UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

NÁVRH A IMPLEMENTACE SYSTÉMU PRO ŘÍZENÍ VZDUCHOTECHNIKY

Bc. Jan Dryml

Diplomová práce

2022

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jan Dryml**
Osobní číslo:	**I20202**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Návrh a implementace systému pro řízení vzduchotechniky**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

V teoretické části práce budou popsány současné trendy v oblasti chytrých domů, tzv. smart homes a to jak z pohledu technologií tak aplikací.
Dále budou popsány veškeré SW a HW technologie a komponenty, které budou finálně pro realizace páce vybrány.
V rámci praktické části diplomové práce bude prvně nutné vybrat vhodné hardwarové komponenty systému. Následně bude pozornost věnována vlastnímu návrhu a implementaci komplexního softwarového řešení pro řízení vzduchotechniky. Systém bude rovněž autonomně reagovat na měření sledovaných hodnot v objektu. Kromě vlastního softwaru pro řízení vzduchotechniky bude nutné dále navrhnout aplikaci pro dálkovou správu a nastavení.

Rozsah pracovní zprávy: **50-60**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

NORRIS, Donald. *Home Automation with Raspberry Pi: Projects Using Google Home, Amazon Echo, and Other Intelligent Personal Assistants*. Shelter, Island, NY: McGraw-Hill Education Ltd , 2019 ISBN 978-1260440355
Vaadin Team. *Book of Vaadin: Vaadin 14* Publisher: Independently published, 2019. 571 pages. ISBN 978-1692121440.

Vedoucí diplomové práce: **Ing. Jan Fikejz, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **8. listopadu 2021**
Termín odevzdání diplomové práce: **20. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2021

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č.111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 11. 8. 2022

Bc. Jan Dryml

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu práce Ing. Janu Fikejzovi, Ph.D. za vstřícný přístup, jeho čas a rady, které mi věnoval při zpracovávání diplomové práce.

ANOTACE

Diplomová práce se zabývá návrhem a implementací komplexního systému pro autonomní řízení klimatického komfortu budov. Systém je navržen tak, aby poskytl jednotné rozhraní pro monitorování a regulaci spravovaného prostředí. Sestává ze tří částí: (i) nízko-úrovňových C++ implementací pro správu senzorů propojených bezdrátovou sítí, (ii) centrálního řídicího Java modulu běžícího na Raspberry Pi, (iii) v cloudu dostupné webové aplikace vytvořené pomocí frameworku Vaadin. K uchovávání dat je využita databáze MySQL, která je zároveň využívána jako prostředek ke sdílení dat mezi řídící jednotkou a webovou aplikací. V rámci práce je popsána problematika chytrých budov a také hardwarové a softwarové technologie použité v implementaci.

KLÍČOVÁ SLOVA

Chytré budovy, měření a regulace, komplexní systém, senzory, Arduino, Raspberry Pi, Java, C++, Vaadin, Spring, MySQL

TITLE

DESIGN AND IMPLEMENTATION OF A SYSTEM FOR AIR CONDITIONING CONTROL

ANNOTATION

The thesis deals with the design and implementation of a complex system for autonomous climate comfort control of buildings. The system is designed to provide a unified interface for monitoring and controlling the managed environment. It consists of three parts: a low-level C++ implementation for sensor management connected by a wireless network, a central control Java module running on a Raspberry Pi, and a cloud-based web application developed using the Vaadin framework. A MySQL database is used to store the data, which is also used to share data between the control unit and the web application. The thesis describes the topic of smart buildings and the hardware and software technologies used in the implementation.

KEYWORDS

Smart buildings, measurement and control, complex system, sensors, Arduino, Raspberry Pi, Java, C++, Vaadin, Spring, MySQL

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZKRATEK

ADC	Analog-to-Digital Converter
AOP	Aspektově Orientované Programování
API	Application Programming Interface
ASK	Amplitude Shift Keying
AWS	Amazon Web Services
AWT	Abstract Window Toolkit
CD	Continuous Deployment
CI	Continuous Integration
CRC	Cyclic Redundancy Check
DB	Database
DI	Dependency Injection
DevOps	Development and IT Operations
EEPROM	Electrically Erasable Programmable Read-Only Memory
EJB	Enterprise Java Beans
GND	Ground
GPIO	General Purpose Input/Output
GPL	General-Public License
HDMI	High-Definition Multimedia Interface
HTML	Hyper Text Markup Language
HVAC	Heating, Ventilation and Air Conditioning
HW	Hardware
I/O	Input/Output
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IOC	Inversion of Control
ISM	Institute for Supply Management
ISO	International Organization for Standardization
JAVA EE	Java Enterprise
JDBC	Java Database Connectivity
JPA	Java Persistence API
JSON	JavaScript Object Notation

JTA	Java Transaction API
JVM	Java Virtual Machine
MVC	Model-View-Controller
NDIR	Nondispersive Infrared
ODBC	Open Database Connectivity
OP	Operační
ORM	Object relational mapping
OS	Operational system
PLC	Programmable Logic Controller
PLL	Phase Lock Loop
ppm	particles per million
PWM	Pulse-Width Modulation
RAM	Random Access Memory
RDBMS	Relational Database Management System
RF	Radio frequency
RIA	Rich Internet Application
SAW	Surface Acoustic Wave
SLOC	Source Lines of Code
SPA	Single Page Application
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SRAM	Static Random Access Memory
SW	Software
TCP	Transmission Control Protocol
USB	Universal Serial Bus
VF	Vysokofrekvenční
WORA	Write Once, Run Everywhere
WiFi	Wireless Fidelity
XML	Extensible Markup Language
YAML	YAML Ain't Markup Langua

1 ÚVOD

V dnešní době se lze stále častěji setkávat s technologiemi, v jejichž názvu se objevuje slovo „chytrý“, ať už jde o různá zařízení, budovy, domácnosti a mnohé další. I přesto, že se tyto technologie využívají v různých segmentech, často je spojuje schopnost sbírat a vyhodnocovat informace získané z okolního prostředí a na jejich základě dělat různá inteligentní rozhodnutí. Pojem inteligentní může být v určitých systémech zavádějící, jelikož se často jedná pouze o rozhodnutí vycházející z uživatelských konfigurací. Stále se však jedná o autonomní chování, do kterého není nutné zasahovat. Existují ovšem i systémy, které ke svému řízení využívají pokročilých expertních systémů nebo umělé inteligence.

