Jihočeská univerzita
v Českých Budějovicích

Přírodovědecká fakulta

Modulární systém pro automatické zavlažování

Bakalářská práce

Bruno Semerek

Vedoucí práce : PhDr. Milan Novák Ph.D.

České Budějovice 2023

Bibliografické údaje

Semerek Bruno, 2023: Modulární systém pro automatické zavlažování. [Modular system for automatic irrigation. Bc. Thesis in Czech] – 70 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace:

Bakalářská práce se zabývá vytvořením a implementací systému pro automatické zavlažování s ohledem na nasazení ve skleníkových provozech. V teoretické části se zaobírá analýzou problému zálivky rostlin a možnostmi automatizace celého procesu. Praktická část se zabývá návrhem a sestrojením funkčního řešení včetně napsání obslužného software pro jednotlivé komponenty a napsání uživatelské aplikace.

Annotation:

Thesis aims at the development of a system for automatic irrigation, taking into account the deployment in greenhouses. Theoretical part deals with the analysis of plant irrigation problem and possibilities of automation of whole proccess. Practical part deals with design and construction of the functional solution, including writing utillity software for individual components and writing user application.

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval(a) pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V Českých Budějovicích, dne 13.04.2023 _____

Bruno Semerek

Poděkování

Rád bych poděkoval panu PhDr. Milanu Novákovi Ph.D., vedoucímu mé bakalářské práce, za cenné rady, odborné vedení, pomoc při zpracování a zapůjčení řídící jednotky Raspberry Pi. Velké poděkování dále patří rodině a přátelům za jejich podporu a trpělivost po dobu psaní mé práce.

Obsah

1. Úvod

V dnešní době je velká snaha zautomatizovat běžné činnosti. Automatizace je proces, který se snažíme začlenit do všech možných odvětví, ať už se jedná o výrobní provozy, chytrá auta, či domácnosti. Důvody jsou různé, od ulehčení práce, přes zvýšení efektivity, až po shromažďování dat, které není možno efektivně a spolehlivě získat při vykonávání stejného procesu manuálně.

Téma modulárního systému pro automatické zavlažování bylo vybráno z důvodu, že má rodina již přes 20 let vlastní a spravuje rodinné zahradnictví, kde jednou z činností, která se dělá velice často, je závlaha rostlin. Na jednu stranu velice jednoduchá věc, pro kterou si člověk řekne, že stačí pár pracovníků a konví, na tu druhou se jedná o proces, který je třeba dělat pravidelně a neznalý člověk lehko rostlinu přelije, či ji dostatečně nezalije, nehledě na to, že zalít celé zahradnictví, či několik plných skleníků může zabrat podstatné množství času.

Tato práce se zaobírá sestavením vhodného automatizovaného modulárního systému pro zálivku s ohledem na nasazení ve skleníkových provozech.

Práce se bude skládat z části teoretické, jež bude rozebírat problematikou skleníkových provozů a budou zde nastíněny již existující řešení automatizace v oblasti zavlažování rostlin ve skleníkových provozech. Další částí bude část praktická, která bude obsahovat návrhem a sestrojením systému pro automatickou závlahu rostlin.

1.1 Cíle práce

Cílem bakalářské práce bude navrhnout řídící systém schopný automatické zálivky na libovolně velké ploše. Tento systém bude řídit zálivku na základě vlhkosti půdy. Uživatel tohoto systému bude schopen měnit frekvenci zálivky pro jednotlivé oblasti tímto systémem pokrývané. Z návrhu vzejde reálný prototyp řešení. Pro splnění hlavního cíle je potřeba splnit následující podružné úkoly:

1. Analyzovat potřeby rostlin pro jejich růst v souvislosti s jejich pěstováním ve sklenících.
2. Prozkoumat existující a dostupná řešení, jež se reálně nasazují ve skleníkových provozech s ohledem na použitá zařízení, způsoby komunikace a uživatelského přístupu.
3. Navrhnout a sestrojit funkční prototyp, který bude zohledňovat předchozí body a bude vycházet z již vytvořených řešení prozkoumaných v bodě nad.

1.2 Metodika

Pro dosažení cílů bude provedena analýza formou rešerše, která se zaměří na současný stav problematiky automatizace ve skleníkových provozech. Analýza se bude zaobírat již vytvořenými systémy a bude zohledňovat následující vlastnosti a funkce daného systému :

- Úroveň samostatnosti systému při provozu
- Jednotlivé hardwarové komponenty systému
- Možnosti vzdáleného přístupu uživatele k systému
- Modularita systému

Analýza poskytne přehled v oblasti používaných postupů a komponent, který bude využit při návrhu a sestrojení vlastního systému.

2. Teoretická část

V této části se práce bude zaobírat analýzou potřeb rostlin, co se závlahy týče, dále analýzou dostupných řešeních pro automatickou zálivku ve skleníkových provozech na trhu a existujícími embedded projekty na toto téma.

2.1 Problematika závlahy rostlin ve skleníkových provozech

Na každém skleníku, ať už se jedná o skleník na zahradě, či skleník pro plošné pěstování, lze pozorovat několik věcí. První je ta, že se jedná o větší plochu. Druhou věcí je, že na této ploše se povětšinou nenachází jeden druh rostliny, ale je jich tam několik. Každá s individuálním požadavkem na míru zálivky. Pokud existuje těchto skleníků více, mluvíme o skleníkovém provoz. V takovémto provozu může jeden celý skleník obsahovat specifický druh rostliny, ale i nemusí. Nejenže je tedy nutné pozorovat individualitu jednotlivých rostlin, ale je třeba brát ohled i na individualitu jednotlivých skleníků. Stejně tak je třeba brát v potaz systém přívodu vody do skleníku a počítat s přizpůsobitelností zavlažovacího systému již existujícímu systému rozvodu vody. Vodní ventily, které propouštějí vodu musí vydržet určitý tlak, většina skleníkových provozů využívá vodní čerpadlo, které je schopné přivádět do oběhu vodu o maximálním tlaku 60 psi (pounds per square inch), což je 4,13 barů, nebo 0,413 MPa [17].

2.1.1 Požadavky rostlin na závlahu

Každá rostlina, ať už se jedná o malou sazeničku, záhonovou rostlinu, keř, strom, či trávník, spotřebuje určité množství vody. To, kdy rostlina vyžaduje vodu závisí z největší míry na tom, zdali je půda, ve které je rostlina zakořeněna, suchá. Rostliny si berou vodu z půdy neustále a ač se na první pohled může zdát, že rostlina potřebuje zalít a půda je suchá, u kořenů tomu tak být nemusí. Vlhkost půdy je tedy třeba měřit nikoliv na povrchu, ale tam, kde se nachází kořenový systém rostliny [2].

Avšak přes přílišné zalévání má negativní vliv téměř na každou rostlinu. Pokud budeme dodávat kořenovému systému více vody, než je schopný vstřebat, může se rostlina začít tzv. „topit“. Pakliže bychom tento styl zalévání udržovali delší dobu, rostlina nenávratně shnije. Proto je důležité měřit vlhkost půdy u kořenů, nikoliv na povrchu [2].

Závlaha rostlin se také značně odvíjí od typu půdy, ve které se rostlina nachází. Tyto druhy půdy se rozdělují od písčitých, tedy půd, které špatně zadržují vodu, až po jílovité, které naopak vodu velmi špatně vstřebávají [2].

Dalším důležitým faktorem je denní doba, která se váže k ročnímu období. Pokud je jaro, a teploty se pohybují kolem 20 stupňů, rostliny stačí zalívat 2x denně, v dopoledních a odpoledních hodinách. Naproti tomu v létě, nebo kdykoliv, kdy jsou teploty vyšší (32.2 stupňů +) je třeba rostliny zalívat buď brzy z rána, nebo k večeru, kdy jsou teploty nízké. Při vysokých teplotách dochází k většímu odpařování vody a kořenový systém rostliny ji není schopen včas vstřebat [1].

2.1.2 Možnosti závlahy rostlin

Požadavky rostlin na závlahu, jež jsou uvedeny v kapitole 2.1.1 Požadavky rostlin na závlahu, lze obecně vyřešit dvěma způsoby.

- Využití automatizace
- Použití lidské síly

Pokud přistoupíme k možnosti využití dnes dostupných technologií pro automatizaci, dosáhneme značné redukce potřeby lidských zdrojů pro zálivku ve skleníkovém provozu. Pro toto řešení je nezbytné provést analýzu v oblasti využívaného hardware v kontextu již vytvořených řešeních.

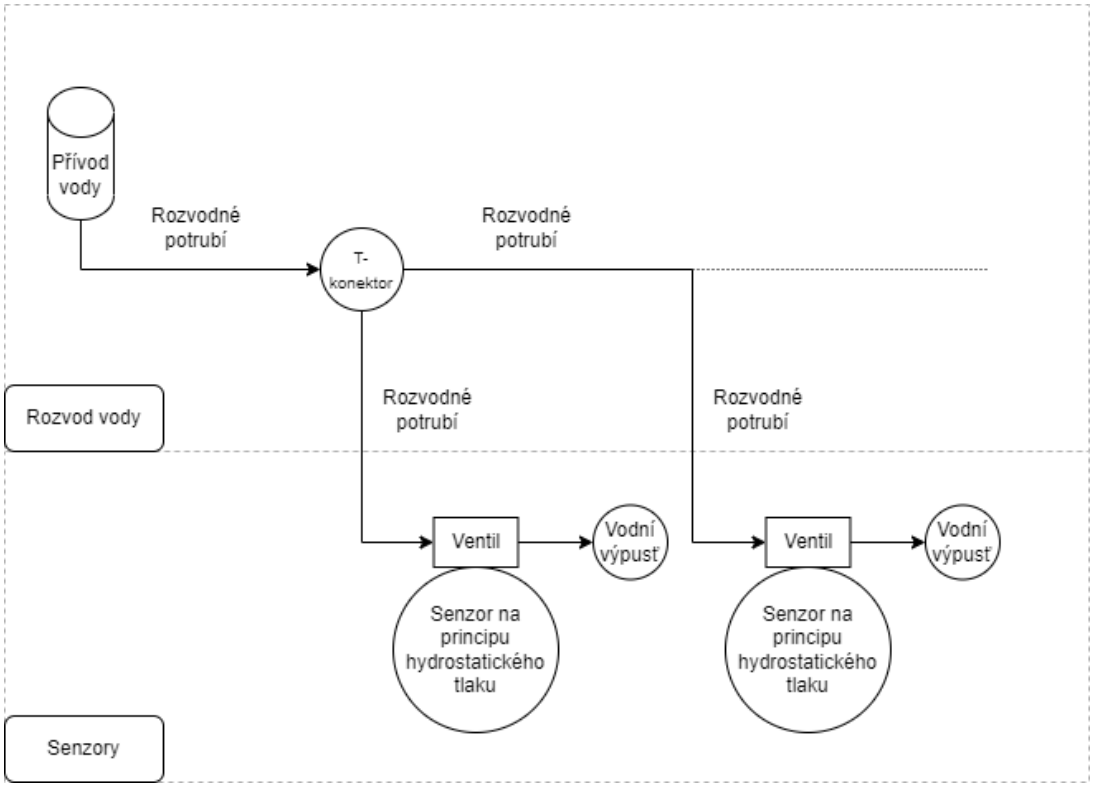
Druhou možností je využití lidské síly k manuálnímu zalévání a sledování růstu rostlin. Což s sebou nese značné nedostatky v podobě nekvalitní zálivky a vynaloženého času. Popřípadě máme možnost tyto dva nastíněné způsoby zkombinovat.

2.2 Analýza dostupných řešení

Na trhu se nachází mnoho řešení, jež poskytují systém pro automatickou zálivku skleníků, nebo rodinných zahrádek. Čím dál více také vznikají embedded projekty od různých nadšenců, ať už jde o koníček, nebo čistě vědecké práce. Analýza embedded řešení bude zaměřená spíše obecně, nežli na konkrétní systémy, jednak kvůli nepřebernému množství jednotlivých řešení, jednak díky možnosti jednotlivé dílčí prvky mezi s sebou libovolně kombinovat a utvářet tak komplexní systém.

2.2.1 Tropf-Blumat

Tropf-Blumat je jednoduchý automatizovaný senzorový systém pro zálivku. Skládá se z rozvodných trubek, rozdvojek v podobě T-konektorů a samoregulačních ovládacích senzorů, které operují na základě tlaku vody, není tedy potřeba žádné elektřiny. Senzor má v sobě vodu a funguje na základě osmózy, kdy pozná, pokud je okolní půda suchá. Pakliže ano, uvolní se tlak působící na senzor a ten otevře vodní ventil. V opačném případě suché půdy naopak okolní tlak způsobí uzavření ventilu.



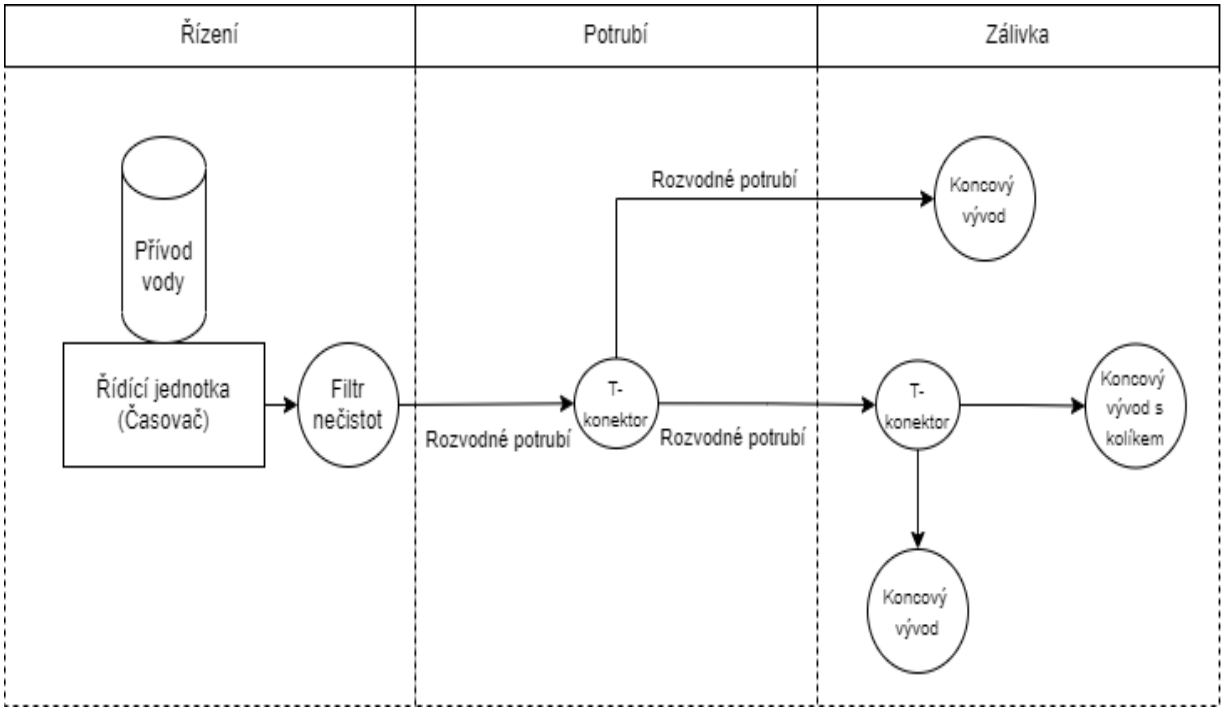
Obr 1: Troph-Blumat principle

Výhodou je nulová potřeba elektrických zdrojů, jediné co je nutné je přívod vody. Samotná instalace toho systému je tedy velice jednoduchá a je ho možné libovolně rozšířit o zvolený počet senzorů. [3]

Nevýhodou je nemožnost zpracování senzorických dat, neboť senzory nemají žádnou procesní jednotku, ani nejsou připojeny k žádné formě jednotky řídící. Jedná se o jednoduchý automatizovaný systém pro nenáročné uživatele s možností modularity v oblasti počtu senzorů, jak je uvedeno na schématu výše [15].

2.2.2 Hozelock

Zavlažovací systém od firmy Hozelock je schopen zalévat až 25 různých míst, v závislosti na velikosti balíčku, jenž si uživatel zakoupí. Skládá se z jednotlivých rozstřikovačů, které jsou připojení k řídící jednotce napájenou baterií. Ta obsahuje 16 před determinovaných programů pro distribuci vody, ke si lze vybrat zda chceme zalévat 1x týdně, až po možnost zálivky 4x denně. Rostřikovače dodávají vodu přímo ke kořenovému systému rostliny. Systém je dodáván společně se sadou potrubí, kterým lze jednotlivé rozstřikovače připojit k centrální jednotce [4]. Schéma tohoto systému je uvedeno níže [16].



Obr 2: Hozelock. Irrigation System.

Výhody tohoto systému jsou uživatelská jednoduchost, snadné nasazení do provozu a základní možnost modularity v podobě připojení dalších rozstřikovačů.

Značnou nevýhodou je fungování systému na principu plošného zalévání, kde se nebere v potaz individuální potřeba každé z rostlin kvůli absenci senzorů. Dále nemožnost uživatele vzdálené komunikace se systémem, neboť řídící jednotka má pouze funkcionalitu časovače.

2.2.3 Využít Embedded systémů při závlaze skleníkových provozů

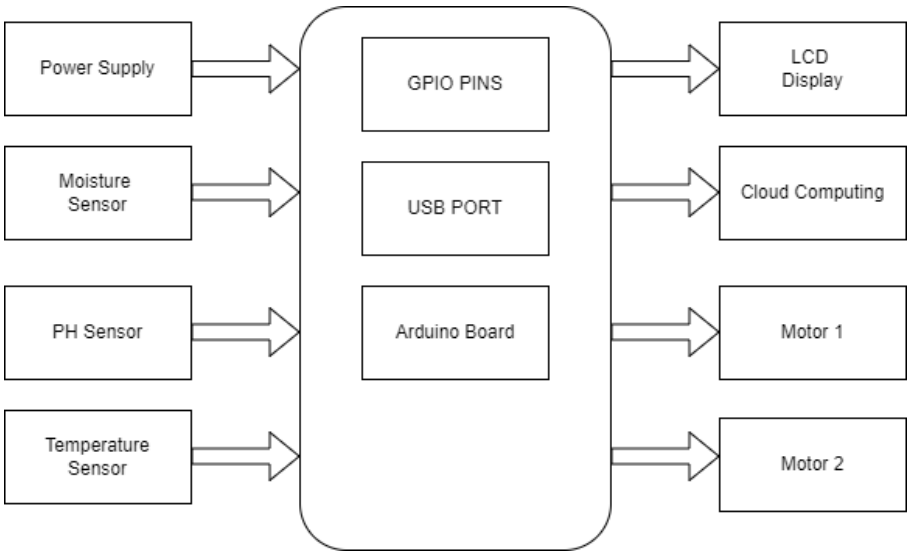
Embedded řešení zahrnují projekty využívající design embedded systémů. Řídící jednotka v takovémto případě obsluhuje rozhodovací procesy a spouští zálivku individuálně tam, kde senzory k jednotce připojené hlásí potřebu zálivky. Tyto senzory sbírají vlhkost půdy, ale jsou schopny měřit i údaje o vlhkosti vzduchu, dešti, kyselosti půdy a mnohé další. Často se implementují řešení, která využívají možnost připojení řídící jednotky k internetu a dálkovou správu celého systému.

Řídící jednotka

Ukazuje se, že nejčastěji se pro řídící jednotku používá mikrokontrolér Arduino. Výhodou Arduina jsou malé pořizovací náklady a dostatečný výpočetní výkon pro základní úlohy zpracování dat. [6] [8] [12].

