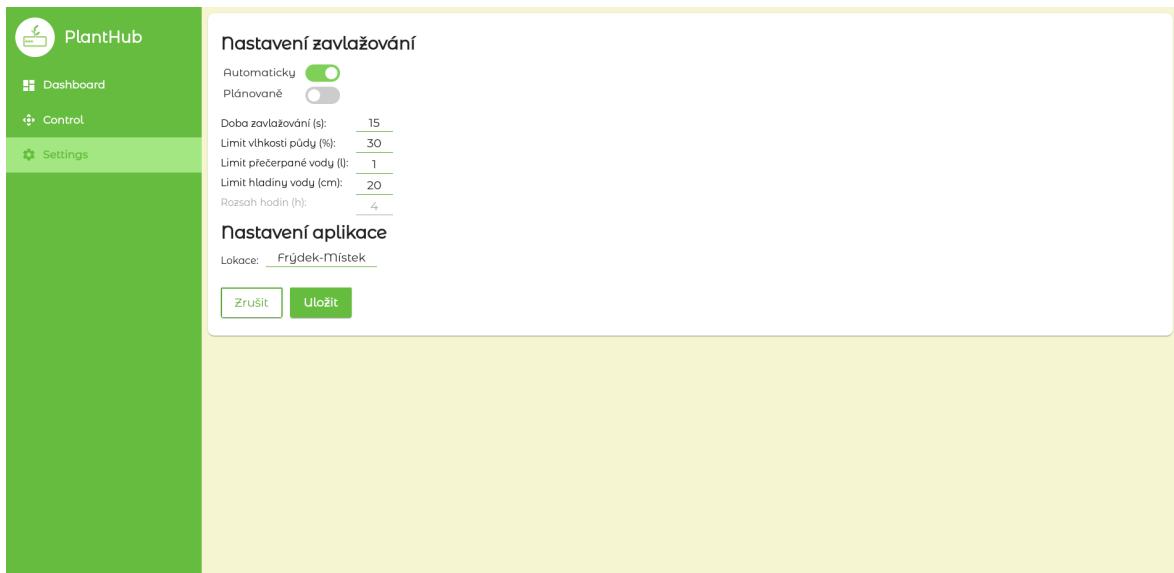
## 7.4 Manuální ovládání



Obr. 13: Manuální ovládání

Manuální ovládání čerpadla dovoluje uživateli kdykoliv spustit zavlažování. Po dokončení manuálního zavlažování, se také spustí kontrola hladiny vody v nádrži. Při úpravě limitních hodnot, či jiných nastavení řídící jednotky, je nutno zařízení restartovat. K tomu v menu WUI slouží možnost „Restart HW“.

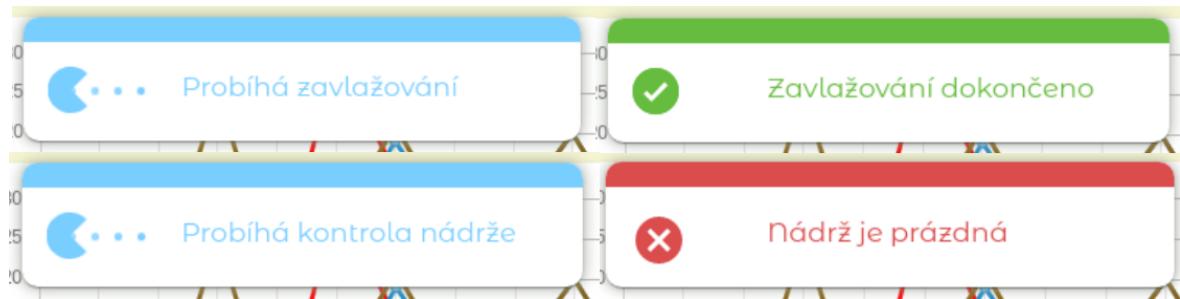
## 7.5 Nastavení



Obr. 14: Nastavení

V nastavení lze změnit nastavení aplikace, včetně limitů pro zavlažování a doby zavlažování. Na výběr máme automatické a plánované zavlažování. V prvním režimu PlantHub zavlažuje automaticky, kdy a jak moc má zavlažit, vyhodnotí z dat senzoru vlhkosti půdy. Při plánovaném zavlažování nastavíme intervaly, ve kterých se má zavlažovat. V nastavení aplikace se nastavuje lokace pro předpověď počasí. Nastavení se ukládají do databáze.