Tato diplomová práce se primárně zaměřuje na problematiku chytrých budov, přesněji pak na návrh a implementaci autonomního systému pro řízení topení, chlazení a větrání se zpětnou rekuperací tepla. Hlavním cílem tohoto systému bude poskytnout jednotné uživatelské rozhraní pro monitorování a konfiguraci autonomního řízení všech spravovaných zařízení. Ke konfiguraci řízení bude využíváno několika typů plánů, ve kterých budou stanoveny reakce na různé události a stavy systému. Tyto plány bude uživatel moci dynamicky přidávat, či modifikovat.

Navrhovaný systém bude velmi komplexní, protože bude pokrývat různé aspekty implementace chytré budovy. Ty se od sebe dost často liší využitými technologiemi. Z toho důvodu bude systém rozdělen do tří modulů. První z nich bude zajišťovat nízko-úrovňovou implementaci sběru dat ze senzorů a zároveň poskytovat infrastrukturu pro bezdrátovou komunikaci využívanou k jejich propojení s dalšími částmi systému. Druhá část bude spravovat senzorové moduly, zpracovávat získaná data a na základě uživatelských konfigurací řídit spravovaná zařízení. Tento modul by se dal považovat za hlavní řídící část a v podstatě mozek celého systému. Poslední modul bude webová responzivní aplikace, kterou budou uživatelé moci využít k monitoringu a správě celého systému.

V teoretické části prvně proběhne seznámení s konceptem chytrých budov, s principem, jak fungují a s jejich vlastnostmi. Dále bude čtenář seznámen s nejdůležitějšími hardwarovými a softwarovými technologiemi, které budou využity v rámci návrhu a implementace samotné práce.

V praktické části práce se čtenář podrobněji seznámí s návrhem, klíčovými funkcionalitami, způsobem používání a nasazením celého systému. Tato část práce bude doplněna screenshoty

a ukázkami, jak nástroj správně nakonfigurovat a využívat. Cílem praktické části je seznámit čtenáře s celým systémem, jeho implementací a ukázat mu, jak s ním samostatně pracovat.

2 INTELIGENTNÍ BUDOVY

Za inteligentní budovu lze považovat takovou stavbu, která využívá moderní technologie k efektivnímu a úspornému využívání zdrojů a zároveň poskytuje bezpečné a pohodlné prostředí pro uživatele. Takto by mohla znít jedna z mnoha možných definic tohoto termínu. Samotný pojem „inteligentní budovy“, v originále „intelligent buildings“, byl poprvé použit v roce 1981 americkou společností United Technology Building Systems. I přesto, že tento termín existuje už několik dekád a v průběhu let proběhlo mnoho pokusů o zavedení jeho všeobecně uznávané definice, doposud se tomu tak nestalo. Existuje tedy mnoho definic s různou úrovní detailů a různou mírou důrazu na odlišné aspekty inteligence budov. Důvodem jsou právě zmíněné detaily a důraz na různé aspekty v jednotlivých implementacích. Jednoduše řečeno, jde o zastřešující pojem, který může každý chápat subjektivně podle toho, co od něj očekává a jak ho realizuje. Budova se dá do jisté míry již považovat za inteligentní, pokud je například řízena termostatem, nebo poskytuje rezervační systém místností. Obecně řečeno jí lze považovat za inteligentní, pokud v ní figuruje nějaký prvek, kterým může být její prostředí nebo funkčnost řízena a optimalizována. Spektrum realizací je opravdu široké, od výše zmíněného jednoduchého řešení, přes plně monitorované a autonomně řízené domácnosti, až po koncept chytrých měst, využívajících umělou inteligenci pro efektivní řízení své infrastruktury a spotřeby energie.

Tato práce si neklade za cíl přesnou definici zmíněného pojmu nalézt. Definice uvedená na začátku kapitoly bude v kontextu práce plně dostačující, jelikož v obecné rovině dobře vystihuje, jak inteligentní budovy chápat a k čemu slouží. V následujících podkapitolách bude popsáno, jakým způsobem obecně inteligentní budovy fungují, příklady jejich využití a co od nich očekávat do budoucna.

Zdroje: [1]

2.1 Princip

Jak bylo řečeno v předchozích odstavcích, problematika inteligentních budov je značně rozsáhlá a sdružuje spoustu různých konceptů. Nicméně obecně by se dalo říci, že se jedná o systém, který na základě informací z řízeného prostředí s daným prostředím interaguje a ovlivňuje ho. Nejlépe tento princip vystihuje zkratka z oboru stavebnictví „MaR“, tedy měření a regulace. Po přeformulování by definice zněla takto: „systém regulující své prostředí na základě hodnot z něj naměřených.“ Pro úplné porozumění bude ještě tento princip

demonstrován na příkladu z minulé části, tj. budovy vybavené termostatem. Taková budova monitoruje vnitřní teplotu pomocí teplotních senzorů. Pokud je například zjištěna teplota vyšší, než je pro obyvatele únosná, (tedy na termostatu nastavená jako hraniční) spustí cirkulace vzduchu a chlazení. V definici principu chytrých budov se dá sledovat jistá paralela k definici principu automatizace. Je to z toho důvodu, že koncept chytrých budov spadá do odvětví automatizace. Samotný princip je v celku jednoduchý, složitější je pak navrhnout a implementovat systémy, které mají za úkol regulovat komplexní prostředí vyžadující exaktní řízení. Z tohoto důvodu vznikly dokonce studijní obory věnující se této problematice.