V případě, že potřebujeme bezdrátové připojení k sítí, lze nainstalovat na Arduino WiFi shield, nebo vyměnit Arduino za ESP8266, které je vybaveno pro komunikaci WiFi a zároveň zvládá vykonávat základní úlohy nutné pro zpracování dat. [6] [12] [13].

Pokud řešení potřebuje větší výpočetní výkon řídící jednotky a zároveň vyžaduje funkce připojení k internetu za pomocí Bluetooth, WiFi, nebo pomocí Ethernet kabelu, využívá se Raspberry Pi, jehož novější modely, počínaje od modelu Rasperry Pi 2, tyto funkce obsahují bez nutnosti instalace dalších rozšiřujících modulů, jako je tomu v případě Arduina. [7] [14]. Příklad schématu popisující funkci řídící jednotky je popsáno na obr. č.3 [6].



Obr 3: Příklad blokového schématu funkce řídící jednotky.

Senzory

Senzory jsou nedílnou součástí embedded systémů pro zálivku. Měření dat umožňuje sledování hodnot v reálném čase a také umožňuje systému automaticky reagovat. Měřit lze různé hodnoty, ale nejčastěji měřenou hodnotou je vlhkost půdy, jež je přítomna téměř ve všech embedded řešeních zabývajících se problematikou zálivky. Zkvalitnění zálivky lze dosáhnout pomocí měření dílčích hodnot, jako je kyselost půdy [6] [12] okolní teplota [6] [12] [13] nebo vlhkost vzduchu [12].

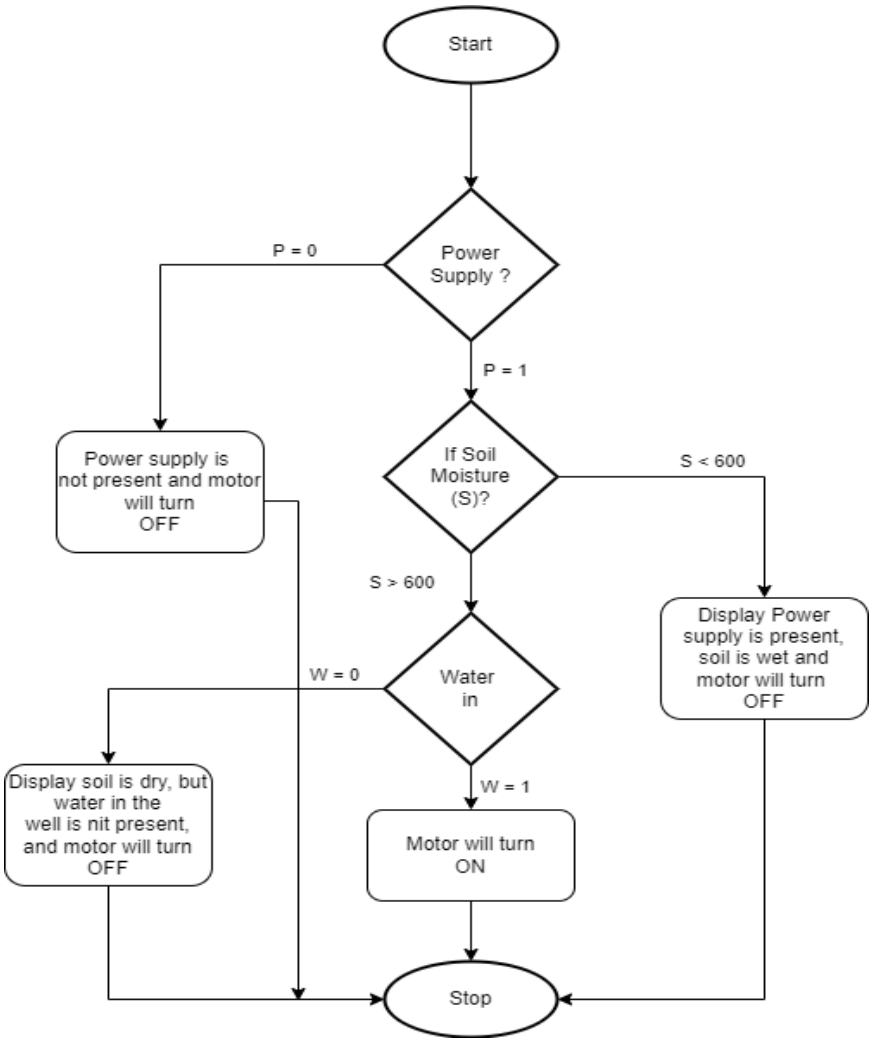
Komunikace se senzory

Cílem řídící jednotky je komunikace ze senzory. Tato komunikace může probíhat několika způsoby. Prvním, nejpřímějším způsobem komunikace řídící jednotky a senzorů je drátové spojení. V těchto řešeních je senzor přímo připojen na I/O piny řídící jednotky [6] [7] [13].

Jiné způsoby využívají možností bezdrátové komunikace mezi řídící jednotkou a senzory, jako je například WiFi [14], Bluetooth [12], nebo MQTT [8]. Stejné způsoby lze poté využít při připojení řídící jednotky k internetu.

Zpracování naměřených dat

Data naměřená senzory je třeba zpracovat a vyhodnotit. Způsob zpracování zůstává podobný napříč většinou řešení, kdy na se na základě naměřených dat vypočítá hraniční hodnota a v případě potřeby se spouští zavlažovací systém. Schéma tohoto procesu popisuje obrázek č. 4 [14].



Obr 4: Flow chart pro rozhodnutí o zálivce s přidanou kontrolou vodní nádrže

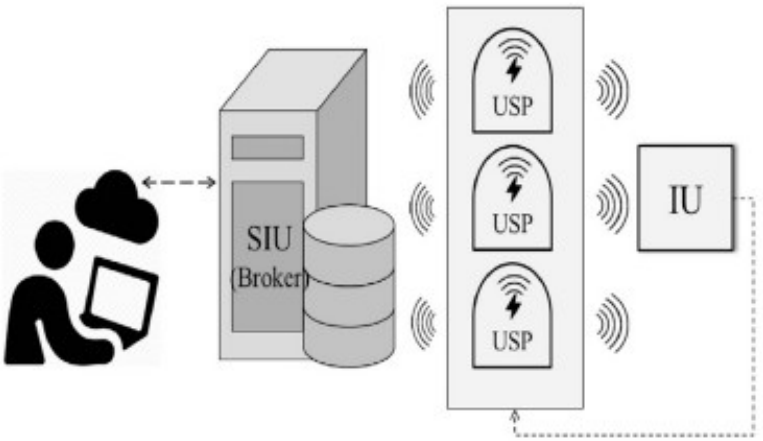
Kde se řešení liší, je místo, ve kterém se data zpracovávají a vyhodnocují. Pokud se vyhodnocuje větší množství dat, tak proces vyhodnocení a řízení zálivky probíhá v samotné řídící jednotce [6] [7] [12]. Pakliže samotný senzor obsahuje mikrokontrolér, je možno vyhodnocovat jednoduché podmínky a řídit zálivku v dané oblasti přímo za pomocí senzoru, v takovém případě mluvíme o senzorické jednotce [14]. Další rolí řídící jednotky je sběr dat ze senzorů pro monitoring a řízení uživatelského přístupu, jak je uvedeno v kapitole Uživatelský přístup. V některých případech řídící jednotka zajišťuje obsluhu jiných částí systému, jako je tomu v případě automatického zavlažovacího vozidla. [12]. Další možnosti mohou zahrnovat vyhodnocování dat o nadcházejícím počasí [7].

Uživatelský přístup

Uživatelským přístupem je myšlena možnost uživatele kontrolovat a potencionálně řídit chod systému či jednotlivých komponent.

Některá řešení utilizují možnost přístupu uživatele skrz LCD display, na kterém je možné vidět aktuální hodnoty měření, či umožňují nastavení parametrů skrz tlačítkové ovládání v kombinací s výše zmíněným displayem [6] [12].

Mnoho embedded řešení umožňuje vzdálený přístup k řídící jednotce skrz uživatelské rozhraní, které může být v desktopové, webové, či mobilní podobě. Toto rozhraní umožňuje sledovat naměřené hodnoty jednotlivých senzorů, případně vzdáleně upravovat hraniční hodnoty pro spuštění zálivky. Pro vzdálený uživatelský přístup je nutné připojení řídící jednotky k internetu [8] [12] [13] [14]. Příklad takového uživatelského přístupu je uveden na obrázku č.5. Existují řešení, kde přímý uživatelský přístup není, ale monitoring je stále přítomen skrz zasílání výsledků měření na email [7].



Obr 5: Příklad uživatelského přístupu skrze řídící jednotku

Ukládání dat

Pro potřeby uživatelského přístupu a zpětného monitoringu se využívají databázová úložiště, ve kterých je možné skladovat naměřené hodnoty, ale i hodnoty které se přímo měření netýkají, jako jsou přístupové údaje uživatele ID jednotlivých senzorů a další data, jejichž informace se liší na základě potřeb uživatelské aplikace. Data jsou zde přítomná po dobu nezbytnou k jejich vy-

hodnocení a poskytují zpětný přehled o průběhu procesu zavlažování [13] [14]. V některých případech se tyto data využívají pro naučení neuronových sítí [8].

2.3 Shrnutí analýzy

Výše provedená analýza způsobů zálivky ve skleníkových provozech, systémů dostupných na trhu a embedded řešeních odhalila, že většina systémů pro automatické zavlažování je použitelná i pro zavlažování skleníkových provozů, avšak jen do určité míry.

Z provedené analýzy vyplývá, že systémy pro automatickou zálivku se skládají z následujících komponent:

1. Řídící jednotka
2. Senzorická jednotka (zde se nabízí možnost spojení řídící a senzorické jednotky v jednu komponentu)
3. Server
4. Uživatelská aplikace.

Body 3 a 4 jsou přítomny v případě, že se od systému očekává možnost vzdáleného přístupu nebo schopnost systému publikovat výsledky měření.

Nejdůležitější veličinou, která má vliv na kvalitu zálivky je aktuální vlhkost půdy. Další dílčí veličiny zefektivňují výslednou zálivku, ale nejsou potřebné k jejímu zautomatizování. Na základě vlhkosti půdy se dá rozhodnout o tom, zdali potřebuje rostlina zálivku, či nikoliv. Zálivka je provedena pomocí otevření vodního ventilu, který musí vydržet tlak, jež je přítomný v rozvodném systému vody skleníkového provozu. Jiné druhy ventilů jsou zapotřebí, pokud je rozvodný systém samotížný.

Modularita, která je nezbytnou složkou pro úspěšnou implementaci ve skleníkových provozech (kapitola 2.1 Problematika závlahy rostlin ve skleníkových provozech), bývá zastoupena jen do určité míry. Majorita existujících řešení počítá s neškálovatelností systému po dobu provozu a možnou modifikací systému pouze v prvotním kroku instalace. Navíc jsou tyto systémy převážně dělané pro pokrytí jediné plochy.

Jednoduchá, převážně komerční řešení, neumožňují řídit zálivku jinak, než za pomoci časovače, nebo předem nastavené hraniční hodnoty dat, která jsou příjmány ze senzorů. Pokud je přítomna řídící jednotka v podobě mikrokontroléru, tak tato jednotka vykonává rozhodovací procesy ohledně spouštění zálivky na základě naměřených dat. V některých případech zálivku spouštějí

samotné senzory a řídící jednotka se stará převážně o uživatelský přístup a komunikaci s úložištěm dat.

Komplexní řešení pro řízení provozu skleníků mohou využívat možností robotiky a strojového učení. Tyto možnosti mají značný vliv na modularitu systému, který výše zmíněné oblasti využívá.. Ač se tato práce těmito tématy nezabývá, je třeba je v kontextu možných řešení zmínit.

Co se týká uživatelského přístupu, povětšinou platí, že pokud má systém připojení k síti, umožňuje alespoň základní formu zobrazení měřených dat za pomocí desktopové, mobilní, či webové aplikace. Složitější systémy umožňují vzdáleně do určité míry řízení provozu.

Většina systémů využívá jako řídící jednotku Arduino, Raspberry, nebo konkrétně ESP8266, které má za velmi malou pořizovací cenu zabudovaný WiFi modul.

Z hlediska propojení jednotlivých prvků se velice často používá drátové spojení senzorů s řídící jednotkou. V případě dostupného monitoringu řídící jednotka komunikuje se serverovou aplikací, která obsluhuje jednotlivé uživatele, nebo využívá možností řídící jednotky hostovat vlastní web server. Pro dálkové propojení dílčích komponent jako jsou senzory a řídící jednotky lze využít WiFi společně s MQTT, či Bluetooth, ale také další protokoly, jež nejsou použity v řešeních uváděných v kapitole 2.2 Analýza dostupných řešení, jako je ZigBee [10] nebo LoRaWAN [11].

Všechny systémy, na kterých byla provedena analýza, splňují podmínky nezávislosti systému na lidské obsluze, avšak platí, že s rostoucí komplexností systému roste také náročnost instalace takového systému do provozu.

3. Návrh řešení

Na základě provedené analýzy lze sestavit řešení, které by zohledňovalo problematiku skleníkových provozů a zároveň vycházelo z již využívaných konstrukčních řešení při automatizaci zálivky [27]. Řešení bude zohledňovat následující nedostatky :

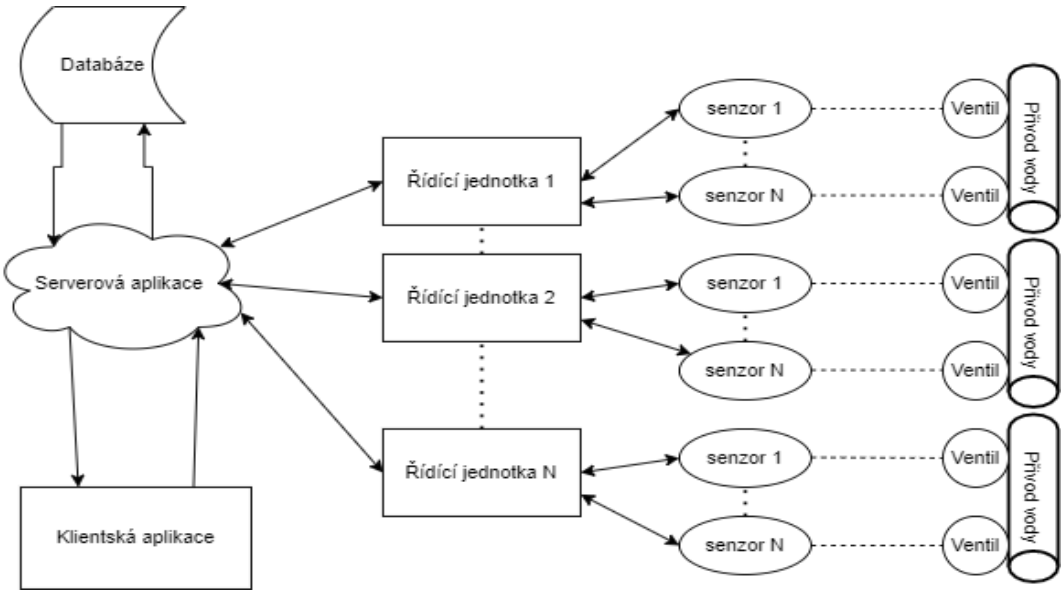
- Individuální potřeby pro zálivku vzhledem k umístění senzoru
- Pokrytí více ploch jedním systémem
- Jednoduchá rozšířitelnost systému i po jeho instalaci
- Obsluha systému za pomocí vzdáleného přístupu
- Nezávislost na systému rozvodu vody

Pokrytí těchto nedostatků povede k modulárnímu systému pro automatickému závlahu.

3.1 Obecná konstrukce řešení

K dosažení nastíněného návrhu řešení budou využity jednotlivé prvky automatizace systému vycházející z provedené analýzy . Systém bude sestaven z následujících prvků:

1. Senzor, který slouží jako samostatná jednotka pro spouštění zálivky za pomocí otevření ventilu při dané hranici vlhkosti půdy.
2. Řídící jednotka, schopna pod sebou pojmout N senzorů. Účel řídící jednotky bude dálková komunikace ze senzory a komunikace se serverem.
3. Server sloužící k připojení jednotlivých řídících jednotek, klientů v podobě uživatelů a dále zajištující komunikaci s databází.
4. Databáze, jež poslouží pro ukládání naměřených dat, informací o tom, který senzor spadá pod jakou řídící jednotku, který uživatel vlastní danou řídící jednotku a ukládání přihlašovacích údajů.
5. Klientská aplikace jež bude sloužit ke registraci řídících jednotek pod konkrétní uživatelský účet, registraci jednotlivých senzorů pod konkrétní řídící jednotku a správu jednotlivých senzorů co se vlhkostní hranice zálivky týče.



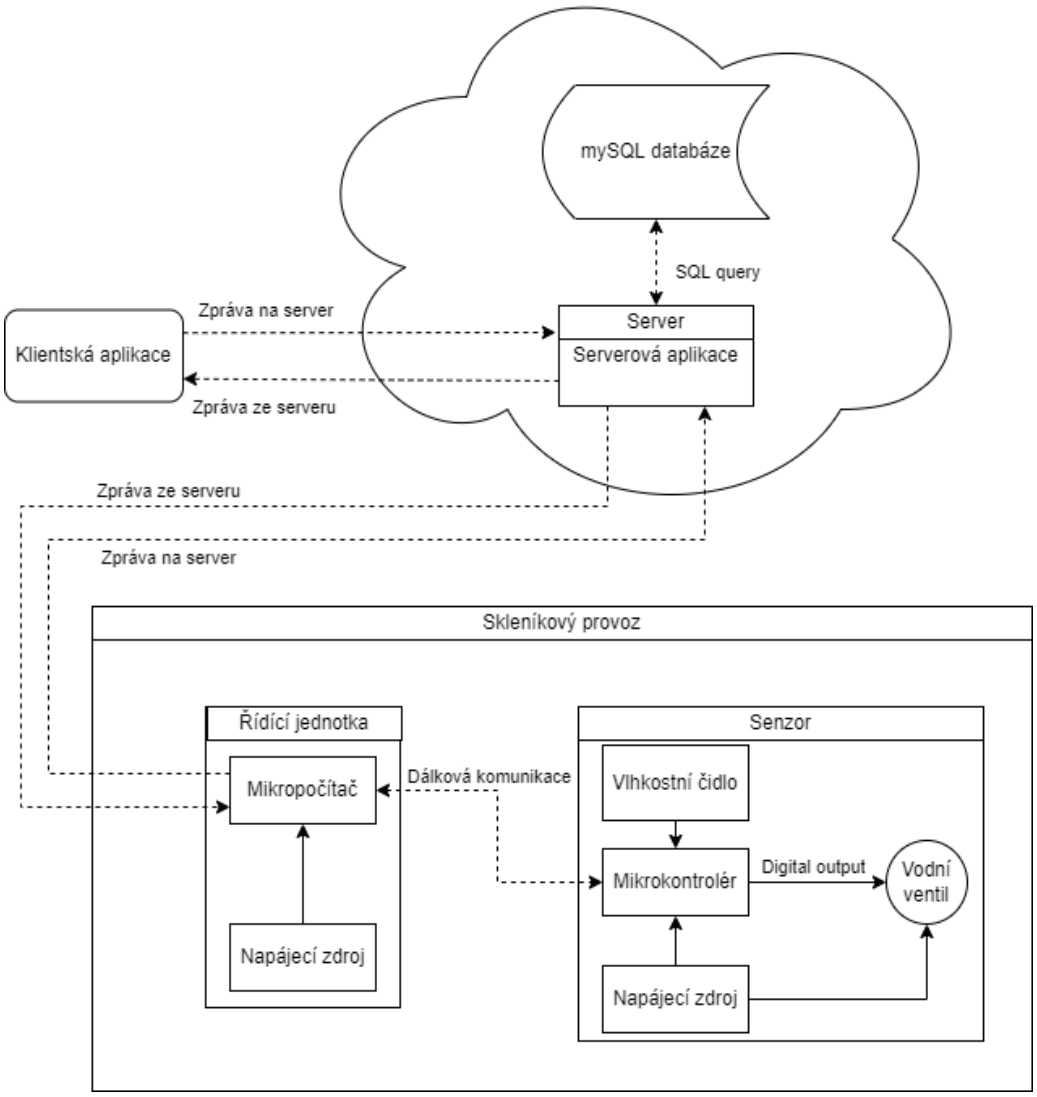
Obr 6: Obecné schéma navrhovaného systému.