## 7.6 Notifikace



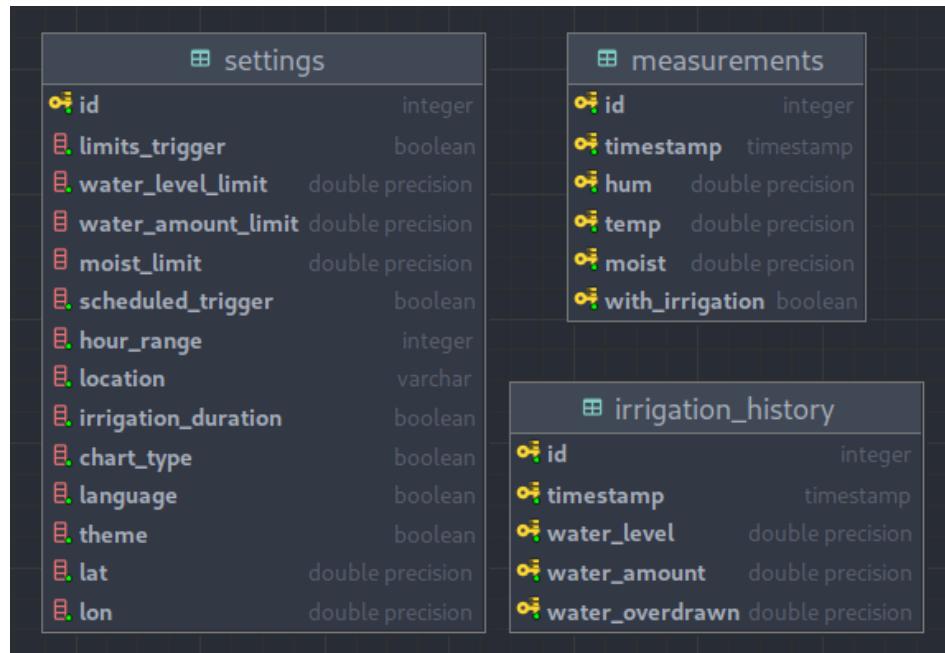
Obr. 15: Sada notifikací

Součástí webového rozhraní jsou i praktické notifikace, které uživatele upozorňují na aktuální stav PlantHubu. Poskytují vizuální zpětnou vazbu, při průběhu zavlažování a kontrole nádrže. Při úspěšném dokončení zavlažování, informují uživatele. V případě, že je nádrž na vodu prázdná, tak uživatele s touto skutečností seznámí.

## 8 Databáze

### 8.1 SQL

Pro databázi jsme se rozhodli použít relační databázový systém PostgreSQL. Jak vyplývá z názvu, jedná se o Structured Query Language (SQL) databázi, ty jsou vhodné pro ukládání velkého objemu dat, jako právě data z našich senzorů. Webová aplikace používá pro přístup k datům z databáze graph query language (GraphQL) [API](#). Naše schéma, jako je ukázáno na (Obr. 16), je propojeno pomocí [GraphQL](#).



Obr. 16: Schéma databáze

### 8.2 GraphQL

Jedná se o typ přenosu dat z možností filtrace a dotazovací funkcionality. V mnoha ohledech je to konkurent řešení přenosu dat pomocí representation programming interface (REST) [API](#). Nabízí rychlejší tvorbu [API](#) a také umožňuje efektivnější přístup k datům z databáze. Umožňuje vybrat pouze data, která momentálně aplikace používá a dovoluje vyněchat data, která zrovna potřebná nejsou.

## 9 Webový server

Web server jsme napsali v moderním jazyku Go, který nyní získává na popularitě, hlavně mezi cloudovými vývojáři a v oblasti vývoje microservices. Dokáže se v rychlosti provedení programu přiblížit k nízkoúrovňovým jazykům, jako je C nebo Rust, ale zároveň zůstává velmi lidsky čitelný a jednoduchý na použití. Na rozdíl od jazyků jako C a Rust, má například garbage collector (automatický správce paměti), který periodicky čistí paměť a zabránuje memory leakům.

## 10 Docker a kontejnerizace

Kontejnerizace je zabalení aplikace spolu se všemi nutnými komponenty, jako jsou knihovny, frameworky a další závislosti (dependencies), do svého vlastního „kontejneru“, který je izolovaný od zbytku systému.

Daná aplikace nebo program, potom může být spuštěna konzistentně kdekoliv, nezávisle na prostředí, operačním systému i architektury hostového systému. Aplikace běží zcela nezávisle na zbytku systému. Kontejner imituje jakousi bublinu, která aplikaci obklopuje a izoluje. Je to v podstatě plně funkční a přenosné výpočetní prostředí. Toto je hlavním důvodem, proč jsme zvolili kontejnerizaci pro náš projekt. V případě, že bychom chtěli vyrobit více jednotek, tak můžeme na jednotlivé RPi nasadit předpřipravené kontejnery a nemusíme se skoro vůbec zabývat konfigurací operačního systému [RPi](#).