Zdroje: [1], [5].

2.2 Charakteristiky a využití

Chytré budovy jsou schopny výrazně snížit jak náklady spojené s provozem, tak i s údržbou. Zavedením systémů HVAC (systém topení, provětrávání a klimatizace), inteligentního osvětlení, senzorů monitorujících přítomnost osob a dalších IoT řešení mohou tyto typy budov ušetřit výrazné procento financí vynaložených na provozní náklady. Například tím, že díky informacím získaných ze senzorů monitorujících přítomnost osob se při zjištění absence osob v místnosti vypne osvětlení, topení atp. Tím se dá dosáhnout nejen snížení výdajů za spotřebu energií, ale má to i pozitivní dopad na životní prostředí.

Analyzováním dat ze zařízení v reálném čase je možné snížit objem prací na údržbě a tím snížit náklady, které by na ni bylo potřeba vynaložit. Lze toho dosáhnout za pomoci strojového učení, jehož algoritmy jsou schopny detekovat události, respektive stavy zařízení (například změna napětí v zařízení, odchylka od běžného chování atp.), které mohou postupem času vést k poškození zařízení, nebo v horším případě i k poškození okolního prostředí. Díky včasné detekci lze tomuto předcházet. Systém dokáže preventivně zařízení vypnout nebo varovat odpovědné osoby dříve, než se náprava stane mnohem nákladnější.

Implementace chytrých budov se využívá i v podnikové sféře, například v kancelářích, kde dokáže automatizovaně zajistit optimální podmínky k práci, například udržováním ideální úrovně osvětlení a klimatických podmínek. Díky tomu se zaměstnanci cítí příjemněji a zvedá se jejich produktivita. Další možnou optimalizací pracovního prostředí, je využití aplikací, které využívají rozšířené reality k navigování uvnitř větších budov a systémů k rezervaci místností. Tyto systémy jsou užitečné hlavně pro velké firmy se spoustou zaměstnanců, kde pomáhají zvýšit produktivitu práce a zjednodušují pracovní postupy.

Ekonomických výhod, spojených s využitím chytrých budov, existuje celá řada, nicméně se dají využít i jinými způsoby. Existují systémy, které jsou využívány pro monitorování domácností starších osob a osob se zdravotními problémy. Takové systémy jsou vybaveny senzory, které dokáží rozpoznat, zda monitorovaná osoba netrpí nějakými akutními problémy a není ohrožena na životě. Pokud systém zjistí, že je osoba v nesnázích, zalarmuje pečovatele, nebo záchrannou službu, která se může do objektu jednoduše dostat díky systému řízení přístupů.

Kromě spousty benefitů, které chytré domy poskytují, existuje i pár nevýhod a rizik. V případě, že je systém veřejně dostupný z internetu, například z důvodu monitorování, nebo dokonce i řízení systému na dálku, vystavuje se vlastník riziku, že bude jeho systém napaden. Útočník může takto napadený systém zneužít různými způsoby, od takzvaného „zotročení“ chytrých prvků, které jsou zneužity například k těžbě kryptoměn nebo zapojeny do botnetů, přes zablokování systému, kdy požaduje výkupné za odblokování až po umožnění nepovoleného přístupu a vykradení budovy. V rámci boje proti tomuto problému zavádí mnoho zemí a organizací nařízení nebo požadavky na systémy, které mají společnosti donutit, aby své nabízené systémy lépe zabezpečili.

Další nevýhodou jsou vysoké náklady za prvotní instalaci prvku chytrých budov, které se mohou vyšplhat na mnoho stovek tisíc až jednotek milionů korun. Hlavně pro menší investory, nebo společnosti to může být příliš velká investice. Cena se však odvíjí od velikosti budovy, počtu instalovaných prvků a komplexnosti systému. Pokud je však tato překážka překonána, díky výhodám popsaným výše je návratnost této investice téměř jistá.

Poslední z nevýhod, kterou sebou realizace chytrých budov nese je, že pokud dojde k výpadku řídicího systému, mohou nastat nepříjemnosti v podobě zablokování celého systému a spravovaná zařízení nebudou pracovat tak, jak je požadováno. Tomuto se ovšem dá předcházet definováním řízení tohoto rizika, konfigurováním výchozího chování pro jednotlivé prvky, možnosti je spravovat manuálně nebo mít pro takový případ specializovaný personál.

Zdroje: [1], [5].

3 POUŽITÉ TECHNOLOGIE

V rámci implementace diplomové práce bylo využito vícero hardwarových částí, softwarových technologii a frameworků. V této kapitole budou představeny nejdůležitější z nich a bude uvedeno jejich začlenění v samotné práci.

3.1 Hardware

V praktické části práce, která zajišťuje řízení chytré domácnosti, je použito několik hardwarových částí. Kromě senzorů, obstarávajících samotný sběr dat, jde o zařízení, která tato data shromažďují, agregují a řídí reakce na naměřené hodnoty a další konfigurace. Tyto zařízení budou představena v následujících podkapitolách. Budou zde popsána i zařízení, která jsou v implementaci systémem řízena.