4. Praktická část

Řešení praktické části vychází z provedené analýzy, převážně z části zabývající se embedded systémy a návrhu řešení zmíněném v bodě 3.1 Obecná konstrukce řešení. Praktická část bude složena z podrobného návrhu řešení, výběru vhodných komponent na základě dílčí analýzy, návrhu a sepsání obslužného software pro jednotlivé komponenty a sestrojením funkčního prototypu celého systému.

4.1 Podrobná konstrukce řešení

Konstrukce vychází z obecného schématu navrhovaného systému přítomného na obrázku č.6, na základě kterého lze sestrojit podrobnější schéma. (obr č. 7). Konkrétní varianty hardwarového řešení pro každý prvek budou rozebrány dále, počínaje senzorickou jednotkou.



Obr 7: Podrobné schéma řešení

4.1.1 Senzorická jednotka

Na základě provedené analýzy používaných senzorických jednotek, se od této jednotky očekávají následující funkcionality:

- Měření aktuální vlhkosti půdy
- Automatické spouštění zálivky
- Možnost vzdálené komunikace

Vlhkostní čidlo

Senzory na měření vlhkosti fungují na principu detekce změn, které ovlivňují elektrické napětí nebo teplotu ve vzduchu. Existují 3 základní typy senzorů pro měření vlhkosti : kapacitní, rezistentní a termální. Všechny 3 monitorují změny v atmosféře na základě kterých určují vlhkost ve vzduchu.

Kapacitní senzory fungují na principu kondenzátoru. Při přívodu napětí dojde k vytvoření elektrického pole mezi dvěma kondenzátorovými deskami. Mezi těmito deskami se nachází tenký kovový proužek, jehož kapacita se mění na základě relativní vlhkosti ve vzduchu.

Rezistentní senzory operují na principu odporu vodiče, jehož délka je přímo proporcionální vůči jeho odporu. Jak se mění odpor na vodiči, tak také stoupá nebo klesá výstupní napětí, na základě kterého lze určit výslednou vlhkost.

Termální senzory fungují na principu dvou vodivých objektů, z nichž jeden je obalen suchým dusíkem a druhý měří okolní vzduch. Rozdíl mezi těmito dvěma měřeními je naměřená vlhkost.

Jako vhodnou volbou se ukazuje analogový půdní vlhkoměr s antikorozní sondou V1.2. Jedná se o kapacitní vlhkoměr, který požaduje vstupní napětí 4,5 – 5,5 V a dodává na analogový výstup napětí 0- 4,5 V, v závislosti na naměřené hodnotě vlhkosti. Vybraný vlhkoměr je uveden na obrázku č.8 [20].

Obr 8: Půdní vlhkoměr analogový s antikorozní sondou V1.2

Vodní ventil

Vodní ventil je důležitou součástí, bez které by se nemohla spustit zálivka v době potřeby. Vzhledem k teoretickým východiskům práce je u ventilu nutná hlavně jeho výdrž, nejčastěji udávaná v Mega Pascalech (MPa) a spolehlivost.

Co se týče výdrže, 0,5 MPa by mělo zabránit proražení ventilu, což splňuje většina dostupných ventilů na trhu. Ventily se liší hlavně průtokem vody a výrobním materiálem. Pro účely práce přijde vhod využit jednoduchý elektromagnetický membránový ventil s požadovaným napětím 12 V, proudem 1 A a vstupně/výstupním průtokem ½ coulu. Defaultní poloha ventilu je zavřená, po přivedení napětí se změní do polohy otevřeno. Tento druh ventilu plně vyhovuje všem podmínkám, až na velké požadované napětí. Proto bude použit bistabilní ventil, jehož požadované napětí je pouze 3,6 V a požadovaný proud 500 mA. Mezi další výhody tohoto ventilu patří i jeho delší životnost. Ventil je možné vidět na obrázku č.9 [22].

Obr 9: Bistabilní ventil 1/2" ovládaný pulsem

Mikrokontrolér

Jádrem senzorické jednotky, zodpovědným za kontrolu senzoru na měření vlhkosti půdy a na základě jím naměřené hodnoty spouštějícím zálivku, je mikrokontrolér. Toto zařízení by vzhledem k provedené analýze, návrhu řešení a zvoleným komponentám mělo splňovat následující požadavky:

1. Dostatečný výpočetní výkon na provádění jednoduchých operací
2. Dostačující počet vstupních a výstupních pinů vhodných pro připojení jednotlivých dílčích komponent senzorické jednotky
3. Možnost vzdálené komunikace s řídící jednotkou

Prvním bodem se rozumí výkon, který je nutný pro zpracování naměřených dat, na základě kterých se otevře vodní ventil pro spuštění zálivky a zároveň umožní mikrokontroléru komunikaci s řídící jednotkou.

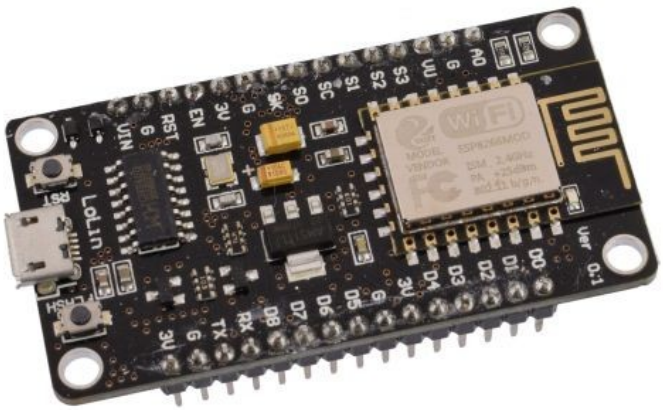
Druhý bod je podmiňován možností pro připojení senzoru schopného měřit vlhkost a připojení vodního ventilu. Je tedy potřebný alespoň vstupní 1 analogový pin, výstupní pin schopný dodávat napětí 5 V a digitální pin pro možnost spínat vodní ventil.

Vzdálená komunikace s řídící jednotkou je podmíněna podporovanou funkcionalitou v podobě WiFi modulu.

Z hlediska provedené analýzy a požadovaných funkcionalit se pro potřeby práce jeví jako nejlepší možnost využít ESP 8266. Deska obsahuje 32-bitový procesor Tensilica L106 s frekvencí 80 MHz, 32 Kb instrukční RAM, 32 Kb, instrukční cache RAM, 80 Kb Ram pro uživatelská data a 16 Kb RAM pro systémová data. Dále deska podporuje standart 802.11 s možností komunikace přes WiFi v pásmu 2.4 GHz až 2.5 GHz. Deska obsahuje sběrnici UART pro sériovou komunikaci a napájení, dále I^2C obsluhující 16 GPIO pinů pro připojení externích zařízení a 1 analogový pin, který je napojený na 10-bitový ADC převodník. Deska vyžaduje napětí 4,5 – 5 V a její provozní napětí na jednotlivých pinech je 3,3 V.

Oproti jiným možnostem, jako je například Arduino Nano, ESP disponuje WiFi modulem podporujícím standart 802.11 s možností připojení k EAP a WPA2 sítím bez nutnosti instalace dodatečného TCP/IP modulu.

Jako vhodné alternativy, vzhledem k požadované funkcionalitě, se jeví Wemos D1 mini, nebo Arduino Nano 33 IoT. Jedná se o kompaktnější mikrokontroléry, které však ztrácí oproti ESP 8266 NodeMcu z hlediska výkonu procesoru.



Obr 10: NodeMcu Espressif 8266

4.1.2 Řídící jednotka

Vzhledem k provedené analýze používaných řídících jednotek a možnosti dosažení cílů práce se od řídící jednotky systému očekávají následující funkcionality:

1. Výkon dostatečný na provádění výpočetních operací
2. Možnost vícevláknového zpracování
3. Schopnost vzdálené komunikace
4. Připojení k síti

Prvním bodem se rozumí takový výkon, který bude stačit na zpracovávání požadavků ze serveru a zároveň bude dostatečný na komunikaci s neurčitým počtem senzorických jednotek.

Splnění druhého bodu je nutné pro zajištění simultánní komunikace jak se serverem, tak s již zmíněným neurčitým počtem senzorických jednotek.

Třetí bod, tedy schopnost vzdálené komunikace je důležitý pro potřeby komunikace s jednotlivými senzorickými jednotkami bez potřeby natahování kabeláže.

Z hlediska uživatelského přístupu je nutné splnit bod 4.

S ohledem na výsledky analýzy a požadované funkcionality zmíněné výše bylo rozhodnuto o použití Raspberry Pi, jako řídící jednotky.

Řídící jednotkou je model Raspberry Pi 3 B+. Tato deska disponuje procesorem Quad Core 1.2GHz Broadcom BCM2837 64bit a 1 GB RAM. Deska podporuje jak 2.4 GHz, tak 5 GHz 802.11ac Wireless LAN (WLAN) a Bluetooth 4.1. Dále je možnost připojit zařízení do sítě skrz Gigabit Ethernet s maximální rychlostí přenosu 300Mbps. Z hlediska napájení je možno nainstalovat shield PoE Hat (Power over Ethernet), umožňující napájení přes Ethernet bez nutnosti připojení desky do elektrické sítě. Pro klasické napájení lze využít micro-usb konektor [21].

Pro možnosti vhodného řešení se bral v potaz i model Raspberry Pi 2 a Raspberry Pi 3. Od modelu 2 bylo upuštěno kvůli absenci bezdrátové komunikace. Model Raspberry Pi 3 oproti 3 B+ nabízí lehce nižší výkon a absenci možnosti instalace PoE shieldu.

Model 3 B+ poskytuje potřebný výkon pro simultánní běh několika vláken najednou, souběžný běh MQTT brokeru a dostatečnou podporu bezdrátové komunikace.

Obr 11: Raspberry Pi 3b+

4.1.3 Napájení

Způsob napájení jednotlivých prvků je důležité řešit z důvodů jejich neustále potřeby odběru elektrické energie.

Napájení řídící jednotky

Raspberry Pi použité v této práci vyžaduje ke svému správnému chodu napětí 5 V se spotřebou kolem 400 mA. Řídící jednotka by mohla být v tomto případě napájena klasickým adaptérem skrze micro-usb konektor. Lepším řešením, které přichází i s aktivním chlazením procesoru je využít PoE Hat. PoE (Power over Ethernet), je shield, který se nainstaluje na desku Raspberry Pi. Tento rozšiřující modul využívá klasický standart 802.3af, který umožňuje dodat zařízení až 15 W energie. Vylepšenou verzí je PoE+ Hat, který krom zvýšení proudu z 2,5 A na 5 A také umožňuje kontrolovat aktuální napětí pomocí technologie current sense. Nevýhodou je, nutnost směrovače / switche podporovat PoE technologii [23].

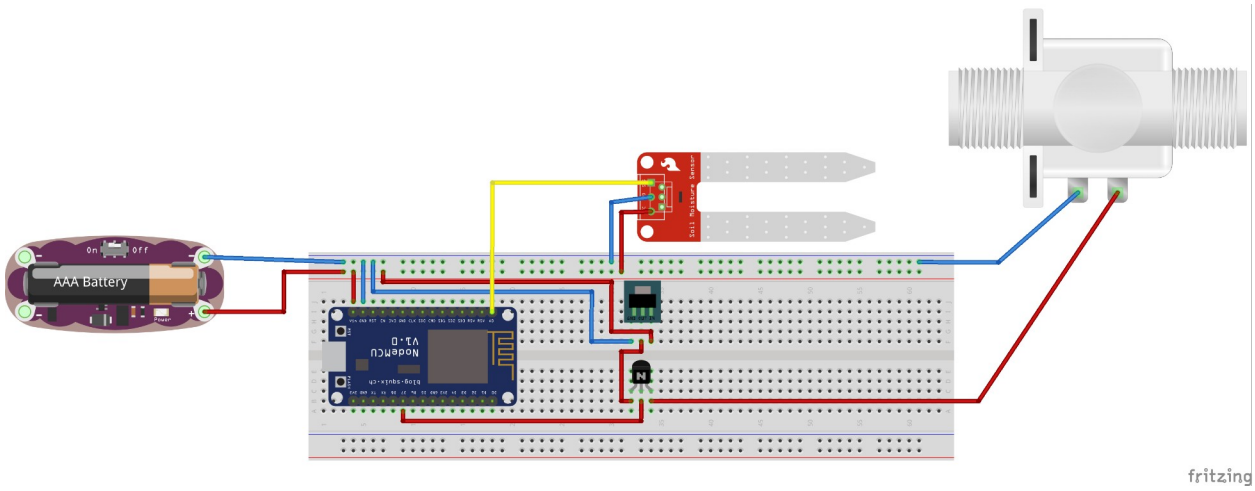
Vzhledem k použitému způsobu připojení řídící jednotky k sítí (kapitola 4.2 Dálková komunikace), by v této práci bylo nejvhodnější použít zmíněný PoE hat, ale kvůli nemožnosti otestovat

tento způsob napájení, z důvodu absence této technologie na mém směrovacím zařízení, bude využito napájení přes klasický adaptér z elektrické sítě.

Napájení senzorické jednotky

Napájení senzorické jednotky se skládá z potřeby napájet 3 různá zařízení. Samotný mikrokontrolér, vlhkostní čidlo a vodní ventil. Mikrokontrolér vyžaduje napětí 4,5 – 5 V a proud o velikosti 80 - 170 mA. Senzor požaduje vstupní napětí také 4,5 – 5V a malé, zanedbatelné množství miliampér. Ventil požaduje napětí o velikosti 3,6 V, ale požadovaný proud o velikosti 500 mA. Výsledný minimální nutný příkon zdroje je tedy 2,43 W, s minimální voltáží 4,5 V a dodávaným proudem alespoň 0,7 A. Z hlediska zajištění správné funkcionality prvků je lepší tato čísla nadsadit a počítat jako vhodný počet dodávaných ampér 1 A.

Pro účely testování se hodí mít senzorickou jednotku napájenou přímo ze sítě. K tomuto účelu by mohl posloužit klasický adaptér s USB připojením a vyvedením kabelů na druhé straně. Z hlediska praktického použití by bylo mnohem vhodnější využít akumulátor. V obou případech je potřeba minimální napětí 5V pro senzor a mikrokontrolér. Pro přivedení 3,3 V k vodnímu ventilu by byl využit step-Down z 5 V na 3,3 V, který by byl dále připojený na tranzistor, jež by umožňoval ovládání ventilu za pomocí přivádění napětí na jeho bázi z GPIO pinu řídící jednotky. Namísto tranzistoru a step-Down prvku by mohl být použit mosfet, jež obě tyto funkcionality zajišťuje.



Obr 12: Příklad zapojení s využitím step-down prvku a tranzistoru

4.2 Dálková komunikace

Touto komunikací je myšlena komunikace mezi konkrétní řídící jednotkou a konkrétním senzorem (obr. č. 7). Z hlediska provedené analýzy se zde nabízí několik možností.

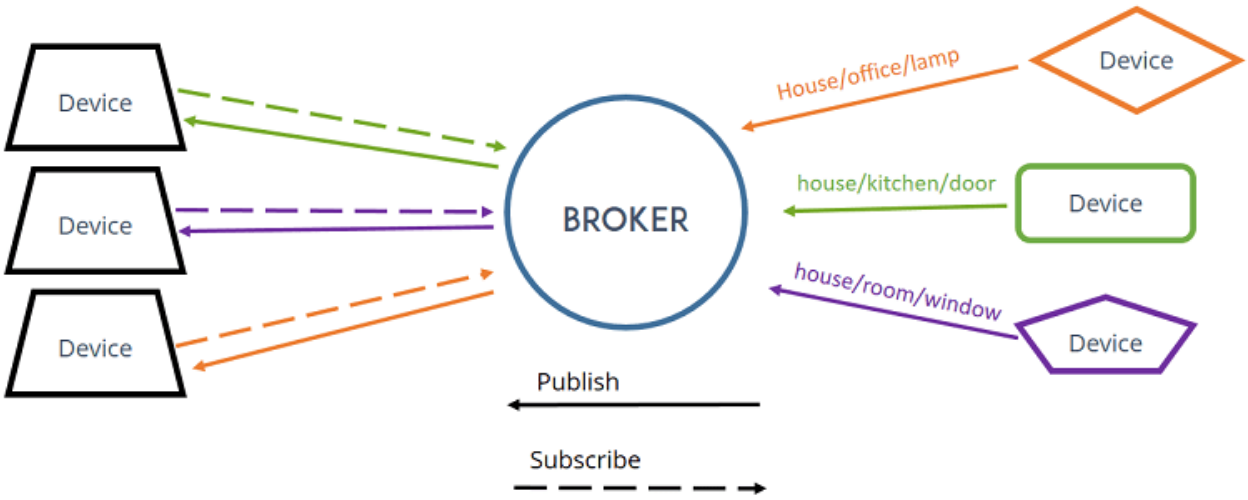
První možností je použít přímou komunikaci. Tato možnost, ač se vyskytuje ve velkém počtu řešení, na kterých byla provedena analýza, vzhledem k vytyčenému cíli modularity, není optimální. Důvodem je konečný počet možných připojených zařízení spadajících pod jednu řídící jednotku.

Druhou možností je bezdrátová komunikace, kde se nabízí, vzhledem ke zvolené řídící jednotce a provedené analýze dva způsoby komunikace. Bluetooth, a WiFi. Za předpokladu, že využijeme Bluetooth, můžeme řídící jednotku připojit k sítí přes WiFi. Nevýhodou Bluetooth je velmi krátký dosah, který je kolem 10 metrů. Při použití vzájemné komunikace přes WiFi je tento dosah až 100 metrů. Pokud použijeme WiFi, je nutné nakonfigurovat Raspberry Pi jako access point, ke kterému se budou jednotlivé senzory připojovat.

V této práci bude využito WiFi připojení, na kterém operuje MQTT protokol. Vzhledem k provedené analýze se jedná o velmi používaný a poměrně flexibilní způsob připojení který vyhovuje požadavkům této práce. Samotná řídící jednotka bude k sítí připojena skrze Ethernet.

4.2.1 MQTT

Protokolem, který se v IoT používá dnes čím dál více, je MQTT. Jedná se o kompaktní protokol postavený nad TCP/IP, jež slouží k přenosu dat mezi zařízeními. Princip MQTT tkví v existenci topiců a subscriberů. Zařízení v sítí je subscriberem pro určité topicy, nebo nemusí mít žádné. Pokud jiné zařízení bude chtít poslat zprávu, nepošle ji na určité zařízení, ale pošle ji na daný topic. V takové chvíli je toto zařízení publisher. Zařízení, která mají subscribe u topicu, na který byla zpráva odeslána, tuto zprávu získají. O přijímání zpráv, jejich filtraci, rozhodování komu zprávu poslat a publikování zpráv všem subscribnutým klientům se stará MQTT Broker. Schéma této komunikace je uvedeno na obrázku č.13 [18] . V MQTT sítí zařízení nekomunikují mezi sebou, komunikace probíhá vždy mezi zařízením a brokerem. Raspberry Pi má v sobě zabudovaný MQTT broker Mosquitto, který bude využit v této práci pro komunikaci řídících jednotek a senzorů.



Obr 13: Princip fungování MQTT brokeru.