Kontejnery poskytují alternativu ke klasickému vývoji aplikací na jedné platformě, nebo operačním systému. V některých případěch nevýhodou kontejnerů je, že mohou zhoršit kompatibilitu aplikace s jinými systémy, které nemusí sdílet stejné prostředí, jako systém, na kterém byla aplikace původně vyvíjena. Tato zhoršená kompatibilita může způsobit různorodé problémy a chyby aplikace, které se potom musí řešit navíc. V realitě to pak znamená prodloužený čas vývoje, snížení produktivity a frustraci vývojářů.

Díky zabalení aplikace v kontejneru, pak může být jednoduše spouštěna napříč platformami a operačními systémy. Také ji můžeme jednoduše přesouvat mezi zařízeními, protože vše co potřebuje, má zabalenou ve svém kontejneru.

Myšlenka kontejnerizace existuje dlouho. Docker, ale v roce 2013 stanovil standart kontejnerizace technologií Docker Engine. Docker Engine přinesl jednoduché nástroje pro vývoj, díky nimž se stal vývoj s použitím kontejnerů snadný. Zavedl také univerzální přístup k distribuci kontejnerů, což zrychlilo adopci kontejnerizace. Dnes si vývojáři mohou vybrat z nabídky kontejnerizačních platform, které podporují standart Open Container Initiative, vyvinuté společností Docker.

### 10.1 Výhody kontejnerizace

Kontejnery sdílejí kernel (jádro) operačního systému s jejich hostem. Kontejnery tak nepotřebují svůj vlastní operační systém a díky tomu jsou výpočetně nenáročné. Můžou tak běžet stejně na jakémkoliv zařízení, ať už se jedná o malé [RPi](#), nebo cloudová datacentra.

## 10.2 Kontejnery vs. virtuální počítače

Virtuální počítač je virtuální prostředí, které funguje jako běžný počítač. Má svůj vlastní virtuální procesor, paměť, síťový adaptér a úložiště. Tyto komponenty jsou virtualizovány na fyzickém hardwaru hostujícího systému.

Kontejnerizace a virtualizace jsou podobné v tom, že poskytují plnou izolaci aplikace, aby byla jednoduše spustitelná v různých typech prostředí. Hlavním rozdílem je velikost, výpočetní náročnost a přenosnost.

Virtuální počítač je větší z této dvojice, jeho velikost je typicky měřena v gigabytech. Obsahuje svůj vlastní operační systém, který mu dovoluje provádět několik výpočetně náročných funkcí zároveň. Jeho zvýšený výpočetní výkon, mu umožňuje zapouzdřit, rozdělit, duplikovat a emulovat celé servery, operační systémy, desktopy, databáze a počítačové sítě.

Kontejnery jsou mnohem menší, velikostí většinou v řádech megabytů a neobsahují nic víc, než samotnou aplikaci a nutné prostředí pro její běh.

Virtuální počítače fungují dobře v tradičních a monolitických IT architekturách. Kontejnery byly navrženy s důrazem na nejnovější a vznikající technologie, jako cloudy, CI/CD a DevOps.

## 11 Bezpečnost

Náš projekt spadá do kategorie zařízení Internet of things (IoT), neboli Internet of Things. Česky by to byl „Internet věcí“. Bezpečnost IoT zařízení bylo vždy velké téma. Zařízení IoT jsou totiž nechvalně známé svou nedostatečnou bezpečností. Tyto zařízení často bývají pro útočníky pomyslnou bránou do lokální sítě jejich oběti.

Rozmach těchto zařízení u běžných lidí tento problém zvýraznil. Mezi poslední trendy technologie patří chytré domy, kde je vše ovládáno chytrými zařízeními, které lze nastavovat na dálku pomocí telefonu. Hlavní charakteristikou IoT je schopnost zařízení připojit se k internetu, a interagovat se svým okolím pomocí sběru a výměny dat. Jedná se o zařízení od chytrých ledniček, termostatů, automatických vysavačů a žaluzí až po žárovky. Tím že jsou všechny tyto zařízení připojeny k internetu, jen zvyšujeme šanci napadení.

Náš projekt je zcela nezávislý na serverech v cloudu, místo toho je komunikace s PlantHubem zprostředkována pouze v lokální síti. Tímto designem jsme sice znemožnili přístup k PlantHubu mimo lokální síť, ale zároveň jsme velmi pozitivně ovlivnili bezpečnost celého systému. Zařízení stále musí být připojeno k internetu, z důvodu získávání dat ohledně předpovědi počasí a polohy z API třetí strany. K přístupu a možnosti ovládání mimo lokální síť bychom potřebovali VPN server v lokální síti.

K tomu jsme vybrali Fedora IoT jako naši linuxovou distribuci. Tato distribuce je přímo určená zařízením jako je naše, jak již vyplývá z názvu distribuce. Byla navržena s velkým důrazem na bezpečnost a základní nastavení distribuce je velmi bezpečné. Mezi bezpečnostní funkce patří SecureBoot, automatické šifrování úložiště, firewall a další.