3.1.1 AM2120

AM2120 je kompozitní čidlo schopné měřit teplotu a vlhkost. Čidlo obsahuje kapacitní snímač vlhkosti a integrovaný teplotní senzor připojený k zabudovanému mikroprocesoru. Technologie snímání teploty a vlhkosti, kterou čidlo využívá, zajišťuje vysokou spolehlivost a dlouhodobou přesnost měření. Výrobek má v poměru k ceně rychlou odezvu, dobrou kvalitu a schopnost odolávat rušení.

AM2120 se do systému zapojuje v režimu využívajícím jednosběrnicovou komunikaci, integrace se tak stává rychlou a jednoduchou. Senzor má velmi malé rozměry, nízkou spotřebu energie, vzdálenost přenosu signálu až 20 metrů, je tedy dobrou volbou pro různé druhy realizací, včetně náročnějších. Využívá přímého výstupu teplotně kompenzovaných digitálních informací, jako je vlhkost a teplota. Uživatelé tak nemusí provádět sekundární výpočty na digitálním výstupu a nepotřebují teplotní kompenzaci, aby získali přesná data o teplotě a vlhkosti.

Teplotu lze měřit v rozsahu -40 °C až +80 °C, kdy odchylka od skutečné teploty je ±0,5 °C. Vlhkost je možné měřit v rozsahu 0-99,9 % relativní vlhkosti, kdy je odchylka ±3 %. Pro napájení modulu je možné použít napětí v rozsahu 3,3-5,5 V.

Jak z předchozího textu vyplývá, senzor se v implementaci systému využívá jako teploměr a vlhkoměr. Více informací o senzoru je dostupné v této dokumentaci [6].

Zdroje: [6].

3.1.2 MH-Z19B

MH-Z19B je čidlo sloužící k přesnému měření obsahu oxidu uhličitého v prostředí. Disponuje také méně přesným senzorem pro měření teploty. V poměru k relativně nízké ceně se čidlo oxidu uhličitého vyznačuje vysokou citlivostí a stabilitou, nízkou spotřebou energie, dobrou teplotní kompenzací a lineárním výstupem.

K přesnému měření obsahu oxidu uhličitého se využívá princip nedisperzního infračerveného záření (NDIR). NDIR senzor funguje tak, že infračervená lampa vysílá světelné vlny skrz trubici naplněnou vzorkem vzduchu. Tento vzduch se pohybuje směrem k optickému filtru před detektorem infračerveného světla. Detektor infračerveného světla následně měří množství infračerveného světla, které projde optickým filtrem.

Hodnotu oxidu uhličitého lze měřit v rozsahu 0 ppm až 2000 ppm, kdy odchylka od skutečné hodnoty je ± 50 ppm. Zařízení je již předkalibrováno z výroby, nicméně se jej doporučuje každý půl rok překalibrovat. Lze zapnout autokalibraci, která si každých 24 hodin nastavuje svou spodní hranici podle nejnižší naměřené hodnoty. Tento režim se ale doporučuje pouze pro venkovní užití. Celé zařízení je uzavřeno v pozlacené komoře, která je vodotěsná a odolná proti korozi. Teplota prostředí, ve které senzor měří, by se neměla dostat mimo rozsah 0 °C až 50 °C a relativní vlhkost by neměla být vyšší než 95 %. Pro napájení modulu je možné použít napětí v rozsahu 3,6-5,5 V.

Jak z předchozího textu vyplývá, senzor se v implementaci systému využívá pro měření oxidu uhličitého. Více informací o senzoru je dostupné v této dokumentaci [7].

Zdroje: [7].

3.1.3 NRF24L01+

NRF24L01+ je jednočipový rádiový transceiver (pracuje na protokolu SPI), který slouží k odesílání a přijímání dat na rádiové frekvenci 2,4-2,5 GHz v pásmu ISM. V rámci jedné frekvence dokáže komunikovat až na 125 kanálech. V rámci jednoho kanálu dokáže komunikovat se šesti dalšími moduly. Díky těmto vlastnostem jej můžeme použít v mesh sítích a dalších síťových aplikacích.

Skládá se z plně integrovaného frekvenčního syntetizéru, výkonového zesilovače, modulátoru krystalového oscilátoru, demodulátoru a regulátoru nárazového režimu (ten řeší krátkodobý vysoký provoz v rámci sítě). Disponuje režimem nízké spotřeby, kdy v režimu vysílače spotřebovává pouze 11,3 mA a v režimu přijímače 13,5 mA. Tento modul je určen

pro přenos dat na velké vzdálenosti a rychlý přenos. Rychlost bezdrátového přenosu dat se pohybuje kolem 2 Mb/s. Jeho vysoká rychlost přenosu dat v kombinaci s úsporným režimem jej činí velmi výhodným pro aplikace s požadavkem na nízkou spotřebu energie. Má také kompaktní rozměry a lze jej snadno použít ve stísněných prostorech. Je navržen pro provoz při napětí 3,3 V.

Zařízení se v implementaci používá pro komunikaci mezi Arduiny, které obsluhují senzory a aplikací pro řízení chytré domácností, která se nachází na RaspberryPi. Více informací o transcieveru je dostupné v této dokumentaci [8].

Zdroje: [8].

3.1.4 RF 433Mhz

RF 433 MHz vysílací a přijímací modul je dvojice malých radiofrekvenčních elektronických součástek, které slouží k vysílání a příjmu rádiových signálů mezi dvěma libovolnými zařízeními.

Menší z modulů je vysílač. Jde o velice jednoduché zařízení. Srdcem modulu je SAW rezonátor, který je naladěn pro provoz na frekvenci 433 MHz. Na modulu se dále nachází spínací tranzistor a několik pasivních součástek. Když je na vstup „data“ přivedena logická hodnota „high“, oscilátor produkuje konstantní výstupní RF nosnou vlnu na frekvenci 433 MHz. Když je na vstup „data“ přivedena logická hodnota „low“, oscilátor se zastaví. Tato technika je známá pod zkratkou ASK nebo také jinak klíčování amplitudovým posunem.