4.3 Serverové řešení

K ukládání dat a vzájemné komunikaci řídících jednotek s klientskou aplikací bude využito serverové řešení, zajišťující vzájemnou TCP/IP komunikaci.

Pokud budeme vycházet z provedené analýzy funkčních řešení, tak platí, že většina stávajících řešení využívá web server běžící na řídící jednotce, což by se dalo využít, za předpokladu, že máme právě jednu řídící jednotku a můžeme předem říct, to jaké sítě se budeme z klientské aplikace připojovat. Poněvadž máme řídících jednotek více, na různých místech, v různých sítích, klientská aplikace nemůže vědět síť, ve které se každá řídící jednotka nachází. Stejně tak řídící jednotky nemohou vědět, v jaké sítí a na jakém zařízení běží zmíněná klientská aplikace. Problém se dá vyřešit redukcí problému 1 : N na problém 1: 1. Pokud bude veškerá komunikace s řídící jednotkami řízena přes server, jak klientská aplikace, tak řídící jednotky mohou mít předem definovanou právě jednu adresu, na kterou se musí připojit, pro navázání vzájemné komunikace.

Z hlediska ukládání dat bude využita MySQL databáze, se kterou bude komunikovat výhradně server a všechny požadavky, jež budou jednotlivý klienti na databázi mít, budou zpracovávány na serveru. Výhodou tohoto řešení je bezpečnost dat uložených v databázi, tedy nemožnost jakéhokoliv klienta přistoupit k datům bez správného požadavku na server.

4.4 Klientská aplikace

Pro účely monitorování průběhu zálivky, zobrazování naměřených dat a možnosti interakce s jednotlivými řídícími jednotkami a senzory bude vytvořena klientská desktopová aplikace, jež bude splňovat následující požadavky, vedoucí k přímé kontrole nad zálivkou a modularitou celého systému:

- Vytvoření a přihlášení se k uživatelskému účtu
- Registrování řídící jednotky pod konkrétní účet.
- Zobrazení dostupných senzorů v okolí registrované řídící jednotky
- Registrace dostupných senzorů pod konkrétní řídící jednotku.
- Nastavení hraniční hodnoty vlhkosti půdy pro konkrétní registrovaný senzor
- Zobrazení dat o průběhu zálivky a dat o stavu konkrétního senzoru
- Odregistrování senzoru zpod řídící jednotky
- Odregistrování řídící jednotky zpod uživatelského účtu
- Pojmenování řídící jednotek a jednotlivých senzorů

Aplikace bude fungovat na principu vystavování jednotlivých požadavků na server, který tyto požadavky zpracuje a vrátí klientské aplikaci odpověď.

4.5 Software

Jednotlivé komponenty systému požadují pro svoji funkčnost obslužný software. Z hlediska nutnosti programování se jedná o 4 různé komponenty : řídící jednotka, senzorická jednotka, server a klientská aplikace. Z hlediska provedené analýzy ve většině případů platí, že senzorická jednotka i jednotka řídící jsou jeden celek, sdílející tentýž výkonný kód. Vůči naplnění cílů práce je nutné brát jak samotný senzor, tak řídící jednotku jako dva samostatné celky, které jsou na sobě závislé pouze na základě vzájemné komunikace, nikoliv však na základě sdílení společného výkonného kódu.

Celý systém se skládá ze dvou různých částí. První částí je front-end, který zajišťuje interakci ze strany uživatele. V případě tohoto řešené se jedná o klientskou aplikaci. Druhou části, zajišťující zpracování dat mimo zařízení uživatele je back-end. Do této částí spadá programové vybavení

senzorické jednotky, aplikace běžící na řídící jednotce a serverové řešení zajišťující komunikaci mezi front-end uživatelskou aplikací a back-end částmi systému.

4.5.1 Funkční požadavky

Z hlediska celého systému by měli být splněny následující funkční požadavky:

- Vytvoření a přihlášení se k uživatelskému účtu
- Přidělování řídících jednotek k účtu a jejich odregistrování od účtu
- Přidělování senzorických jednotek k řídícím jednotkám a jejích odregistrování od řídící jednotky.
- Zobrazování aktuálně naměřených hodnot vlhkosti u registrovaných senzorů.
- Možnost nastavení hranice zálivky pro jednotlivé registrované senzory.
- Možnost přejmenování řídící jednotky a senzorické jednotky

Tyto požadavky lze dále specifikovat na základě konkrétní komponenty a tomu odpovídající obslužné aplikace.

Software řídící jednotky

Obslužný program řídící jednotky zajišťuje následující funkcionality:

- Komunikace se serverem.
- Navázání spojení s MQTT Brokerem
- Příjímání zpráv z MQTT brokeru
- Odesílání zpráv na MQTT broker

Software senzorické jednotky

Obslužný program, běžící na ESP bude k docílení komunikace s jednotlivými periferiemi a komunikace s řídící jednotkou umožňovat následující funkcionality:

- Připojení se k řídící jednotce skrz WiFi.
- Připojení se k MQTT brokeru, jež běží na řídící jednotce
- Zpracování dat naměřených na měřící vlhkostní sondě.
- Spouštění zálivky na základě naměřených dat.
- Odesílání zpráv o měření řídící jednotce skrze MQTT.
- Příjímání zpráv od řídící jednotky skrze MQTT.

Serverové řešení

Serverové řešení si klade za cíl umožnit komunikaci řídící jednotky a klientské aplikace. Dále umožnit komunikaci s databází za účelem ukládání a načítání hodnot, jako jsou údaje o provedených měření, uživatelské účty, existující řídící jednotky a existující senzorické jednotky. Z tohoto hlediska by měl server být schopen následujících funkcionalit :

- Navázání spojení s klientem.
- Obsluhovat jednotlivé požadavky, jež klienti na server zašlou.
- Přeposílat zprávy mezi klienty
- Komunikace s databází.

Software klientské aplikace

- Komunikace se serverem
- Obsluha uživatelských účtů
- Obsluha řídících jednotek
- Obsluha senzorických jednotek

4.5.2 Uživatelské scénáře

V celém systému mohou nastat události, které jsou zapříčiněny přímou interakcí uživatele se systémem :

Vytvoření uživatelského účtu

Uživatel klikne na tlačítko umožňující vytvoření účtu. Obslužná událost přiřazena k tomuto tlačítku uživatele přesune na obrazovku, která slouží k vytvoření nového účtu za pomocí zadání jména a hesla. Po kliknutí na tlačítko potvrdit obslužná událost tohoto tlačítka vezme uživatelem zadané informace, a zašle požadavek na server, zdali již tento uživatel existuje. Server příjme požadavek a vytvoří dotaz na databázi, ve které se nachází údaje o uživatelských účtech. Pokud není nalezen účet s tímto jménem, je klientské aplikaci vrácena kladná odpověď a účet je zanesen do databáze. V opačném případě je aplikaci vrácena negativní odpověď.

Přihlášení se k uživatelskému účtu

Před uživatelem je vytvořena obrazovka umožňující zadat své přihlašovací údaje, tedy jméno a heslo. Uživatel tyto údaje zadá a klikne na potvrzující tlačítko. Obslužná událost tlačítka odešle tyto údaje na server. Server přímá zprávu a kontroluje tyto údaje vůči databázi. Pokud je nalezena shoda, vrací kladnou odpověď, v opačném případě vrací negativní odpověď.

Registrace řídící jednotky

Proces registrace počíná kliknutím na tlačítko registrace nové jednotky. Obslužná událost tohoto tlačítka vytvoří okno, do kterého je možné zadat unikátní identifikátor jednotky a potvrdit kliknutím na tlačítko. Po kliknutí obslužná událost vystavuje požadavek na server, zdali tato jednotka existuje v databázi. Server vrací negativní odpověď v případě že ne. V případě že tato jednotka existuje, pokračuje klientská aplikace dalším požadavkem na server, ve kterém žádá o registraci této řídící jednotky pod aktuálně přihlášeného uživatele. Server zpracuje tento požadavek tím, že upraví záznam v databází a k řídící jednotce přiřadí jméno uživatele. Poté si klientská aplikace vyžádá od serveru všechny řídící jednotky, které jsou registrované pod právě přihlášeného uživatele, čímž dojde k aktualizaci právě zobrazených jednotek na straně uživatele.

Odregistrování řídící jednotky

Při od registraci řídící jednotky uživatel klikne na tlačítko odregistrovat u řídící jednotky, kterou si přeje odebrat zpod svého účtu. Obslužná událost zasílá požadavek na server o odregistraci řídící jednotky. Server příjme zprávu a vystaví dotaz na databázi, ve kterém upravuje záznam v databázi, kde u konkrétní řídící jednotky nastaví uživatele na prázdnou hodnotu. Stejně tak se nastaví jako prázdná hodnota pole pro přezdívku řídící jednotky.

Přejmenování řídící jednotky

Při kliknutí a tlačítko přejmenování u konkrétní řídící jednotky je vytvořeno okno, do kterého lze zadat nové jméno a tuto volbu potvrdit. Po kliknutí na potvrzení obslužná událost vystavuje požadavek na server o přejmenování. Server požadavek zpracuje a vystaví dotaz na databázi, který upraví záznam u konkrétní řídící jednotky a přidává ji přezdívku.

Zobrazení senzorů v dosahu konkrétní řídící jednotky

Kliknutím uživatele na tlačítko pro zobrazení dostupných senzorů u konkrétní jednotky je vygenerována obslužná událost, která vystaví požadavek na server o přeposlání zprávy konkrétní řídící jednotce. Ještě před tímto krokem je na server vyslán požadavek, zdali má tato řídící jednotka se serverem spojení. Server odpovídá na požadavek negativně, za předpokladu, že toto spojení nemá. V opačném případě odpovídá kladně a je z klientské aplikace je vyslán požadavek na řídící jednotku o navrácení viditelných senzorů v dosahu. Řídící jednotka přijímá požadavek a prochází jednotlivé senzory, které jsou k ní připojeny a nejsou registrované u žádné jednotky. Každý tento senzor přeposílá uživateli zpět skrze server. Poté, co řídící jednotka úspěšně odeslala všechny senzory, klientská aplikace dává na server požadavek o získání senzorů, které jsou registrované pod tuto řídící jednotku. Server vysílá dotaz na databázi, který vrací všechny senzory, jež jsou registrovány pod konkrétní řídící jednotkou. Uživatelská aplikace tento požadavek přijímá a zobrazuje senzory, které má řídící jednotka v dosahu a je možné je registrovat a na základě požadavku o registrované senzory vyobrazí ty senzory, které jsou již registrované, ale jen pod konkrétní řídící jednotkou.

Registrace senzoru

Po kliknutí na tlačítko registrace u neregistrovaného senzoru je vytvořena obslužná událost, která zasílá požadavek na server o registraci senzorické jednotky Server tento požadavek zpracuje dotazem na databázi, ve kterém přiřadí senzorické jednotce identifikátor jednotky řídící. Poté klientská aplikace posílá potvrzení o registraci senzoru řídící jednotce skrze server. Řídící jednotka na základě tohoto požadavku přidá senzor na seznam registrovaných jednotek, což způsobí nemožnost řídící jednotky vracet tento senzor při požadavku o zobrazení senzorů v dosahu a zároveň umožní senzoru publikovat naměřené hodnoty na server skrze řídící jednotku.

Odregistrace senzoru

Při kliknutí na tlačítko odregistrace u registrovaného senzoru obslužná událost zasílá požadavek na server o odregistraci senzoru. Server požadavek zpracuje pomocí zaslání dotazu na databázi, který upravuje záznam této senzorické jednotky a nahrazuje hodnotu přiřazené řídící jednotky za prázdnou hodnotu. Po úspěšném dokončení klientská aplikace zasílá skrze server oznámení řídící jednotce o odregistrování tohoto senzoru. Řídící jednotka reaguje tak, že odstraní tento senzor ze seznamu registrovaných jednotek, což způsobí možnost řídící jednotky vracet tento senzor při požadavku o zobrazení senzorů v dosahu a znemožní senzoru publikovat naměřené hodnoty na server skrze řídící jednotku.

Nastavení hranice vlhkosti pro zálivku a času zálivky senzoru

U registrovaného senzoru je možnost vyplnit pole pro zadání hodnot minimální vlhkosti a času zálivky. Kliknutím na příslušné tlačítko lze tuto hodnotu odeslat senzorické jednotce. Při kliknutí na tlačítko je vystaven požadavek na server o aktualizaci senzorické jednotky v databázi . Řídící jednotka periodicky získává aktuální údaje ze serveru, které přeposílá konkrétnímu senzoru skrze vytvoření MQTT zprávy, jejíž topic odpovídá senzoru, kterému je požadavek určen. Předmětem této zprávy je číselná hodnota udávající hodnotu pro zálivku. MQTT broker přeposílá tuto zprávu na topic, který odebírá tento konkrétní senzor. Senzor přijímá zprávu skrze MQTT a upravuje svoji hodnotu pro zálivku na základě obsahu zprávy.

Přejmenování senzoru

Při kliknutí a tlačítko přejmenování u konkrétní registrované senzorické jednotky je vytvořeno okno, do kterého lze zadat nové jméno a tuto volbu potvrdit. Po kliknutí na potvrzení obslužná událost vystavuje požadavek na server o přejmenování senzorické jednotky. Server požadavek zpracuje a vystaví dotaz na databázi, který upraví záznam u konkrétní senzorické jednotky a přidává ji přezdívku.

Zobrazení dat o zálivce

Po kliknutí na tlačítko zobrazení dat u konkrétního senzoru je vytvořeno okno, ve kterém je vyobrazen graf zobrazující průběh vlhkosti půdy v čase Data potřebná k sestrojení grafu získává klientská aplikace vystavením požadavku na server, ve kterém žádá o získání dat o zálivce konkrétního senzoru. Server je schopen tyto data poskytnout přistoupením k SQL databázi. Uživatel dále může vybrat rozsah na ose X v časovém rozmezí od-do, nebo si vybrat zdali chce vidět měření v řádu minut, hodin, či dnů. Okno nabízí uživateli možnost porovnat měření vybraného senzoru s jiným senzorem, opět za pomocí získání dat o zálivce skrze server a sestrojením další křivky v grafu.

4.5.3 Popis zvolené technologie

Řídící jednotka, Raspberry Pi, je řízena vlastním operačním systémem Raspbian, jež vychází z Debianu. Jedná se o plnohodnotný operační systém, a kterém lze programovat v mnoha různých jazycích a využít velkou škálou programovacích prostředích. V této práci bude využit programovací jazyk Java a IDE BlueJ, obojí přichází předpřipravené v původní instalaci Rasbian OS. Na tomto zařízení bude také běžet MQTT Broker, schopný zajišťovat komunikaci mezi řídící jednotkou a senzory. Jako broker byl vybrán Mosquitto, který je díky své jednoduchosti ideálním kandidátem.

Senzorickou jednotku, ESP 8266 NodeMcu, vzhledem k provedené analýze, je nejschůdnější programovat za pomocí Arduino IDE a jazyka Arduino, který vychází z C/C++, ač základní firmware ESP je vytvořený v jazyce LUA.

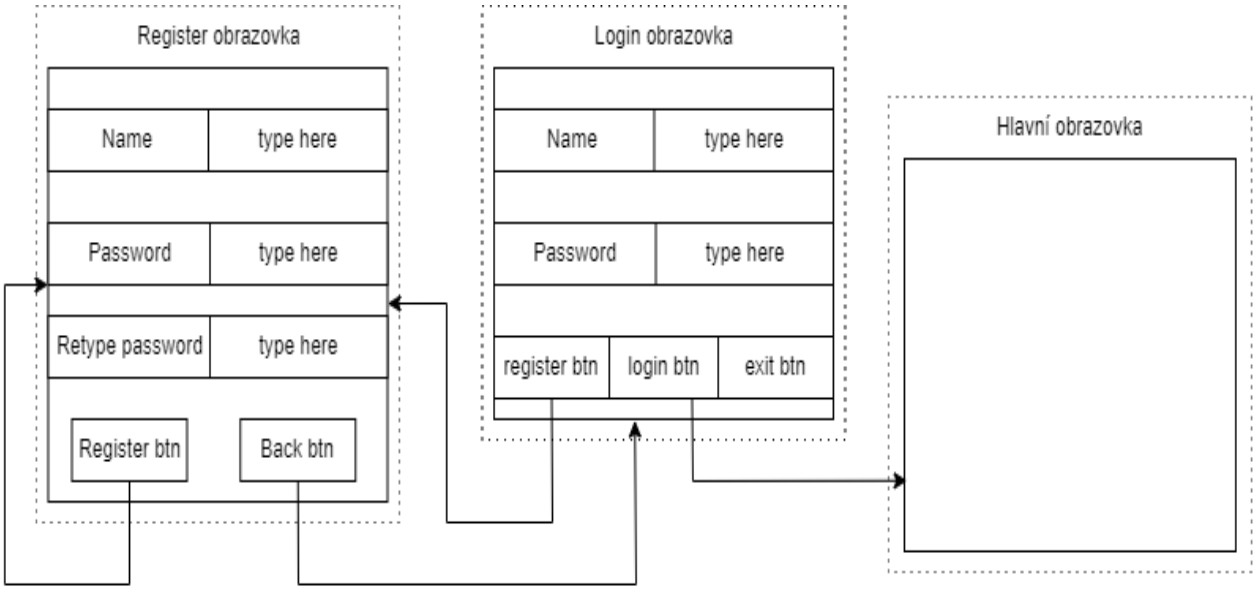
Serverové řešení bude naprogramováno v jazyce Java, stejně tak klientská aplikace, společně s jejím grafickým rozhraním, budou též napsány v Javě. Pro oba tyto subjekty bude využito programovací prostředí Netbeans. V případě klientské aplikace se bude jednat o desktopovou aplikaci.

4.5.4 Design

Grafický design klientské aplikace se bude skládat ze dvou hlavních komponent. První komponentou je přihlašovací obrazovka, jež umožňuje přihlášení ke stávajícímu účtu a vytvoření účtu nového. Druhou částí je hlavní obrazovka, obsahující řídící jednotky spolu se senzory a umožňující jejich obsluhu. Součástí GUI jsou také vyskakovací okna, do kterých může uživatel zadávat své požadavky, nebo být srozuměn s výsledkem některé jeho akce.

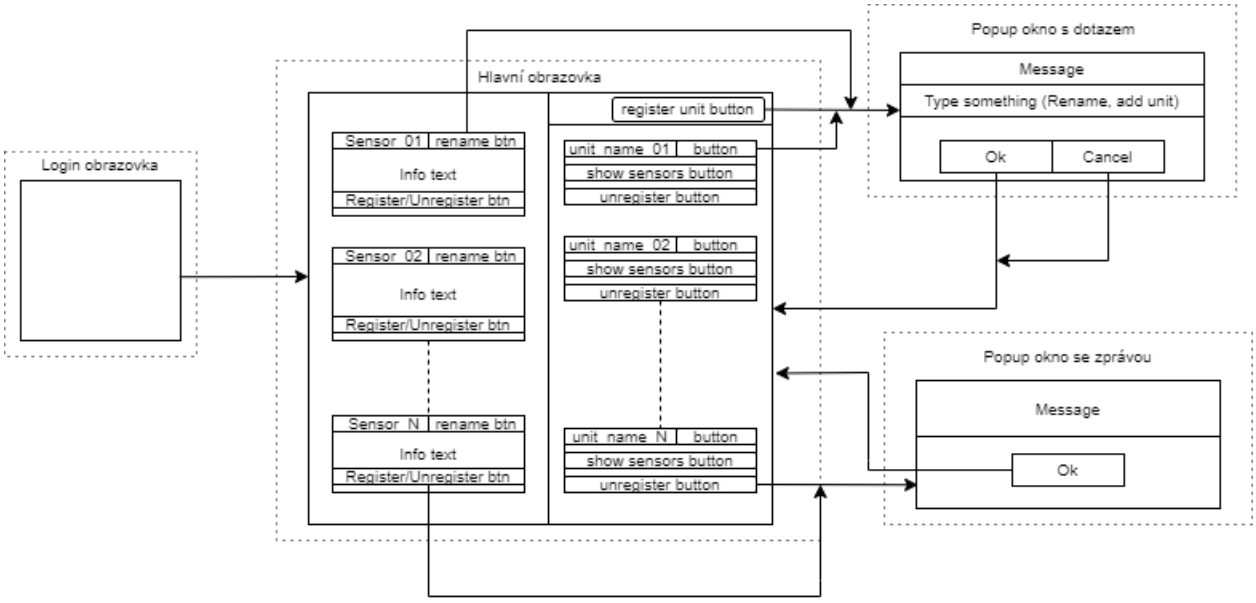
Jednotlivé šipky symbolizují přechody mezi obrazovkami, v případě vyskakujících oken platí, že toto okno překrývá obrazovku, ze které bylo inicializováno.