## 12 Závěr

Náš projekt je stále ve vývoji a některé z funkcí tohoto projektu, nefungují podle našich představ. I přes to, je náš systém v této chvíli použitelný. Myslíme si, že tento projekt má v budoucnu potenciál stát se úspěšným startupem. Proto plánujeme pokračovat v jeho vývoji i po maturitě.

Abychom však tento projekt mohli pojmet jako plně funkční systém, museli bychom ještě dokončit a předělat některé klíčové části. Jednalo by se především o rozdělení hostingu webové aplikace a databáze na samostatný server. Zajištění modulárnosti celého systému rozdělením řídící jednotky na samostatné moduly a hub zodpovědný za jejich provoz. V dalších neprioritních krocích, se můžeme bavit o dokončení plánované tmavé verze WUI a přidání podpory více jazyků. Pro odlišení od konkurence mnoha jiných zavlažovacích systémů, přidáme chytré funkce vypočítávání spotřeby vody, energie a na základě předpovědi počasí, typu rostliny a lokace. Uvažovali jsme také nad soukromou databází rostlin a jejich rozdělení, na základě způsobu pěstování. V komerčním modelu, bychom mohli z této databáze vytvořit placené API, pro jiné zavlažovací systémy.

Tato práce, byla pro nás velkým přínosem. Naučili jsme se mnoho nových dovedností a načerpali nové informace, které v naší profesi bezpochyby dále využijeme.

## 13 Seznam obrázků

1	Schéma obvodu čerpadla . . . . .	12
2	Grafické znázornění obvodu . . . . .	14
3	Schéma obvodu . . . . .	15
4	Finální verze <u>PCB</u> se senzory . . . . .	15
5	Pouzdro . . . . .	16
6	Hub . . . . .	17
7	První verze nádrže . . . . .	18
8	Finální verze nádrže . . . . .	18
9	Celý systém . . . . .	19
10	Vývojový diagram programu . . . . .	20
11	Inicializační formulář . . . . .	24
12	Dashboard . . . . .	25
13	Manuální ovládání . . . . .	26
14	Nastavení . . . . .	27
15	Sada notifikací . . . . .	28
16	Schéma databáze . . . . .	29

## 14 Reference

- [1] "Home," LaTeX, 06-May-2021. [Online]. Dostupné z: <https://latex-tutorial.com/>. [Zpřístupněno: 10-Dubna-2022].
- [2] "Build fast, reliable, and efficient software at scale," Go. [Online]. Dostupné z: <https://go.dev/>. [Zpřístupněno: 10-Dubna-2022].
- [3] "React – a JavaScript library for building user interfaces," – A JavaScript library for building user interfaces. [Online]. Dostupné z: <https://reactjs.org/>. [Zpřístupněno: 10-Dubna-2022].
- [4] "Rapidly build modern websites without ever leaving your HTML," Tailwind CSS. [Online]. Dostupné z: <http://www.tailwindcss.com/>. [Zpřístupněno: 10-Dubna-2022].
- [5] "JavaScript with syntax for types," TypeScript. [Online]. Dostupné z: <http://www.typescriptlang.org/>. [Zpřístupněno: 10-Dubna-2022].
- [6] Microsoft, "Visual studio code - code editing. redefined," RSS, 03-Nov-2021. [Online]. Dostupné z: <https://code.visualstudio.com/>. [Zpřístupněno: 10-Dubna-2022].
- [7] "Goland by jetbrains: More than just A go ide," JetBrains. [Online]. Dostupné z: <https://www.jetbrains.com/go/>. [Zpřístupněno: 10-Dubna-2022].
- [8] Git. [Online]. Dostupné z: <https://git-scm.com/>. [Zpřístupněno: 10-Dubna-2022].
- [9] "The Collaborative Interface Design Tool," Figma. [Online]. Dostupné z: <https://www.figma.com/>. [Zpřístupněno: 10-Dubna-2022].
- [10] "Freecad," FreeCAD. [Online]. Dostupné z: <https://www.freecadweb.org/>. [Zpřístupněno: 10-Dubna-2022].
- [11] An easier and PowerfulOnline PCB design tool. EasyEDA. [Online]. Dostupné z: <https://easyeda.com/>. [Zpřístupněno: 10-Dubna-2022].

## **15 Seznam použitého softwaru**

1. Visual Studio Code
2. GoLand
3. EasyEDA
4. FreeCAD
5. Figma
6. Git

## **16 Seznam příloh**

- Příloha 1: Typový systém v Go.png
- Příloha 2: Source-code
- Příloha 3: Pcb
- Příloha 4: Case