Druhý z modulů je přijímač. Přestože vypadá složitěji, je podobně jednoduchý jako modul vysílače. Skládá se z VF laděného obvodu a dvojice OP zesilovačů, které zesilují přijímanou nosnou vlnu z vysílače. Zesílený signál je dále přiváděn do PLL, která umožňuje dekodéru „uzamknout“ proud digitálních bitů, což poskytuje lepší dekódovaný výstup a odolnost proti šumu.

Tyto moduly byly testovány v prvotní fázi implementace systému, nicméně se neosvědčily, primárně kvůli nízké přenosové vzdálenosti, složité implementaci komunikace mezi více zařízeními a s tím spojenou potřebou většího počtu těchto modulů. Více informací o modulech je dostupné v této dokumentaci [9].

Zdroje: [9].

3.1.5 Arduino Nano

Arduino Nano je malá vývojová deska, vhodná pro použití na nepájivém poli, založená na 8bitovém mikrokontroléru AVR. K dispozici jsou dvě verze této desky, jedna je založena na ATmega328p a druhá na Atmega168. Její systémový kód je typu open-source, do její implementace může tedy přispívat kdokoliv z vývojářské komunity. Arduino Nano dokáže vykonávat podobné funkce jako jiné desky dostupné na trhu, má však menší rozměry a hodí se pro projekty vyžadující méně paměti a méně I/O pinů.

Zařízení disponuje krystalovým oscilátorem, pracujícím na frekvenci 16 MHz. Paměť flash, sloužící k ukládání programů, má v zařízení Atmega168 velikost 16 kB a v zařízení Atmega328 velikost 32 kB, z toho 2 kB jsou využity zavaděčem. Pro Atmega168 a Atmega328 má paměť EEPROM velikost 512 kB a 1 kB a paměť SRAM velikost 1 kB a 2 kB.

Deska podporuje rozhraní USB, ale na rozdíl od většiny desek Arduino, používajících standardní port USB, používá mini USB port. Součástí této jednotky není konektor pro stejnosměrné napájení, desku lze napájet pomocí USB portu nebo připojením napájení na odpovídající piny.

V implementaci je Arduino využito k přímé komunikaci se senzory a sběru naměřených dat, dále k řízení bezdrátové komunikace a ke komunikaci s Data collectorem přes USB port. Více informací o zařízení je dostupné v této dokumentaci [11].

Zdroje: [10], [11].

3.1.6 Raspberry Pi

Raspberry Pi je název jednodeskových počítačů, vytvářených britskou organizací Raspberry Pi Foundation. Jejich cílem je vzdělávat lidi v oblasti výpočetní techniky a zjednodušovat jim přístup k počítačovému vzdělávání. Z tohoto důvodu nabízejí různé druhy velmi levných jednodeskových počítačů.

První z řady počítačů Raspberry Pi byl uveden na trh v roce 2012 a od té doby bylo vydáno několik jeho iterací a variant. První model Raspberry Pi měl jedno-jádrový 700 MHz procesor a 256 MB RAM. Počítače Raspberry Pi byly vždy cenově dostupné, jejich cena se pohybuje kolem 1000 Kč, nejlevnější je Pi Zero, který stojí kolem 125 Kč. Standardně je na něm využíván open-source operační systém Raspberry Pi OS (nazývaný také Raspbian), ale lze na něm rozeběhnout téměř jakoukoliv Linuxovou distribuci. Dokáže také běžet pod Windowsovými

distribucemi. Poskytuje řadu pinů GPIO (vstup/výstup pro všeobecné účely), které umožňují ovládat nepřeberné množství elektronických komponentů.

V implementaci bylo použito Raspberry Pi 3 Model B. Jde o třetí generaci těchto jednodeskových počítačů. Je založen na čipu BCM2837, který obsahuje čtyřjádrový 64bitový procesor ARMv8 s frekvencí 1,2 GHz a výkonný grafický procesor VideoCore IV. Tento model je prvním počítačem Raspberry Pi, který je vybaven vestavěným modulem pro bezdrátové WiFi připojení a rozhraním Bluetooth. Má stejný tvar a umístění konektorů jako starší Raspberry Pi 2 Model B a Raspberry Pi Model B+. Zároveň rozložení GPIO zůstalo beze změn, je tudíž kompatibilní s řešeními, která využívala starší modely. Disponuje také RAM o velikosti 1 GB, 4 USB porty, HDMI portem a portem pro Ethernet.

V rámci implementace je zařízení použito jako hlavní jednotka, která slouží ke shromažďování dat z pozorovaného prostředí a jeho následnému řízení. Zařízení lze považovat za ústřední část implementovaného systému. Více informací o zařízení je dostupné v této dokumentaci [13].

Zdroje: [12], [13].

3.1.7 Modbus

Modbus je datový komunikační protokol založený na architektuře master-slave, který byl vytvořen v roce 1979 pro potřeby komunikace mezi PLC zařízeními. Protokol Modbus je v průmyslovém prostředí oblíbený, protože je otevřeně publikován, lze tedy nahlédnout do zdrojového kódu, a navíc je bezplatný. Byl vyvinut pro průmyslové aplikace, v porovnání s jinými standardy se poměrně snadno zavádí, udržuje a klade jen malá omezení na formát přenášených dat. Díky těmto vlastnostem se Modbus stal de facto standardním komunikačním protokolem a dnes je běžně dostupným prostředkem pro připojení průmyslových elektronických zařízení.