Na obrázku č.14 se nachází schéma znázorňující proces fungování GUI pro přihlašovací část.



Obr 14: GUI návrh pro přihlašovací část aplikace

Po kliknutí na tlačítko login a ověření ze strany serveru aplikace přechází do hlavní části, jak je znázorněno na obrázku č. 15.



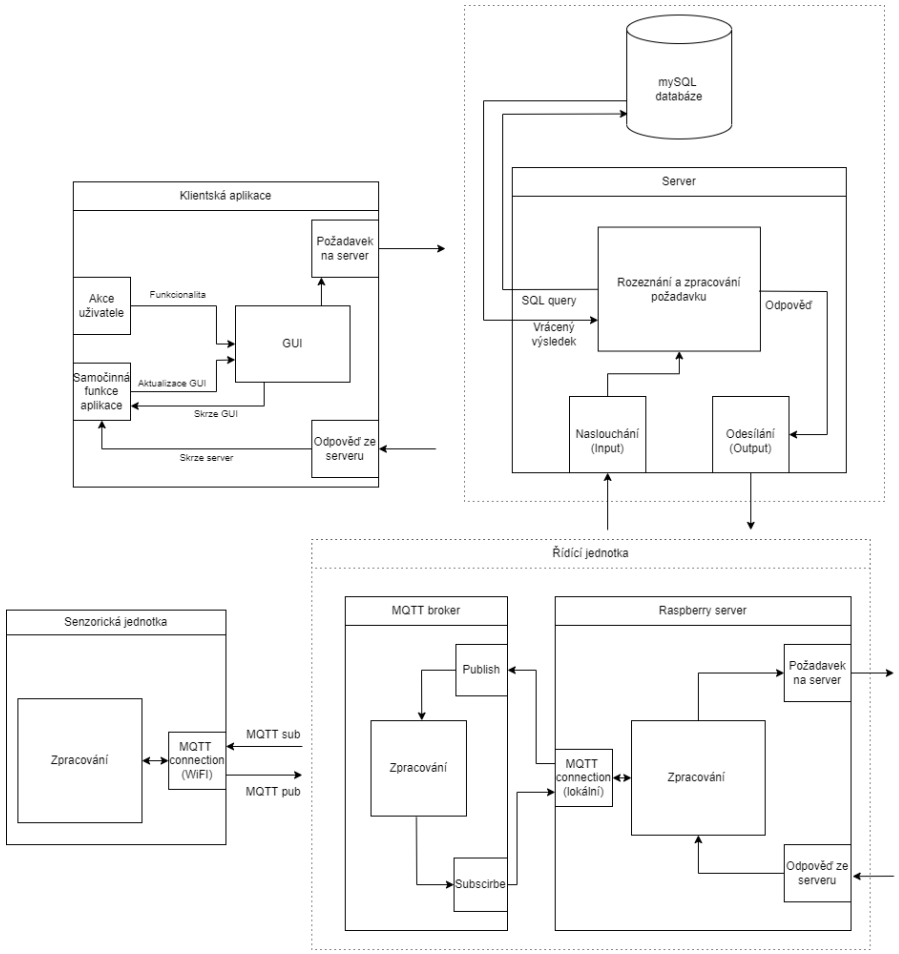
Obr 15: GUI návrh hlavní částí aplikace

4.5.5 Architektura systému

Klientská aplikace a její GUI (obr. č.15), slouží jako front-end celého systému. K zajištění správné, či vůbec nějaké funkcionality je důležité propojit funkční část systému s ostatními back-end částmi a zároveň tyto části mezi sebou. Těmito částmi myslíme server, řídící jednotky a jednotky senzorické.

Klientská aplikace se připojuje k serveru, aby mohla komunikovat se zbývající částmi systému. Server bere takovéto příchozí spojení jako svého klienta. Stejným způsobem se na server připojují i řídící jednotky, které server také považuje za své klienty. Každému klientovi, který se na server připojí, je vytvořeno jeho vlastní vlákno, na kterém probíhá vzájemná komunikace, čímž je zajištěna možnost serveru sloužit jako prostředník pro komunikaci mezi libovolným počtem klientů. Oproti tomu senzorická jednotka nemá s tímto server žádné spojení. Na místo toho je spojení navázáno s jednotkou řídící, kde tato jednotka plní funkci prostředníka mezi serverem a danou senzorickou jednotkou.

Schéma architektury systému je ukázáno na obrázku č. 16



Obr 16: Architektura systému

4.5.6 Použité technologie

Tato kapitola se zaobírá použitými technologiemi při návrhu front-end části a back-end části aplikace.

Back-end

Serverová aplikace, společně s řídící jednotkou, je naprogramovaná v programovacím jazyce Java s využitím programovacího prostředí Netbeans a JDK 11.

Back-end využívá TCP/IP pro komunikaci s jednotlivými klienty. V Javě je k tomuto účelu možno použít knihovnu java.io.* .

Komunikace serveru s databází zajišťuje knihovna mySQLConnector, kterou je také nutné manuálně vložit do projektu [24].

Pro komunikaci mezi senzorickou a řídící jednotkou je využita knihovna paho MQTT, která umožňuje komunikaci s MQTT brokerem. Tato knihovna není součástí základní instalace vývojového prostředí. Jedná se o externí knihovnu a je třeba ji manuálně naimportovat.

V případě senzorické jednotky je tato knihovna již obsažena a jedná se o knihovnu PubSubClient..

Front-end

Uživatelská aplikace je naprogramovaná v programovacím jazyce Java s využitím programovacího prostředí Netbeans a JDK 11.

Pro vytvoření GUI klientské aplikace je použit framework Swing, kde byla snaha o dodržení základů architektonického vzoru model-view-presenter. Pro vzhled GUI byla využita knihovna třetí strany Flatlaf verze 3.0.

Pro vykreslování grafů je využita knihovna JFreeChart,. Jedná se o knihovnu třetí strany, jež je volně dostupná k použití.

Komunikace se serverem je zajištěna za pomocí knihovny java.io.* .

4.5.7 Implementace

Tato část se zabývá zásadami dodržovanými při vývoji systému, implementací back-endu,jeho strukturou a implementací klientské částí systému.

Zásady vývoje

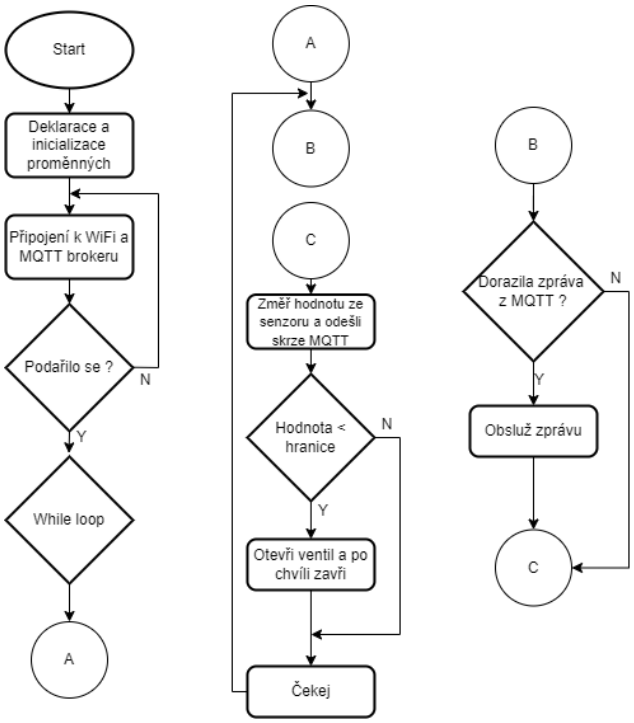
Z hlediska vývoje a struktury systému budou dodržovány následující zásady:

- Objektový přístup
- Vícevláknové zpracování a eliminace konkurence vláken

Back-end

Senzorická jednotka

První následuje deklarace a inicializace proměnných, tyto proměnné jsou povětšinou primitivního typu, kde se jedná o proměnné pro ukládání času, aktuální vlhkosti, hranice pro zálivku a proměnné definující jednotlivé piny. Je třeba také deklarovat objekty definující WiFi připojení a MQTT připojení. Průběh programu je vyobrazen na obrázku č. 17.



Obr 17: Vývojový diagram senzorické jednotky

Mikrokontrolér se nejprve pokusí o připojení k WiFi síti. Tato síť je předem daná a jedná se o access point řídící jednotky. Pokus o připojení opakuje, dokud se nepodaří.

```
int WiFiCon() {
  if (WiFi.status() != WL_CONNECTED){
      WiFi.mode(WIFI_STA);
      WiFi.begin(ssid, password);
      while (WiFi.status() != WL_CONNECTED) {
        delay(2000);
      }
      if (WiFi.status() != WL_CONNECTED){
        return 0;
      } else {
        return 1; //1 is initial connection
      }
  } else {
    return 2; //2 is already connected

  }
}
```

Fragment 1: Připojení senzorické jednotky k WiFi

Následuje obdobný proces s připojením k MQTT brokeru. Senzorická jednotka předpokládá, že broker se nachází na stejné adrese, jako je adresa WiFi access pointu. Pokud jsou potřeba k přístupu k brokeru přihlašovací údaje, senzorická jednotka je poskytne. Opět opakuje svůj pokus o připojení, dokud se ji to nepodaří.

```
void MQTTcon(){
 client.setServer(mqtt_broker, mqtt_port);
 client.setCallback(callback);
 while (!client.connected()) {
   if (client.connect(sensor_ID, mqtt_username, mqtt_password)) {
       break;
    } else {
       delay(2000);
    }
 }
}
```

Fragment 2: Připojení senzorické jednotky k MQTT brokeru

Okamžitě po připojení k brokeru jednotka publikuje na topic „acknowledge“ zprávu, jejíž obsah je ID senzorické jednotky, čímž se identifikuje a umožní řídící jednotce přidat / odebrat senzor z pro ni registrovaných či viditelných.

```
void sendAcknowledge(){
  client.publish("acknowledge",sensor_ID);
}
```

Fragment 3: Zaslání inicializující zprávy řídící jednotce skrze MQTT

Poté se pokusí o získání hodnoty ze senzoru na měření vlhkosti. Naměřená hodnota je poté normalizována do rozsahu 0 – 100.. Dále je třeba hodnoty vyšší než 100 zredukovat na hodnotu 100 a hodnoty nižší než 0 pozvednout na 0. Důvodem je nepřesnost při mapování hodnot. Naměřená hodnota je nakonec odeslána skrze metodu sendMoisture().

```
int moistureSensor(){
  data = analogRead(analogPin);
  data = map(data,dryValue,wetValue, 0, 100);
  if (data > 100){
    data == 100;
  }
  if (data < 0){
    data = 0;
  }
  sendMoisture(data);
  return data;
}
```

Fragment 4: Měření vlhkosti senzorickou jednotkou

```
void sendMoisture(int data){
   String moist;
   moist = String(data);
   char Buf[50];
   moist.toCharArray(Buf, 50);
   client.publish(topic,Buf);
}
```

Fragment 5: Odeslání naměřené vlhkosti řídicí jednotce skrze MQTT

Zbytek programu operuje v cyklu, kde po pevně dané době posílá acknowledge a provádí měření senzoru. Pokud je změřená hodnota nižší než hodnota nastavená jako hraniční, tak senzorická jednotka otevře vodní ventil.

Vodní ventil ovládaný pulsem nepotřebuje stálé napětí pro jeho otevření, stačí jeden puls. Stejně tak stačí jeden puls pro jeho zavření.

```
void water(){
  digitalWrite(waterPin,HIGH); // Otevřít ventil, 1 puls
  writeValveState(true); // zapiš stav ventilu do EEPROM
  delay(1000);
  digitalWrite(waterPin,LOW);
  delay(5000); //Doba otevření ventilu
  digitalWrite(waterPin,HIGH); // Zavřít ventil, 2 puls
  writeValveState(false); // zapiš stav ventilu do EEPROM
  delay(1000);
  digitalWrite(waterPin,LOW);
}
```

Fragment 6: Otevření vodního ventilu senzorické jednotky

Nakonec smyčky je volána funkce client.loop(), která zpracovává zprávy, jež přišli na topicy, ke kterým je senzorická jednotka subscribnutá, skrze funkci *callback()*. Tyto topicy jsou pevně dané a senzorická jednotka se k těmto topicům hlásí při inicializaci senzorické jednotky.

```
char * thresoldTopic = "sensors/999999/thresold";
...
void subscribeToTopics(){
   client.subscribe(thresoldTopic);
}

void callback(char *topic, byte *payload, unsigned int length) {
   int value;
   for (int i = 0; i < length; i++) {
      Serial.print((char) payload[i]);
      inputString += ((char) payload[i]);
    }
....
     if(String(topic).equals(thresoldTopic)){
       value = inputString.toInt();
       setThresold(value);
    }
}
```

Fragment 7: Přijmutí MQTT zprávy ohledně nastavení hranice pro otevření ventilu

Pakliže jednotka příjme zprávu na topic thresholdTopic, nastaví její obsah jako svoji novou hraniční hodnotu pomocí metody setThresold().

```
void setThresold(int givenThresold){
   thresold = givenThresold;
}
```

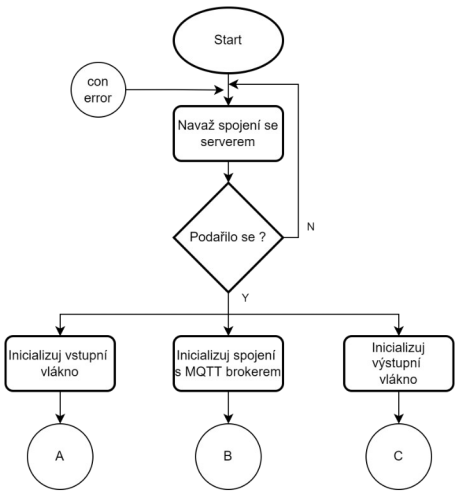
Fragment 8: Nastavení hraniční hodnoty pro zálivku senzorické jednotky

Řídící jednotka

Program se skládá z několika hlavních tříd, pokud pomineme hlavní třídu Main, která slouží jako zaváděcí bod, máme k dispozici třídu InputThread, jejíž hlavním účelem je naslouchání příchozích zpráv ze serveru, tato třída operuje jako vlákno, aby nebyl narušen další běh programu. Pro odesílání požadavků na server využíváme třídu OutputThread, která obsahuje metody odpovída-

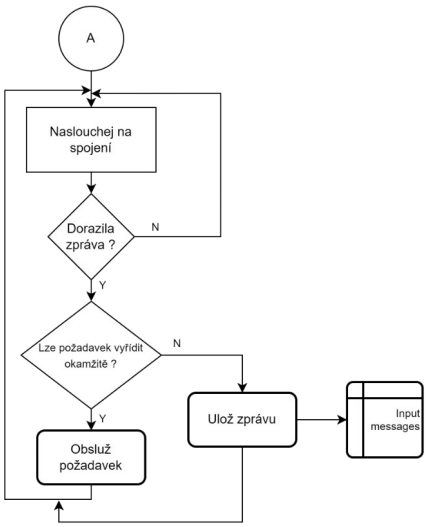
jící konkrétnímu požadavku. Poslední třídou je třída MQTTClient, jež se stará o komunikaci s MQTT brokerem.

První věcí, kterou program udělá, je pokus o připojení k serveru, který bude opakovat, dokud se úspěšně nepřipojí. Poté následuje inicializace tříd InputThread, MQTTBroker a OutputThread. (obr. č. 18)



Obr 18: Start a inicializace programu

Třída InputThread běží v nekonečné smyčce, ve které naslouchá příchozím zprávám ze serveru. Tuto příchozí zprávu může obsloužit buď ihned, tedy provést potřebné úkony, jako je například úprava hodnot proměnných, nebo odeslat odpověď serveru na základě příchozí zprávy. Druhou možností je zprávu uložit a ponechat ji pro budoucí využití.(obr. č.19) .



Obr 19: Vývojový diagram funkcionality třídy InputThread

Specifikum implementace výstupního vlákna u řídící jednotky je to, že obsahuje tyto vlákna tři. Jedno, které se stará o přijímání odpovědí ze serveru na vlastní požadavky řídící jednotky, druhé, jehož účelem je registrovat zprávy, jež přicházejí od ostatních klient a třetí, jež obsluhuje ping zprávy ze serveru,, jak je ukázáno na fragmentu Detekce výpadku řídící jednotky serverem.

Zpráva od serveru je rozbalena a na základě typu zprávy se rozhodne, zdali se požadavek obslouží okamžitě (Fragment 9), nebo zda se uloží do kolekce přijatých požadavků pro pozdější zpracování. Toto zpracování je rozebráno v kapitole Front-end

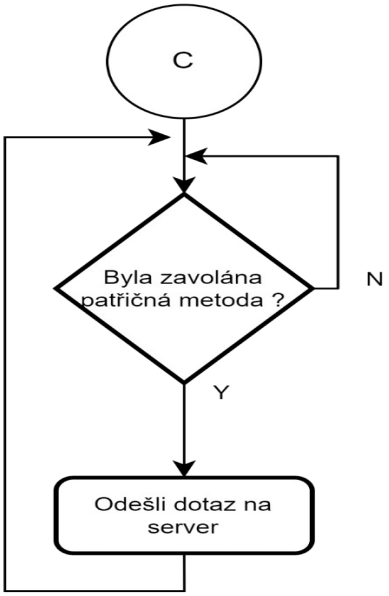
```
while(true){
   PayloadBehavior message = null;
   message = ((PayloadBehavior)objectInput.readObject());
   MessageType messageType = message.getMessagePayload().getTypeOfMessage();
   ArrayList<String> messageBody = message.getMessage();
   switch(messageType){
      case GET_SENSORS_IN_RANGE :
         manager.sendAvailableSensors(message.getDestination(), message.getSource());
      break;
      case REGISTER_SENSOR :
          manager.registerSensor(messageBody.get(0));
      break;
      case UNREGISTER_SENSOR:
          manager.unregisterSensor(messageBody.get(0));
      break;
      case IS_SENSOR_ACTIVE :
         manager.sendIsSensorActive(message.getDestination(),
         manager.isSensorActive(messageBody.get(0)),message.getSource());
      break;
      case PING :
          manager.sendTestback();
      break;
      default :
           taskQueue.put(message);
   }
}
```

Fragment 9: Zpracování zprávy od klienta řídící jednotkou

Přijímání a zpracování odpovědí ze serveru na vlastní požadavky řídící jednotky je implementováno obdobně jako klientskou aplikací - fragment Logika získání odpovědí ze serveru.

Pokud přijatá zpráva vyžaduje odpověď, tak je tato odpověď zaslána skrze implementaci třídy OutputThread.

Třída OutputThread obsahuje předdefinované metody, které po zavolání odešlou na server konkrétní požadavek. Sama o sobě žádnou jinou funkcionalitu nemá, její hlavní důvod je sloužit jako wrapper pro pohodlný přístup k metodám odesílání požadavků na server. (obr.. č. 20) .



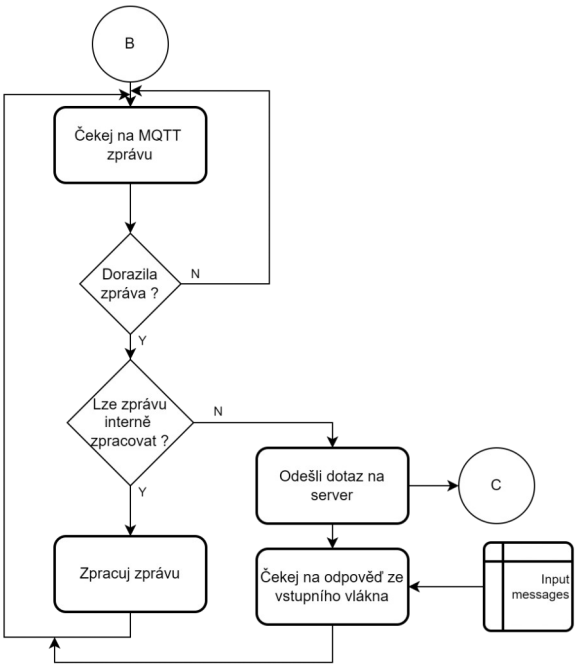
Obr 20: Vývojový diagram funkcionality třídy OutputThread

Pro využívání služeb se používá třída ServiceManager. Tato třída slouží jako wrapper pro dostupné služby a lze skrze ní přistupovat k metodám pro komunikaci. Účelem existence třídy je podpora škálovatelnosti aplikace a snížení vzájemné závislosti ostatních tříd. Níže se nachází příklady metody pro odesílání zpráv na server.

```
...
 public void sendAvailableSensors(String from,String to){
      ArrayList<String> sensors = new ArrayList();
      for(Sensor sensor : sensorsManager.getAvailableSensorsInRange()){
            if(!sensor.isIsRegistered()){
              sensors.add(sensor.getId());
            }
         }
       updateRequest.sendMessageToClient(from,MessageType.GET_SENSORS_IN_RANGE,
          sensors,to);
   }
...
```

Fragment 10: Zasílání zpráv na server řídící jednotkou

Třída MQTTBroker, jak již bylo řečeno, zajišťuje komunikaci s MQTT Brokerem. Třída využívá knihovnu paho MQTT k implementaci metod pro publikování zpráv na jednotlivé topicy, subscribování k jednotlivým topicům a příjímání zpráv, jež na tyto topicy dorazili. (obr. č. 21)



Obr 21: Vývojový diagram funkcionality třídy MQTT Client

U této třídy je důležitá zejména metoda MessageArrived(). Přepsání této metody umožňuje reagovat na zprávy, jež přicházejí na topicy registrované řídící jednotkou.

```
@Override
  public void messageArrived(String topic, MqttMessage message){
     observer.onMQTTMessageArrived(topic, message.toString(), this);
  }
```

Fragment 11: Přijmutí MQTT zprávy řídící jednotkou

Reakce na MQTT zprávu se liší na základě topicu obdržené zprávy. Ve stávajícím řešení řídící jednotka reaguje na topicy dvojího druhu. Prvním je topic acknowledge, který přichází od libovolného senzoru v okolí a umožňuje řídící jednotce identifikovat senzor - Fragment 12: Zpracování inicializační MQTT zprávy od senzorické jednotky řídící jednotkou.

```
private void processAckMessage(String msg, MQTTBroker brokerRef) throws Interrupte-
dException, MqttException{
        boolean isHidden;
        boolean doesBelong;
        String sensorOwner = getSensorOwner(msg);
        if(sensorOwner.equals("") || sensorOwner.equals("null")){
           doesBelong = false;
           isHidden = false;
        }
        else if(sensorOwner.equals(Main.unit_ID)){
           doesBelong = true;
           isHidden = true;
        }
        else{

           doesBelong = false;
           isHidden = true;
        }
        registerSensorLocally(msg,isHidden,doesBelong,brokerRef);
}
```

Fragment 12: Zpracování inicializační MQTT zprávy od senzorické jednotky řídící jednotkou

Metoda registerSensorLocally() poté přidá senzor do kolekce a nastaví mu příznak, zdali se jedná o jednotkou registrovaný senzor, či nikoliv (Fragment 13: Uložení viditelného senzoru řídící jednotkou),

```
if(!isHidden){
          sensorsManager.addAvailableSensor(sensorID);
}
else{
        if(doesBelong == false){
                sensorsManager.removeAvailableSensor(sensorID);
           }
           else{
            sensorsManager.addAvailableSensor(sensorID);
           }
 }
 if(doesBelong){
        registerSensor(sensorID);
 }
 else{
        unregisterSensor(sensorID);
 }
```

Fragment 13: Uložení viditelného senzoru řídící jednotkou

Pokud jde o senzor, který je registrovaný pod touto řídící jednotkou, je zavolána metoda registerSensor(). Metoda přidá senzoru příznak značící, že se jedná o registrovaný senzor.. Poté si zaregistruje topic, na který tento senzor odesílá údaje o naměřené vlhkosti. Nakonec jednotka odešle na topic senzoru hodnotu odpovídající uživatelem nastavené hranici vlhkosti a času zálivky. (Fragment 14: Uložení viditelného senzoru jako registrovaného řídící jednotkou).

```
for(Sensor s : sensorsManager.getAvailableSensorsInRange()){
        if(s.getId().equals(sensorID) && s.isIsRegistered() == true){
          try {
            String value = getThresold(sensorID);
            broker.getClient().subscribe("sensors/"+sensorID+"/moisture");
            broker.publishMessage("sensors/"+sensorID+"/thresold",value);
            value = getTime(sensorID);
            broker.publishMessage("sensors/"+sensorID+"/waterTime",value);
            return;
          } catch (MqttException ex) {
          } catch (InterruptedException ex) {
          }
        }
      }
```

Fragment 14: Uložení viditelného senzoru jako registrovaného řídící jednotkou

Druhým topicem, na který je jednotka schopná reagovat je moisture topic. Jednotka reaguje jen na zprávy přicházející na ty moisture topicy, ke kterým se registrovala při registraci senzoru.

Řídící jednotka na základě názvu topicu dokáže zjistit ID senzoru, který zasílá zprávu. Samotná zpráva je hodnota, udávající vlhkost půdy. K ID senzoru a vlhkosti je přidána informace o aktuálním datu a času, poté jsou tyto informace odeslány na server, kde se zapíší do databáze.

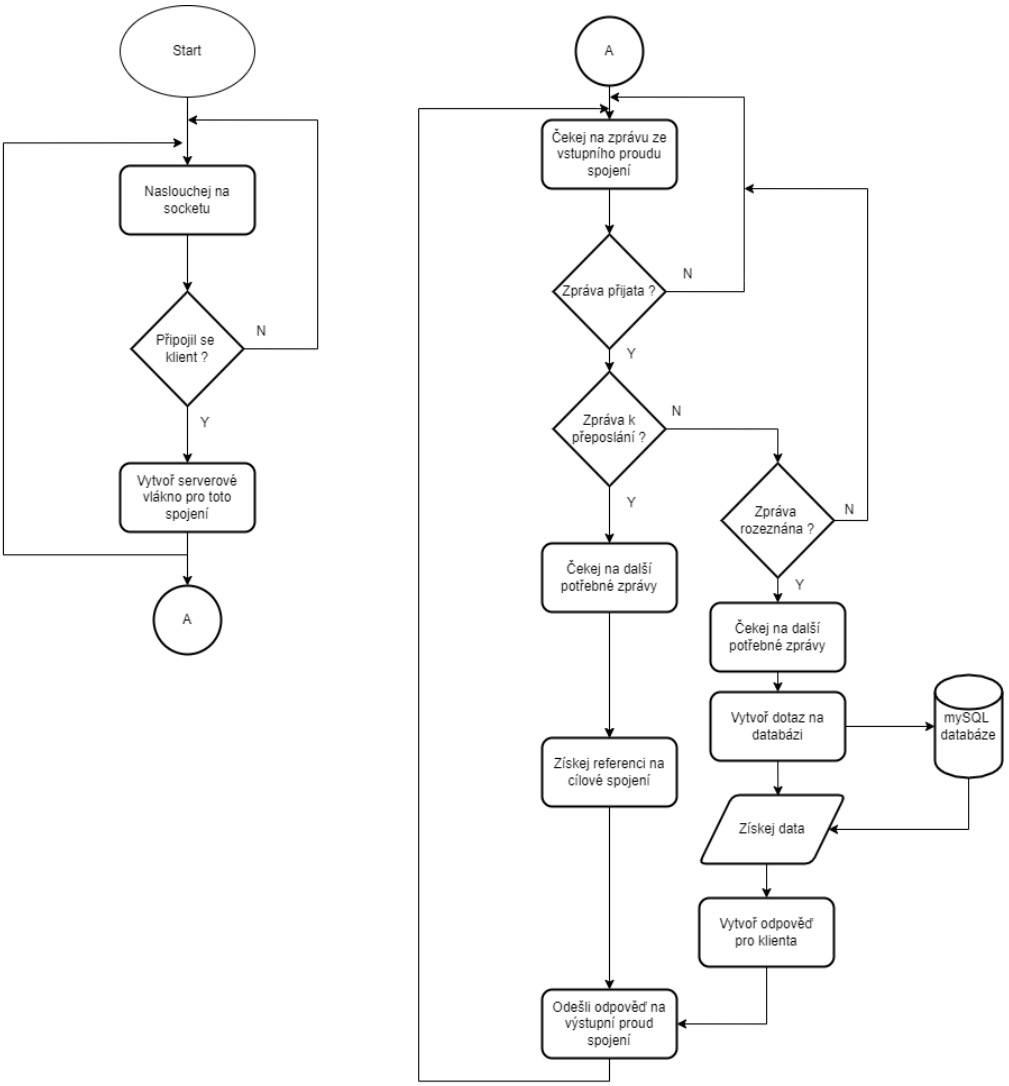
```
private void processMoistureMessage(String sensorID,String msg){
    LocalDateTime currentTime = LocalDateTime.now();
    DateTimeFormatter testFormatter =
        DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
    String formatDateTime = currentTime.format(testFormatter);
    addMeasuredValue(sensorID,msg,formatDateTime);
 }

 public void addMeasuredValue(String sensor_ID,String value,String date){
     request.addMeasuredValue(sensor_ID, value, date);
  }
```

Fragment 15: Zpracování MQTT zprávy od senzorické jednotky ohledně naměřené hodnoty vlhkosti

Server

Serverové řešení funguje na principu klient-server s využitím TCP/IP komunikace jež je založena na spojení. Druhou možností je využít UDP komunikaci, která na spojení založená není. Poněvadž potřebujeme stále spojení se serverem, bude využita komunikace za pomocí TCP. [26]

Serverová aplikace obsahuje tři hlavní třídy. Třídu Main která obstarává připojování jednotlivých klientů, inicializuje spojení a vytváří serverová vlákna třídy ServerThread pro konkrétní spojení. Třída ServerThread obsluhuje příchozí a odchozí komunikaci daného spojení. Dělá tak za pomocí cyklu, který čeká na příchozí zprávu ze vstupního proudu. Na základě přijaté zprávy poté přijímá zprávy další a volá metodu, která vystavuje dotaz na databázi. Třída nakonec vrací na výstupní proud výsledek, který je zaslán klientovi. (obr č. 22)



Obr 22: Vývojový diagram funkcionality serveru

Implementace funkcionality vytváření obslužných serverových vláken pro jednotlivé klienty je uvedena zde.

```
    try (ServerSocket serverSocket = new ServerSocket(portNumber)){
          while (true){
          Socket clientSocket = serverSocket.accept();
          DataAccess dataAccess =  new QueryManager(connection);
          ServerThread clientThread = new ServerThread(clientSocket,dataAccess);
          clientThread.start();
        }
    } catch (IOException e){
       System.err.println("Zachycena vyjímka při pokusu o naslouchání na portu: " +
        portNumber + " nebo při navazování spojení.");
  }
```

Fragment 16: Vytvoření obslužného serverového vlákna pro připojujícího se klienta

Po vytvoření klientského vlákna začne vlákno naslouchat požadavkům pro konkrétní spojení. Je důležité aby každý klient měl vyhrazené svoje serverové vlákno, pokud by tomu tak nebylo, byla by narušena kauzalita událostí pro jednotlivé klienty a navíc by nebylo možné přeposílat zprávy mezi jednotlivými klienty.

Příklad implementace serverového vlákna pro připojeného klienta je uveden níže.

```
public ServerThread (Socket clientsocket,DataAccess dataAccess) throws IOException {
    socket = clientsocket;
    objectInput = new ObjectInputStream(socket.getInputStream());
    objectOutput = new ObjectOutputStream(socket.getOutputStream());
    this.dataAccess = dataAccess;
  }
  @Override
  synchronized public void run () {
    try {
      MessagePayload message;
      while (true) {
       ...
       if(message.getTypeOfMessage().equals(MessageType.ADD_MEASURED_VALUE)){
            ID = message.getMessage().get(0);
            value = message.getMessage().get(1);
            date = message.getMessage().get(2);
            dataAccess.addMeasurmentQuery(ID,value, date);
       }
```

```
        if (message.getTypeOfMessage().equals(MessageType.ADD_USER)){
          user = message.getMessage().get(0);
          passwd = message.getMessage().get(1);
          dataAccess.addUserQuery(date, passwd);
        }
        if (message.getTypeOfMessage().equals(MessageType.GET_UNIT)){
          value = message.getMessage().get(0);
          ArrayList list = new ArrayList();
          list.add(dataAccess.getUnitQuery(value));
          sendReply(message.getTypeOfMessage(),list);
        }
       if(message.getTypeOfMessage().equals(MessageType.GET_REGISTERED_UNITS)){
          user = message.getMessage().get(0);
          sendReply(message.getTypeOfMessage(),
                        dataAccess.getRegisteredUnitsQuery(user));
        }
        ...
```

Fragment 17: Přijmutí a rozlišení typu zprávy od klienta

Po přijmutí požadavku server vykoná danou funkcionalitu. Příkladem takové požadavku může být žádost o získání řídících jednotek spadajících pod konkrétního uživatele.

Implementace zpracování tohoto požadavku je zobrazena níže.

```
 @Override
public ArrayList<String> getRegisteredUnitsQuery(String user) throws SQLException, In-
terruptedException{
            ArrayList<String> registeredUnits = new ArrayList();
            String query;
            query = " SELECT* FROM units WHERE user = " + "\"" + user + "\"";
            PreparedStatement pst;
            pst = connection.prepareStatement(query);
            ResultSet result = pst.executeQuery();

            String val = "";
            while(result.next()){
               val = result.getString("unit_ID");
               registeredUnits.add(val);
            }
            pst.close();
            return registeredUnits;
 }
```

Fragment 18: Struktura dotazu na databázi

Odpověď na požadavek klienta je odeslána skrze metodu sendReply().

```
public void sendReply(MessageType messageType, ArrayList<String> messageBody)
      getObjectOut().writeObject(new MessagePayload(messageType,messageBody));
}
```

Fragment 19: Odeslání zprávy klientovi ze serveru

Za zmínku také stojí způsob předávání zpráv mezi jednotlivými klienty, tedy mezi uživatelem a řídící jednotkou. Zpráva od klienta klientovi má vždy tvar od – zpráva – komu. Server, který si drží kolekci připojených klientů, reprezentovaných jejich serverovými vlákny je schopen zprávu, která je adresována jinému klientovi vzít a přeposlat na dané vlákno.

Pokud přichází zpráva určena jinému klientovi, její přeposlání je implementováno následovně.

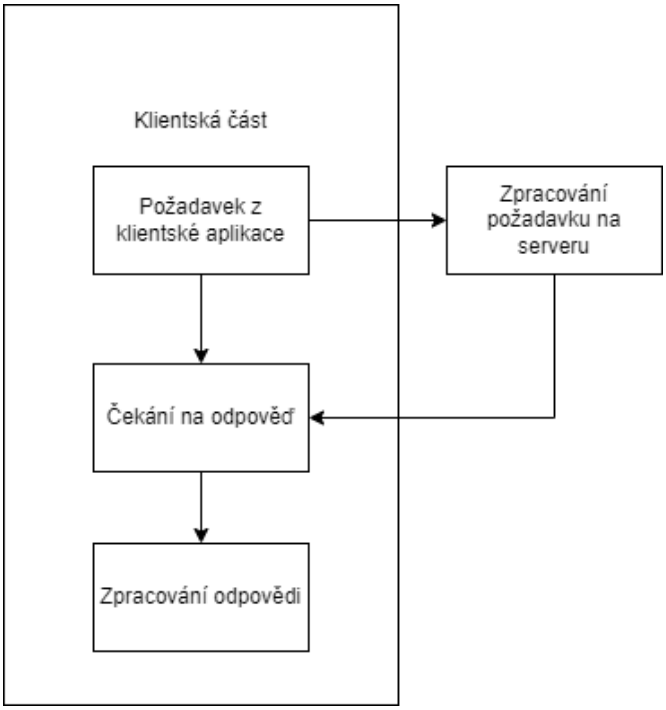
```
if(message.getDestination() != null){
  ClientConnection c = ConnectedDevices.getAnyConnection(message.getDestination());
   if(c == null){
       sendMessageToClient(this.getName(),null,new          MessagePayload(Message-
Type.ERROR,new ArrayList()));
    }
    else{
        ServerThread s = (ServerThread)c.getUpdateThread();
          if(s == null){
                s = (ServerThread) c.getOutputThread();
                s.sendReply(MessageType.ERROR,new ArrayList());
           }
           else{
                sendMessageToClient(message.getDestination(),message.getSource(),new
                   MessagePayload(message.getTypeOfMessage(),message.getMessage()));
           }
     }
     continue;
}
...
```

Fragment 20: Přeposlání zprávy od klienta jinému klientovi

Front-end

Implementace klientské části tkví v zajištění komunikace se serverem, která, stejně jako je tomu u řídící jednotky, funguje na principu vstupního a výstupního vlákna.

Aplikace vystavuje na popud uživatele požadavky na server a dále očekává odpověď ze serveru - obrázek č.23.



Obr 23: Funkcionalita klientské aplikace vůči serveru

Požadavek z klientské aplikace může mít mnoho podob v závislosti na tom, co se od serveru požaduje. V následujícím příkladu bude podrobně rozebrán požadavek aplikace na získání registrovaných jednotek uživatele. Všechny požadavky na server se řídí stejnou logikou, jejíž implementace je vidět níže.

Požadavky na server se odesílají skrze modelovou třídu ServiceManager. Metody této třídy zajišťují jak odeslání požadavku, tak přijmutí odpovědí.

```
public ArrayList<UnitObject> getRegisteredUnits(String username) throws InterruptedE-
xception{
      ArrayList<UnitObject> units = new ArrayList();
      request.getRegisteredUnits(username);
      for (String unit : getMultipleResponses(MessageType.GET_REGISTERED_UNITS)){
         UnitObject unitObject = new UnitObject(unit);
         unitObject.setNickname(getUnitNickname(unit));
         units.add(unitObject);
      }
      return units;
 }
```

Fragment 21: Požadavek uživatelské aplikace na server

Odeslání samotného požadavku skrze instanci třídy OutputThread, probíhá pomocí odesílání dat na výstupní proud.

```
public void getRegisteredUnits(String username){
      content = new ArrayList();
      content.add(username);
      sendPayload(MessageType.GET_REGISTERED_UNITS,content);
 }

private void sendPayload(MessageType typeOfMessage,ArrayList<String> messages){
       objectOutput.writeObject(new MessagePayload(typeOfMessage,messages));
}
```

Fragment 22: Odeslání požadavku uživatelské aplikace na server

Aplikace poté čeká, dokud ze serveru nedorazí odpověď.

```
@Override
 public synchronized List<String> getComplexAnswer(MessageType type) throws In-
terruptedException{
      List<String> message = (List<String>) getMessageAnswer(type).getMessage();
      return message;
}
```

Fragment 23: Přijmutí více odpovědí zaslaných serverem

Namísto metody getComplexAnswer() vracející kolekci zpráv v odpovědi, je možné použít metodu getAnswer(), která vrací pouze první zprávu v odpovědi.

```
public String getAnswer(MessageType messageType) throws InterruptedException{
      synchronized(this){
         return getMessageAnswer(messageType).getMessage().get(0);
      }
   }
```

Fragment 24: Přijmutí jedné odpovědi zaslané serverem

K samotnému přijímání zpráv dochází pomocí naslouchání na vstupním proudu ze serveru. Pokud dorazí nějaká odpověď, je uložena do kolekce.