K přenosu dat využívá rozhraní RS-485, RS-422, RS-232 i TCP/IP sítě Ethernet (protokol Modbus TCP). Modbus podporuje komunikaci z a do více zařízení připojených k témuž kabelu nebo síti Ethernet. Příkladem může být zařízení, které měří teplotu, a další zařízení na měření vlhkosti připojené ke stejnému kabelu, přičemž obě sdělují naměřené hodnoty stejnému počítači, a to prostřednictvím sběrnice Modbus.

Zpráva, posílaná v protokolu Modbus, se skládá z adresy zařízení SlaveId, identifikátoru funkce, samotných dat a kontrolního součtu CRC. Pokud odstraníme adresu zařízení SlaveId a kontrolní součet CRC, získáme datovou část protokolu. Ta se skládá z identifikátoru

funkce, díky ní poznáme, jaká funkce bude vykonána a samotných dat, která jsou interpretována dle zmíněného identifikátoru funkce.

Vývoj a aktualizaci protokolů Modbus řídí organizace Modbus Organization od dubna 2004, kdy na ni výrobce Schneider Electric převedl práva. Modbus Organization je sdružení uživatelů a dodavatelů zařízení kompatibilních s protokolem Modbus, které se zasazuje o další používání této technologie.

Implementace tohoto protokolu je využívána v Data collectoru, kde slouží ke komunikaci s externím modulem, který slouží k řízení externích zařízení, jako je například klimatizace a rekuperace. Více informací o protokolu je dostupné v této dokumentaci [14].

Zdroje: [14], [15].

3.2 Software

3.2.1 C++

C++ je univerzální programovací jazyk, který vytvořil dánský informatik Bjarne Stroustrup jako rozšíření programovacího jazyka C neboli "C s třídami". Jazyk se postupem času výrazně rozšířil a moderní C++ má nyní kromě prostředků pro nízko-úrovňovou manipulaci s pamětí, také objektově orientované, generické a funkcionální funkce. Téměř vždy je implementován jako kompilovaný jazyk, jehož kompilátory poskytuje mnoho výrobců a díky tomu ho lze kompilovat na mnoho různých typů platforem.

Jazyk C++ byl navržen s orientací na programování systémových aplikací, vestavěného softwaru s omezenými zdroji a rozsáhlých systémů. Jeho hlavními přednostmi jsou výkonnost, efektivita a flexibilita použití. Jazyk C++ se ukázal jako užitečný i v mnoha dalších případech, přičemž je hlavně využívaný v oblastech, jako jsou softwarová infrastruktura a aplikace s omezenými zdroji, včetně desktopových aplikací, videoher, serverů (např. webové vyhledávače nebo databáze) a aplikace kritické na výkon.

Jazyk C++ je v práci využit při implementaci programů běžících na Arduinech pro řízení senzorů a bezdrátové komunikace. Více informací o jazyku je dostupné v této dokumentaci [4].

Zdroje: [4].

3.2.2 Java

Java je vysoko-úrovňový objektově orientovaný programovací jazyk založený na třídách, který je navržen tak, aby měl co nejméně implementačních závislostí. Jedná se o univerzální programovací jazyk, který má programátorům umožnit „napsat“ aplikaci jednou a spustit kdekoli (princip WORA). To znamená, že zkompilovaný kód v jazyce Java může běžet na všech platformách, které podporují jazyk Java, bez nutnosti překompilování. Aplikace v jazyce Java jsou obvykle kompilovány do bajtového kódu, který může běžet na jakémkoli virtuálním stroji Java (JVM) bez ohledu na architekturu cílového systému. Syntaxe jazyka Java je podobná jazykům C a C++, ale má méně nízko-úrovňových prostředků. Běhové prostředí jazyka Java poskytuje dynamické možnosti (například reflexi a modifikaci kódu za běhu), které v tradičních kompilovaných jazycích obvykle nejsou k dispozici.

V práci je jazyk využit jako primární programovací jazyk v modulech Data collector a Data manager. Více informací o jazyku je dostupné v této dokumentaci [3].

Zdroje: [3].

3.2.3 Lombok

Java je jedním z nejvíce používaných programovacích jazyků, má však několik nevýhod. Jednou z velmi neoblíbených nevýhod je jeho „upovídanost“. Například musíme psát stále se opakující kód, jako jsou gettery, settery, equals(), toString() atd. Kotlin a Scala, které jsou také založeny na JVM, to nepotřebují, což může být jedním z aspektů jejich vyšší popularity ve vývojářské komunitě. Zde přichází Lombok, který se tuto nevýhodu Javy snaží řešit.

Projekt Lombok je Java knihovna, která slouží k minimalizaci/odstranění opakujícího se kódu a ušetření času vývojářů při vývoji. Dosahuje toho použitím specializovaných anotací. Lze namítnout, že v dnešní době téměř všichni používají IDE, která poskytují možnost generování těchto šablonových kódů, a tím pádem Lombok moc užitečný není. Kdykoli je však použito IDE pro generování těchto šablonových kódů, uživatel si sice ušetří čas strávený psaním, ale generovaný kód je stále přítomný ve zdrojovém kódu, Zvyšuje tak počet SLOC (řádky kódu) a snižuje udržovatelnost a čitelnost. Lombok je spouštěn až ve fázi kompilace a podle anotací generuje kód až v této době.

Projekt Lombok je využíván ve všech částech implementace, kde figuruje i programovací jazyk Java. Více informací o knihovně je dostupné v této dokumentaci [16][16].

Zdroje: [15], [16].