Příklad implementace této třídy je uveden zde.

```
public void run(){
    try {
        objectInput = new ObjectInputStream(clientSocket.getInputStream());
        while(true){
            PayloadBehavior payload = (PayloadBehavior) objectInput.readObject();
            taskQueue.add(payload);

        }
    } catch (IOException | ClassNotFoundException ex) {
        Logger.getLogger(InputThread.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Fragment 25: Logika získání odpovědí ze serveru

4.5.8 Nasazení do provozu

Pro nasazení do provozu je nutné připojit senzorické jednotky k řídící jednotce. Pro možnost jejich připojení přes WiFi je nutné nakonfigurovat jednotku jako access point a spustit službu hostapd na pozadí. Poté se senzorická jednotka automaticky připojí na dostupný interface řídící jednotky.

Pro zajištění vzájemné komunikace je nutné spustit při startu řídící jednotky MQTT Broker, společně s výkonnou aplikací pro obstarávání funkcionality komunikace se serverem a jednotlivými senzorickými jednotkami.

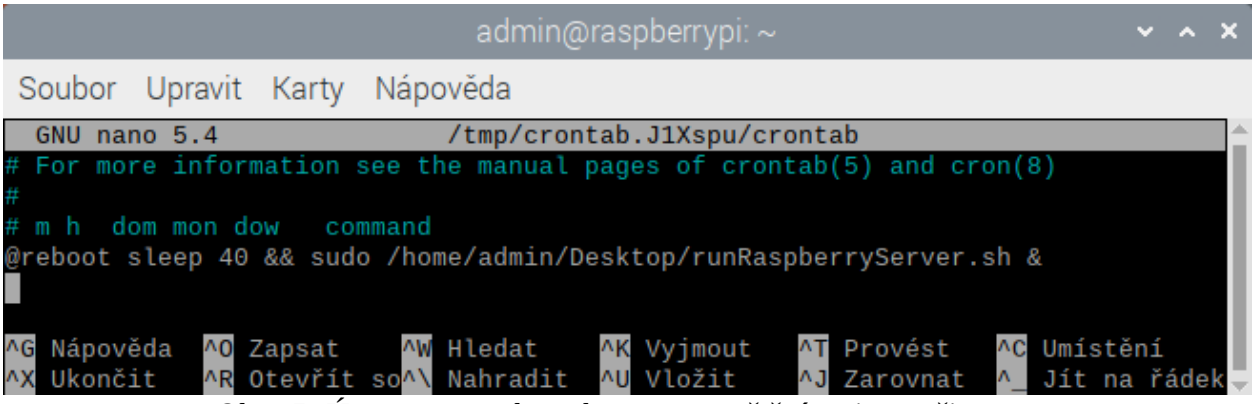
Pro automatický start obslužného programu je nutné napsat skript, který zajistí spuštění programu.

runRaspberryServer.sh

```
1  #!/bin/bash
2  sudo systemctl restart hostapd
3  cd /home/admin/Desktop
4  java -jar RaspberryServer &
5  echo "script started" >> /home/admin/Desktop/log
6
```

Obr 24: Spouštěcí script řídící jednotky

. Tento script je poté spouštěn při startu skrze crontab, jak je vidět na obrázku č.25.



Obr 25: Úprava crontab souboru pro spuštění scriptu při startu

Aby řídící jednotka mohla plnit svoji funkci, je nutné zajistit připojení k serveru. V reálném provozu je velice pravděpodobné, že se server, řídící jednotka i klientská aplikace budou nacházet v různých sítích.

Z tohoto důvodu je nutné, aby síť, ve které se nachází zařízení, na kterém server běží, mělo veřejnou IP adresu. Ve zdrojovém kódu řídící jednotky je poté potřeba přepsat IP adresu serveru, se kterým jednotka komunikuje – obr. č. 26. Stejný proces je nutný i u klientské aplikace.- obr. č.27

```
static final String unit_ID = "1010101";
static final String hostName = "192.168.0.183";

public static void main(String[] args) throws InterruptedException {
```

Obr 26: Změny ve zdrojovém kódu řídící jednotky pro úspěšné nasazení do provozu

```
public class Program {
    private ServiceManager manager;
    final private String hostName = "192.168.0.170";
    public Program(){
```

Obr 27: Změny ve zdrojovém kódu klientské aplikace pro úspěšné nasazení do provozu

Dále je serveru nutné říct, kde najde databázi a jaké jsou přihlašovací údaje.

```
public class Program {
    private final String databaseURL = "jdbc:mysql://localhost/bak_prace_db?allowMultiQueries=true";
    private final String user = "root";
    private final String password = "root";
```

Obr 28: Změny ve zdrojovém kódu serveru pro úspěšné nasazení do provozu

Pro provoz je nezbytné, aby každá řídící a senzorická jednotka měli své unikátní ID. Toto ID je napevno dáno zdrojovým kódem a umožňuje rozlišit jednotlivé senzorické a řídící jednotky – obr č. 26 a 29.

```
char* sensor_ID = "999999";
char* moistureTopic = "sensors/999999/moisture";
char* thresoldTopic = "sensors/999999/thresold";
char* waterTimeTopic = "sensors/999999/waterTime";
```

Obr 29: Změny ve zdrojovém kódu senzorické jednotky pro úspěšné nasazení do provozu

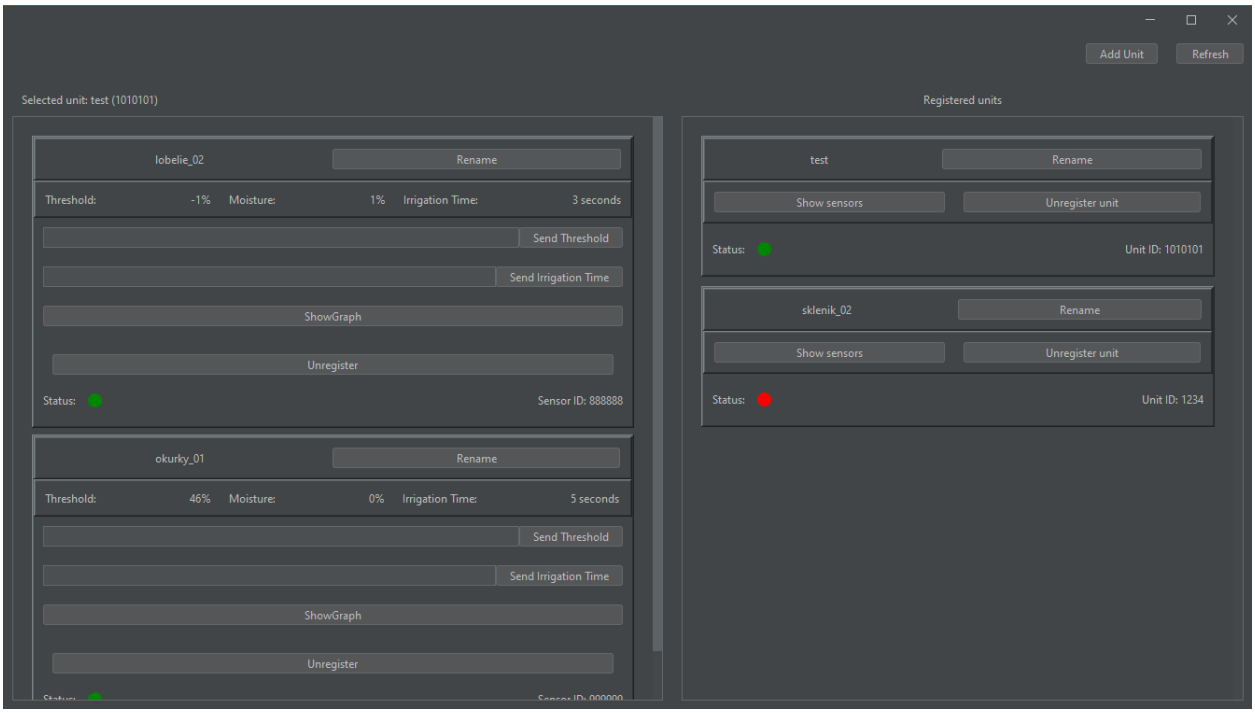
Pro účely testování se senzorická jednotka připojovala k domácí síti. Aby se byla senzorická jednotka připojit k řídící jednotce, je nezbytné změnit název a heslo sítě, se kterou se senzorická jednotka bude snažit navázat spojení. Dále je nutné definovat IP adresu, za kterou se nachází MQTT broker – obr. č. 30.

```
//const char* ssid = "IrrigationClient";
//const char* password = "lP43ca64ErZ617";
//const char *mqtt_broker = "192.168.4.1";
```

Obr 30: Změna přihlašovacích údajů k WiFi ve zdrojovém kódu senzorické jednotky

Po splnění výše uvedených požadavků mohou být jednotlivé části systému nasazeny do provozu.

Příklad využití systému z pohledu uživatele je vyobrazen na obrázcích č. 32 a 31. Více ohledně fungování systému se lze dočíst v průvodci klientskou aplikací (Příloha B)



Obr 32: Hlavní okno klientské aplikace

Obr 31: Okno grafu zobrazující průběh zálivky senzoru

5. Testování

Testování systému bylo provedeno v domácích podmínkách za využití dvou senzorických jednotek, jedné jednotky řídící a dalších dvou řídících jednotek, které byly realizovány jako pouhé záznamy v databázi.

Databáze a serverová aplikace běžely na lokálním domácím serveru.

Test systému za pomocí více instancí řídící jednotky umožnil řádně otestovat funkcionalitu modularity systému za pomocí přidávání a odebrání senzorických jednotek k již zmíněným řídícím jednotkám a registrování, potažmo odregistrování řídících jednotek z pod jednotlivých uživatelů skrze klientskou aplikaci.

5.1 Testování dosahu WiFi signálu řídící jednotky

Toto testování zahrnuje sílu přijatého WiFi signálu senzorickou jednotkou v závislosti na její vzdálenosti od řídící jednotky. Na základě výsledků testování lze určit plochu, kterou je schopna řídící jednotka pokrýt.

Sílu signálu přijímaného senzorickou jednotkou lze určit pomocí volání WiFi.RSSI().

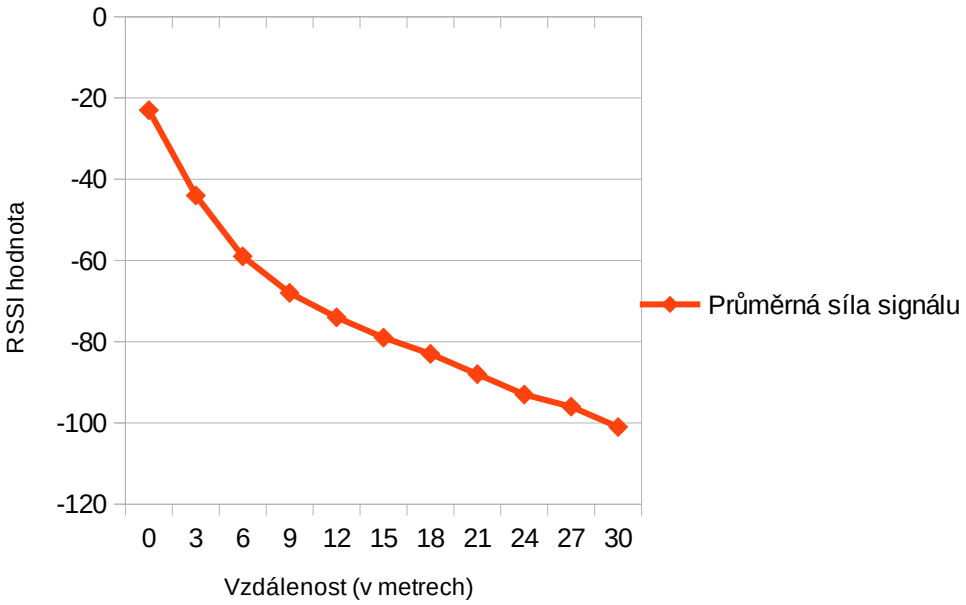
```
void loop() {
  ...
 Serial.printf("RSSI: %d dBm\n", WiFi.RSSI());
  ...
```

Fragment 26: Získání hodnoty RSSI senzorickou jednotkou

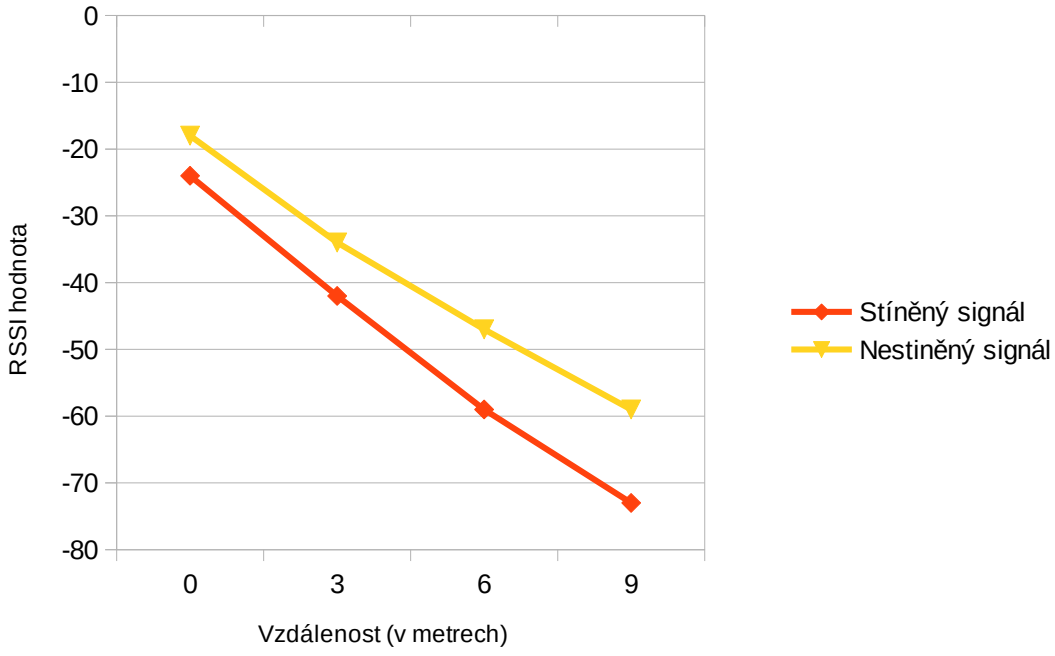
Toto volání vrací hodnotu představující indikátor intenzity signálu.

Obr 33: Výstup senzorické jednotky při měření intenzity přijímaného WiFi signálu

Testování bylo provedeno v částečně stíněném prostředí, kde mezi řídící a senzorickou jednotkou byla řada překážek, senzorická jednotka měla jen částečný výhled na zdroj signálu, při vzdálenostech nad 9 metrů byla tato viditelnost omezena úplně.



Graph 1: Závislost síly přijatého signálu na vzdálenosti od řídící jednotky



Graph 2: Vliv přímého stínění na sílu přijatého signálu

5.2 Testování stability systému při výpadku některé z jeho částí

Testování stability systému spočívá v nečekaných ukončeních spojení mezi jednotlivými částmi systému, na které postižená část nemůže reagovat. Jedná se například o odpojení řídící jednotky od napájení nebo výpadek serveru.

5.2.1 Výpadek senzorické jednotky

Řídící jednotka spravuje kolekci viditelných senzorů kolem sebe. Pokud řídící jednotce nepřijde v konečném čase několika vteřin acknowledge MQTT zpráva od jednotky v této kolekci, je z kolekce vyřazena. Jakmile se senzor stane opět aktivním, tak začíná posílat acknowledge zprávy a řídící jednotka na ně reaguje přidáním senzoru zpět do kolekce.

```
while(true){
        Thread.sleep(1000);
        for (Sensor s : linkedSensors){
          if(s.getTimeStamp() > timeMax){
            s.setIsActive(false);
            s.setTimeStamp(0);
            removeLinkedSensor(s.getId());
            break;
          }
          else{
             s.setTimeStamp(s.getTimeStamp() + 1);
             s.setIsActive(true);
          }
        }
    }
```

Fragment 27: Udržování aktivních senzorů řídící jednotkou

5.2.2 Výpadek řídící jednotky

Server si drží údaje o připojení jednotlivých klientů (uživatelů a řídících jednotek). Pokud je spojení ze strany řídící jednotky ukončeno bez zavření jejího socketu, tak je pro server jednotka i nadále aktivní, i když tomu tak není. Toto vede k nemožnosti obsloužení klientských požadavků, jež jsou na tuto jednotku adresovány a takový klient marně čeká na odpověď od serveru. Zvolené řešení spočívá v neustálém odesílání ověřovacích požadavků ze serveru.

```
while(true){
      try {
         sleep(10);
         for(ClientConnection s : ConnectedDevices.getAllUnitsConnections()){
               s.getAckThread().socket.setSoTimeout(300);
               s.getAckThread().sendReply(MessageType.PING, new ArrayList());
         }
        } catch (InterruptedException ex) {
        } catch (SocketException ex) {
            System.out.println("error");
         }
 }
```

Fragment 28: Detekce výpadku řídící jednotky serverem

Řídící jednotka musí na každou zprávu ping odpovědět.

```
while(true){
       message = ((PayloadBehavior)objectInput.readObject());
       MessageType messageType = message.getMessagePayload().getTypeOfMessage();
       ….
      switch(messageType){
               case PING →  manager.sendPong();
        ….
}
```

```
public void sendPong(){
      content = new ArrayList();
      sendMessageToServer(getMessagePayload(MessageType.PONG,content));
   }
```

Fragment 29: Odeslání ping zprávy serverem

Není důležité, aby server tuto zprávu jakýmkoliv způsobem po přijetí zpracoval, důležité je, aby na serverové vlákno dorazila jakákoliv odpověď, než dojde k timeoutu - Fragment 29: Odeslání ping zprávy serverem.

Ze strany senzorické jednotky přináší výpadek nemožnost navázání spojení přes WiFi s řídící jednotkou. Senzorická jednotka to řeší opakováním pokusu o připojení, dokud se ji nepodaří spojení s řídící jednotkou opět navázat.

```
void onNetworkError(){
  WiFiCon();
  MQTTcon();
  subscribeToTopics();
  client.loop();
}

void loop() {
 if (WiFi.status() != WL_CONNECTED){
    onNetworkError();
 }
...
```

Fragment 30: Detekce a znovupřipojení senzoru při výpadku řídící jednotky

5.2.3 Výpadek serveru

Pakliže se řídící jednotka není schopná připojit k serveru, detekuje tento výpadek v naslouchajícím vlákně.

```
class InputThread extends Thread{
   @Override
   public void run(){
         while(true){
            try{
               //Spojení přerušeno
            }
            catch(IOException | InterruptedException ex){
               callback.onConnectionLost();
            }
      }
}
```

Fragment 31: Detekce výpadku serveru řídící jednotkou

Po detekci výpadku se řídící jednotka pokouší navázat spojení se serverem, dokud se jí to nepodaří

```
static void tryConnectionAgain() throws IOException, ClassNotFoundException{
      try {
         main(new String[0]);
      } catch (InterruptedException ex) {
      }
   }
```

Fragment 32: Znovupřipojení řídící jednotky k serveru

Klientská aplikace na výpadek reaguje ve svém naslouchajícím vlákně vyhozením chybové hlášky a ukončením.

```
catch (IOException ex) {
         textPopup p = new textPopup("Connection error");
         System.exit(0);

}
```

Fragment 33: Reakce uživatelské aplikace na výpadek serveru

Na základě implementované funkcionality můžeme testovat stabilitu celého systému při výpadku jeho jednotlivých částí.

6. Diskuze

V ideálních podmínkách plně otevřeného a nestíněného prostředí je udávaný dosah signálu, který dokáže Raspberry Pi 3 B+ vydávat až 300 metrů. Záleží ovšem na zařízení, které se snaží daný signál přijmout. V případě Espressif NodeMcu 8266 je tato vzdálenost mezi 50 – 80 metry.

Ideálními podmínkami se myslí takové podmínky, kdy není přítomno žádné rušení a jak zdroj signálu tak i jeho příjemce mají jeden na druhého přímý výhled. Testování bylo provedeno v podmínkách, které těmto ideálním neodpovídají. Jedná se o vnitřní prostory bytu, kde je přítomno jak rušení z jiných zdrojů, tak stínění pomocí překážek různé velikosti (nábytek, stěny, apod.)

Hodnota RSSI	Síla signálu
>= -73	dobrá
-74 až - 85	střední
-86 až -94	slabá
-95 až -105	velmi slabá
< -106	hraniční

Tabulka 1: Referenční tabulka intenzity signálu

Měření bylo prováděno po každých 3 metrech. Poněvadž hodnota RSSI není stálá a je ovlivněna několika různými faktory, tak výsledek reprezentující RSSI při určité vzdálenosti byl získán pomocí zprůměrování dílčích výsledků.

Naměřené hodnoty RSSI (graf Závislost síly přijatého signálu na vzdálenosti od řídící jednotky) lze porovnat s referenční tabulkou nad. Můžeme tedy říci , že ve vnitřních prostorách lze zajistit spolehlivé spojení mezi senzorickou jednotkou a řídící jednotkou na vzdálenost kolem 20 metrů. Tato hodnota je ovlivněna překážkami, jež stojí mezí zdrojem signálu a jeho příjemcem (graf Vliv přímého stínění na sílu přijatého signálu).

Pokud porovnáme toto měření s hodnotami uváděnými výrobcem zařízení, můžeme očekávat, že tato vzdálenost bude v alespoň částečně ideálním prostředí vyšší.

Intenzita signálu vzhledem rostoucí vzdálenosti však není dostatečná z důvodů nutnosti pokrytí co největších ploch co nejmenším počtem řídících jednotek. Bylo by tedy na místě vybavit minimálně senzorickou jednotku přídavnou anténou, která by dokázala přijímat signál na větší vzdálenost, nežli ta integrovaná. Pro maximální účinek by bylo zároveň vhodné nainstalovat přídavnou anténu i na řídící jednotku, aby se maximalizoval dosah vysílaného signálu a tím pádem i pokrytí plochy.

Dalším řešením je přidání mezičlenů v podobě extenderů signálu, nebo použití jednotlivých senzorů jako přenašečů signálu ostatních v síti typu mesh. [10].

Testování ohledně výpadků jednotlivých částí systému (5.2 Testování stability systému při výpadku některé z jeho částí) si kladlo za cíl zjistit, zdali systém dokáže přečkat výpadky, které mohou nastat při dlouhodobém provozu. Tyto výpadky, ač z pohledu systému nečekané, mohou být z pohledu koncového uživatele i cílené, jako je například odpojení senzoru či řídící jednotky od napájení z důvodů přesunu.

Vzhledem k výsledkům tohoto testování (tabulka Reakce jednotlivých částí systému při výpadku konkrétní části) lze říci, že systém je stabilní a jednotlivé části dokáží reagovat na výpadky ostatních.

Reakce části	**Výpadkem postihnutá část systému**			
	Klient	**Server**	**Řídící jednotka**	**Senzorická jednotka**
Klient	*ukončení*	*ukončení*	*žádná*	*žádná*
Server	*Odstaněni reference na spojení*	*restart*	*Odstaněni reference na spojení*	*žádná*
Řídící jednotka	*žádná*	*Opětovný pokus o navázání spojení*	*restart*	*Odstanění reference na senzorickou jednotku*
Senzorická jednotka	*žádná*	*žádná*	*Opětovný pokus o navázání spojení*	*restart*

Tabulka 2: Reakce jednotlivých částí systému při výpadku konkrétní části

7. Závěr

V teoretické části byly identifikovány základní stavební kameny systému pro automatickou zálivku za pomocí analýzy jak běžně využívaných systémů, tak i řešeních zohledňujících embedded systémy – kapitola 2.2 Analýza dostupných řešení. Tato analýza přinesla odborný přehled v oblasti využívaných metod a postupů nejen při automatizaci zálivky, ale zejména při komunikaci jednotlivých částí celého systému.

Na základě předchozí analýzy bylo možné sestrojit návrh systému, který by splňoval cíle práce – kapitola 3.1 Obecná konstrukce řešení. Získaný přehled v oblasti využití embedded systémů poté umožnil jasně definovat, jaké prvky bude takový systém obsahovat – kapitola 4.1 Podrobná konstrukce řešení.

Vzhledem k návrhu systému jako plně automatizovaného bylo nutné řešit i způsob napájení senzorických a řídících jednotek. Nejvhodnějším řešením se jevilo napájet senzorické jednotky z vlastního zdroje a jednotky řídící z elektrické sítě - kapitola 4.1.3 Napájení. V rámci výsledného řešení bylo upuštěno od napájení senzorické jednotky skrze akumulátor z důvodů poměrně velké spotřeby elektrické energie (150 – 215 mAh). V případě úpravy senzorické jednotky pro tuto možnost by bylo nutné měřit stav baterie a přeposílat tyto údaje skrze řídící jednotku na server. Zde přichází vhod fakt, že zvolená senzorická jednotka obsahuje více pinů a nebyl by tedy problém senzorickou jednotku modifikovat pro zajištění této funkcionality. Navíc lze nyní volný 5V pin využít k dodání potřebného proudu při napájení ventilu.

Komunikace mezi řídící jednotkou a senzory je rozebrána v kapitole 4.2 Dálková komunikace. Zprvu jasnou volbou se zdálo využít dálkové komunikace přes WiFi za pomocí MQTT protokolu. Důvodem pro využití MQTT byla možnost jednoduchého zasílání zpráv řídící jednotce, na základě kterých je řídící jednotka schopná určit, o které další zprávy má zájem.

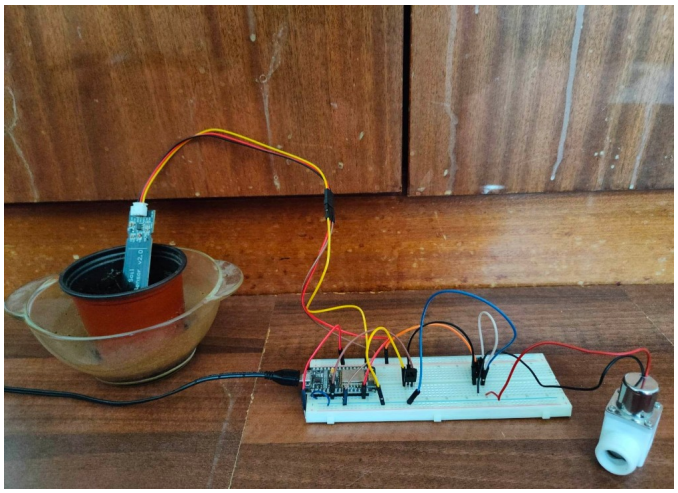
Pro vzájemnou komunikaci uživatele se systémem bylo nutné vytvořit klientskou aplikaci společně se serverovým řešením. Server slouží pro vykonávání požadavků klientské aplikace a řídící jednotky nad databází a také ke komunikaci mezi řídící jednotkou a klientskou aplikací. Softwarovým řešením obou těchto prvků, společně s implementací funkcionalit řídící a senzorické jednotky se zabývá kapitola 4.5 Software.

Testování systému probíhalo v domácích podmínkách pomocí klientské aplikace, skrze kterou lze celý systém ovládat. Scénáře, jež byly tímto způsobem testovány jsou uvedeny v kapitole 4.5.2 Uživatelské scénáře. Testování proběhlo úspěšně a systém je schopen plnit požadavky z hlediska škálovatelnosti.

Na závěr byly provedeny testy stability celého systému a testy zjišťující pokrytí řídící jednotky. Způsob provedení testů lze najít v kapitole 5. Testování. Výsledky těchto testů jsou dále rozebrány v kapitole 6. Diskuze.

Celá problematika systému pro automatickou zálivku je složitější než se zdá. Je tomu tak z důvodů dílčích problematik, jako je napájení, ukládání dat, vzájemná komunikace klientů skrze server nebo správa uživatelských účtů. Všechny tyto problematiky jsou do jisté míry řešeny v této bakalářské práci, avšak jejich optimální řešení by bylo vhodným předmětem pro návrh vylepšení stávajícího řešení.

Velkou oblastí, ve které se musí systém budoucna zlepšit, je fyzická implementace. V aktuálním řešení není zastoupena žádná ochrana senzorických a řídících jednotek před vnějšími vlivy prostředí, jako je například teplo či vlhkost, jak je vidět na obrázku Prototyp senzorické jednotky. Řešením by bylo navrhnout a sestrojit vhodný kontejner pro senzorickou a řídící jednotku, jež by řešil problémy spojené s vnějšími vlivy.



Obr 34: Prototyp senzorické jednotky

Stejně tak by bylo vhodné přidat senzorickým jednotkám LED indikátory, signalizující aktivní stav. Ten lze nyní zkontrolovat pouze v klientské aplikaci - Příloha B.

Ohledně ukládání dat by bylo možné systém do budoucna rozvinout pomocí možnosti ukládat naměřené hodnoty přímo do zařízení uživatele či řídící jednotky, čímž by bylo možné omezit celkovou velikost databáze. Vzhledem k používání uživatelských účtů je zde nevyužitý potenciál v podobě možnosti notifikace uživatele o událostech, které mohou v systému nastat. Jako příklad lze zmínit možnost zasílat uživateli oznámení na email jež zadal při registraci. Tyto oznámení by mohli být různá, od souhrnu denního měření, až po varování, pokud bude některá komponenta systému delší dobu mimo provoz. Samotné ukládání přihlašovacích údajů v databází by bylo vhodné řešit za pomocí hashovací funkce, namísto jejich textové podoby.

Naměřená data nabízí více možností, jak je zpracovat. Z posbíraných hodnot během určitého časové úseku by bylo možné odhadnout trend, kterým by se mohla zálivka řídit. Krom měření dat ohledně vlhkosti půdy by bylo možné měřit i další veličiny, jako je například průtok vody, na základě čehož by bylo možné určit denní spotřebu nejen určité senzorické jednotky, ale i celé oblasti spadající pod určitou řídící jednotku.

Dalšího zlepšení, tentokrát v oblasti škálovatelnosti zdrojového kódu, lze docílit pomocí zasílání objektů jako odpovědí klientům na jejich požadavky. Z hlediska pohodlí uživatele by bylo také vhodné implementovat vlákno, jež by se staralo o automatickou aktualizaci GUI prvků.

Vzhledem k výsledkům testování je možné říct, že i přes nedostatky v podobě nutnosti napájení senzorických jednotek z elektrické sítě a v základu malého pokrytí jednou řídící jednotkou, je celý systém modulární a škálovatelný z pohledu pokrývatelné plochy,čímž, společně s tím, že plní svůj účel automatizované zálivky, splnil cíle stanovené v kapitole 1.1 Cíle práce.

Seznam literatury

[1] *American Meadows* [online]. 2021, [cit. 3.2.2022] AmericanMeadows.com. Dostupné z: https://www.americanmeadows.com/how-hot-is-too-hot-for-plants

[2]*Noble Research Institute*. Soil and Water Relationships. [online], 2020. [cit. 3.2.2022] Jeff Ball. Dostupné z: https://www.noble.org/regenerative-agriculture/soil/soil-and-water-relationships

[3] *Tropf-Blumat.* [online], 2022. [cit. 25.2.2022] Troph-Blumat.cz Dostupné z: https://www.tropf-blumat.cz/

[4] *Hozelock*. [online], 2021. [cit. 25.2.2022] Hozelock. Dostupné z: https://www.hozelock.com/product/25-pot-watering-kit/

[5] *Project Hub*. [online], 2021. [cit. 25.2.2022] MansonHau. Dostupné z: https://create.arduino.cc/projecthub/MansonHau/automatic-watering-system-c40bdf?ref=tag&ref_id=garden&offset=13

[6] THAKARE, Sujit a P.H. BHAGAT. Arduino-Based Smart Irrigation Using Sensors and ESP8266 WiFi Module. In: *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)* [online]. IEEE, 2018, 2018, s. 1-5 [cit. 2021-5-16]. ISBN 978-1-5386-2842-3. Dostupné z: DOI : 10.1109/ICCONS.2018.8663041

[7] Raspberry Pi. IoT community sprinkler system using Raspberry Pi.[online] ©2019[cit 26.2.2022] Alex Bate. Dostupné z: https://www.raspberrypi.com/news/iot-community-sprinkler-system-using-raspberry-pi-the-magpi-issue-83/

[8] IoT based low cost and intelligent module for smart irrigation system. *Computers and Electronics in Agriculture* [online]. ©2019[cit. 26.2. 2022].NAWANDAR, Neha K. a Vishal R. SATPUTE.Dostupné z: DOI : 10.1016/j.compag.2019.05.027

[9] *Lifewire. What Bluetooth is, what it does, and how it works.*[online], 2022. [cit. 5.2.2022] Melanie Uy. Dostupné z: https://www.lifewire.com/what-is-bluetooth-2377412

[10]*IoT Agenda. Zigbee.* 2017, [online]. [cit. 7.2.2022] Linda Rosencrance. Dostupné z: https://internetofthingsagenda.techtarget.com/definition/ZigBee

[11] *The Things Network. What is LoRa and LoRaWAN ?* [online], 2022. [cit. 26.2.2022] LoRa Alliance. Dostupné z: https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/

[12]VALENCIA-GARCÍA, Rafael, Gema ALCARAZ-MÁRMOL, Javier DEL CIOPPO-MORSTADT, Néstor VERA-LUCIO a Martha BUCARAM-LEVERONE, ed. *Technologies and Innovation* [online]. Cham: Springer International Publishing, 2019 [cit. 2022-02-27]. Communications in Computer and Information Science. ISBN 978-3-030-34988-2. Dostupné z: doi : 10.1007/978-3-030-34989-9

[13]KODALI, Ravi Kishore a Borade Samar SARJERAO. A low cost smart irrigation system using MQTT protocol. In: *2017 IEEE Region 10 Symposium (TENSYMP)* [online]. IEEE, 2017, 2017, s. 1-5 [cit. 2022-02-27]. ISBN 978-1-5090-6255-3. Dostupné z: doi:10.1109/TENCONSpring.2017.8070095

[14]International Journal of Advanced Research in Engineering and Technology (IJARET) Volume 10, Issue 2,WEB BASED SMART IRRIGATION SYSTEM USING RASPBERRY PI. ISSN: 0976-6499 2019[online][cit. 26.2. 2022] Dr. Dhiraj Sunehra. Dostupné z: doi: 10.34218/IJARET.10.2.2019.006

[15] Irrigation.Equipment. How Blumat sensors works..In: Sustainable Village LLC [online], 2022. [cit. 2022-02-25] Dostupné z: https://www.irrigation.equipment/wp-content/uploads/2020/05/how-it-works-1024x493.jpg

[16] Hozelock. Irrigation Layout. In: Hozelock.com. [online], 2021. [cit. 2022-02-25] Dostupné z: https://www.hozelock.com/wp-content/uploads/2018/11/2804-25PotWateringKit.png

[17] *Umass Extension Greenhouse Crops and Floriculture Program*.[online], 2022. [cit. 2022-3-03]. University of Massachusetts Amherst. Dostupné z :

https://ag.umass.edu/greenhouse-floriculture/fact-sheets/few-pointers-for-better-irrigation

[18]*Random Nerd Tutorials. What is MQTT and How It Works.* [online], 2021.[cit. 2022-3-04] Rui Santos. Dostupné z: https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/

[19] *NodeMcu Espressif 8266*. [online], 2022. [cit. 2022-03-04]. Dostupné z: https://www.gme.cz/nodemcu-esp8266-lua-wifi-v3-ch340#

[20]*Půdní vlhkoměr analogový s antikorozní sondou V1.2* [online], 2022. [cit. 2022-03-04]. Dostupné z: https://dratek.cz/photos/produkty/d/4/4875.jpg?m=1531824539

[21]*Raspberry Pi 3b+*. [online] , 2022. [cit. 2022-03-04]. Dostupné z: https://dratek.cz/photos/produkty/d/3/3067.jpg?m=1523005326

[22]*Bistabilní ventil ½ "* [online], 2022. [cit. 2022-03-04] . Dostupné z: https://dratek.cz/photos/produkty/d/1/1213.jpg?m=1502871273

[23] *Seedstudio. What is Power over Ethernet (PoE) and How to use PoE HAT with the Raspberry Pi*.[online], 2019 [cit. 2022-03-04]. Dostupné z: https://www.seeedstudio.com/blog/2019/12/11/what-is-poe-and-how-to-use-it-with-the-raspberry-pi/

[24] *JDBC mySQL Connector Library*. [online] 2022. Oracle [cit. 2022-03-14] Dostupné z: https://oracle-base.com/articles/mysql/mysql-connections-in-sql-developer

[25] *Eclipse Paho Java Client* [online], 2022. Eclipse Foundation [cit. 2022-03-14]. Dostupné z: https://www.eclipse.org/paho/index.php?page=clients/java/index.php

[26] *Síťové programování v jazyce Java* [online], 2020. Ing. Jiří Jelínek [cit. 2022-03-17] Dostupné z: https://elearning.jcu.cz/pluginfile.php/274957/mod_resource/content/10/05-Sitove_programovani.pdf

[27] *Basic Microcontroller Use for Measurement and Control* [online], 2021. University College Dublin and Virginia Tech: Nicholas M. Holden [cit. 2023-01-17]. Dostupné z: https://eng.libretexts.org/Bookshelves/Biological_Engineering/Introduction_to_Biosystems_Engineering_(Holden_et_al.)/02%3A_Information_Technology_Sensors_and_Control_Systems/2.01%3A_Basic_Microcontroller_Use_for_Measurement_and_Control

Seznam obrázků

Seznam fragmentů zdrojového kódu

Seznam grafů

Seznam tabulek

Příloha A

Přílohu tvoří soubor *AutomaticIrrigation.zip*. Tento soubor se skládá ze 4 dílčích složek, obsahujících zdrojové kódy. První složkou je *IrrigationClient*, jež obsahuje zdrojové kódy klientské aplikace. Ve druhé složce, pod názvem *IrrigationServer,* se nachází zdrojový kód serverové aplikace. Složka *Raspberry* obsahuje zdrojový kód pro aplikaci řídící jednotky. Poslední složkou je *IrrigationSensor*, obsahující zdrojový kód senzorické jednotky. Zdrojové kódy se nacházejí v patřičných projektech, jejichž součástí je vygenerovaná dokumentace.

Příloha B

Přílohu tvoří soubor *UserGUI.zip*. V tomto souboru lze najít dokument *Průvodce klientskou aplikací.pdf,* jenž obsahuje uživatelský návod pro obsluhu systému.