3.2.4 Vaadin

Vaadin je open source framework používaný pro vývoj webových aplikací. Při vývoji se postupuje podobným přístupem, jakým se postupuje při vývoji desktopových aplikací pomocí běžných nástrojů Java, jako je například AWT, Swing, nebo JavaFx. K vývoji nejsou potřeba žádné webové front-end technologie jako JavaScript nebo HTML. Jediný programovací jazyk, který je potřeba, je Java. Samotnou front-end část lze zapisovat přímo v jazyce Java, nebo navrhovat způsobem drag and drop ve specializovaném designerském programu. Podobně jako je to například v případě JavaFx a designeru Gluon. Framework umožňuje vytvářet bohaté webové aplikace (RIA) ve snaze dosáhnout podobné responzivity (snaha o minimalizaci reloadů stránek), jakou můžeme získat u desktopových aplikací. Umožňuje dokonce vytvářet SPA aplikace, u kterých není reload stránek potřeba vůbec.

Jak již bylo zmíněno, výhodami frameworku je, že není potřeba znalosti dalších webových front-end technologií. Framework obsahuje množství komponentů, modulů a řešení pro problémy, které se často v oblasti webových aplikací vyskytují. Framework je také relativně jednoduchý na naučení a díky množství pluginů ho lze lehce zaintegrovat do již existujících řešení.

Mezi nevýhody patří velikost generovaných souborů potřebných pro front-end část. Ty jsou úzce propojeny s back-endem, takže výsledná aplikace hodně místa v operační paměti. Při komunikaci mezi front-end a back-end částí využívá framework tzv. nízko-úrovňových událostí. V případě běžného přístupu (využívání tzv. sémantických událostí), např. při žádosti o odstranění uživatele, je zaslán z front-endu požadavek na server „odstraň tohoto uživatele“. Avšak při používaní dříve zmíněných nízko-úrovňových událostí je z front-endu zaslán požadavek typu „bylo kliknuto na tlačítko s id 42“. V takové zprávě není žádná aplikační logika. Pouze server ví, co kliknutí na tlačítko sémanticky znamená. Celá aplikační logika se tak nachází výhradně na straně serveru. To vede k vysoké provázanosti front-end části s back-end částí a převádění většiny operativy na server. Tím pádem je tento přístup odsouzen k mnohem horší škálovatelnosti, než běžně využívané technologie (např. Spring v kombinaci s Reactem, Angularem).

Vaadin je využit pro vytvoření webové aplikace Data manageru. S výhodou zde bylo využito designeru pro tvorbu grafického rozhraní. Více informací o frameworku je dostupné v této dokumentaci [2].

Zdroje: [2], [18].

3.2.5 Spring

Spring Framework je open-source framework pro vytváření enterprise aplikací v jazyce Java. Spring si klade za cíl zjednodušit složitý a těžkopádný proces vývoje enterprise aplikací v jazyce Java tím, že nabízí framework, který zahrnuje spoustu užitečných nástrojů, jako je např. aspektově orientované programování (AOP), dependency injection (DI) a inversion of control (IOC) atd. Na obecné úrovni můžeme framework Spring považovat za soubor dílčích frameworků, jako jsou Spring Web Flow, Spring MVC, Spring ORM a mnoha dalších (aktuálně celkem 21 dalších frameworků). I se všemi těmito technologiemi je Spring relativně paměťově nenáročný framework, který lze použít k vytváření škálovatelných, bezpečných a robustních enterprise aplikací. Kromě jazyka Java podporuje Spring také jazyky Kotlin a Groovy.

Kontejner IoC je jednou ze základních a velmi důležitých částí Springu, která poskytuje zjednodušený způsob konfigurace a správy objektů. Tento kontejner je zodpovědný za správu životního cyklu definovaných objektů, což výrazně zvyšuje konfigurovatelnost aplikace. IoC používá vzory dependency injection a dependency lookup k poskytování reference na objekt během běhu.

Cílem AOP je poskytnout větší modularitu. Snaží se sdružit části programu se společnou logikou, rozprostírající se napříč aplikací, do jednoho místa. Jde například o logování, ukládání do mezipaměti, správu transakcí, validaci dat, atp. V AOP je hlavní jednotkou modularity aspekt (průřezové problémy). Kontejner IoC a AOP jsou na sobě nezávislé, což vývojářům nabízí svobodu při výběru preferované metody programování. Nicméně aspektově orientované programování v kombinaci s kontejnerem IoC poskytuje velice robustní řešení.

Problémy s komunikací s databází jsou jedním z častých problémů, se kterými se vývojáři při vývoji aplikací potýkají. Spring zjednodušuje proces komunikace s databází tím, že poskytuje přímou podporu populárních frameworků pro přístup k datům v Javě, jako jsou JDBC, Hibernate, JPA, atd. Navíc nabízí funkce jako je správa zdrojů, zpracování výjimek a sjednocení způsobu práce se všemi podporovanými knihovnami pro přístup k datům, což dále zjednodušuje proces vývoje.

Spring také nabízí abstraktní mechanismus, který uživatelům umožňuje pracovat s lokálními, globálními a vnořenými transakcemi, ukládat záchytné body a zjednodušit správu transakcí v celé aplikaci. To umožňuje vývojářům vytvářet funkčně bohaté transakční systémy, které se rozprostírají napříč aplikacemi, aniž by byly závislé na EJB nebo JTA.

Spring MVC umožňuje vývojářům vytvářet aplikace pomocí oblíbeného architektonického vzoru MVC. Jedná se o přístup založený na zpracovávání požadavků, který vývojářům

umožňuje snadno vytvářet vlastní implementace MVC, které přesně vyhovují jejich potřebám. Základní komponentou Spring MVC je třída DispatcherServlet, která zpracovává požadavky uživatelů a následně je předává odpovídajícímu kontroléru. Kontrolér tak může zpracovat požadavek, vytvořit model a poté poskytnout informace koncovému uživateli prostřednictvím určeného zobrazení.

Testování je základní součástí každého vývoje. Spring zjednodušuje i testování v rámci frameworku pomocí funkcionalit, jako je například mockování objektů, testování pouze určitých kontextů atp.

Spring framework a převážně níže popsaný Spring boot je využit jak v Data collectoru, tak Data manageru. Více informací o frameworku je dostupné v této dokumentaci [20].

Zdroje: [19], [20] .

3.2.5.1 Spring boot

Spring Boot je nástroj, který urychluje a usnadňuje vývoj webových aplikací a mikroslužeb pomocí frameworku Spring, díky třem základním funkcionalitám, a to automatickým konfiguracím, zaujatému přístupu k výchozímu konkretizování konfigurací a schopnosti vytvářet standalone aplikace.

Automatická konfigurace zajišťuje inicializaci aplikace přednastavenými závislostmi, které uživatel nemusí konfigurovat ručně. Spring Boot automaticky konfiguruje jak základní framework Spring, tak i balíčky třetích stran. Ať už na základě uživatelské konfigurace, tak i na základě osvědčených postupů, což pomáhá předcházet chybám. Díky této funkci je umožněno začít rychle vyvíjet business logiku aplikace a neztrácet tolik času prvotní konfigurací. Tyto defaultní konfigurace lze samozřejmě kdykoliv upravit podle potřeb uživatele.

Spring Boot používá zaujatý přístup k přidávání a konfigurování startovacích závislostí na základě potřeb projektu uživatele. Zaujatý ve smyslu, podle nejlepších praktik a prověřených postupů. Dle úsudku Spring Bootu je vybráno, které balíčky budou nainstalovány a jaké výchozí hodnoty budou použity, místo toho, aby všechna tato rozhodnutí musel činit uživatel a vše nastavoval ručně. Potřeby projektu může uživatel definovat během inicializačního procesu, během kterého si vybere z několika startovacích závislostí, takzvaných spouštěčů (v originále Spring Starter), které pokrývají typické potřeby. Spring Boot Initializr lze spustit vyplněním jednoduchého webového formuláře, aniž by uživatel musel cokoli programovat nebo složitě konfigurovat. Například spouštěč Spring Web uživateli umožní vytvářet webové aplikace založené na platformě Spring s minimální konfigurací, přidáním všech potřebných závislostí

do projektu, například webového serveru Apache Tomcat. Další oblíbený spouštěč je Spring Security, který do aplikace automaticky přidá funkcionality autorizace, autentizace a řízení přístupu. Spring Boot obsahuje více než 50 těchto spouštěčů, a k dispozici je mnoho dalších spouštěčů třetích stran.

Spring Boot pomáhá vývojářům vytvářet aplikace, které lze jednoduše spustit a nasadit do produkčního prostředí. Umožňuje vytvářet standalone aplikace, které běží samy o sobě, bez závislosti na externím webovém serveru. Funguje to tak, že během inicializačního procesu je k aplikaci přidán webový server, například Tomcat nebo Netty, na kterém aplikace následně běží. Více informací o frameworku je dostupné v této dokumentaci [22].

Zdroje: [21], [22] .

3.2.6 Docker

Docker je open source platforma pro kontejnerizaci. Umožňuje vývojářům balit aplikace do kontejnerů, standardizovaných spustitelných komponent, které kombinují zdrojový kód aplikace s knihovnami operačního systému a závislostmi potřebnými ke spuštění tohoto kódu v libovolném prostředí. Kontejnery zjednodušují nasazování distribuovaných aplikací a stávají se stále populárnějšími tím, jak organizace přecházejí na vývoj v cloudu a hybridních multicloudových prostředích.

Kontejnery je možné realizovat díky izolaci procesů a virtualizačním schopnostem zabudovaným do Linuxového jádra. Funkcionality, jako jsou například řídicí skupiny (cgroups) pro přidělování prostředků mezi procesy, nebo jmenné prostory pro omezení přístupu procesů k jiným prostředkům či oblastem systému, umožňují více aplikačním komponentám sdílet prostředky jedné instance hostitelského operačního systému podobně, jako hypervizor umožňuje více virtuálním strojům sdílet procesor, paměť a další hardwarové prostředky jednoho fyzického serveru.

Technologie kontejnerů nabízí všechny funkce a výhody virtuálních počítačů, včetně izolace jednotlivých aplikací, nákladově efektivní škálovatelnosti, a další důležité výhody jako nižší paměťovou náročnost, větší efektivitu při využívání HW zdrojů a v neposlední řadě vyšší produktivitu vývojářů.

Na rozdíl od virtuálních strojů, kontejnery nenesou zátěž celé instance operačního systému a hypervizoru. Obsahují pouze procesy operačního systému a závislosti nezbytné k provádění požadovaného kódu. Velikost kontejnerů se pohybuje v megabajtech (oproti gigabajtům

u některých virtuálních strojů), lépe tedy využívají kapacitu hardwaru a mají rychlejší čas spouštění. S kontejnery lze na stejném hardwaru spustit několikanásobně více kopií aplikace než s virtuálními počítači. To může výrazně snížit výdaje za cloudové služby. V porovnání s virtuálními počítači se kontejnery rychleji a snadněji nasazují, spravují a restartují. Díky tomu jsou ideální pro použití v pipelinách kontinuální integrace a kontinuálního nasazení (CI/CD) a lépe se hodí pro vývojové týmy, které používají agilní postupy a DevOps.

V realizaci se technologie používá pro Data manager, kde je výsledná aplikace kontejnerizovaná a nasazována do cloudu. Více informací o nástroji je dostupné v této dokumentaci [23].

Zdroje: [23], [24].

3.2.7 MySQL

MySQL je open source systém pro správu relačních databází SQL. Stejně jako ostatní relační databáze ukládá MySQL data do tabulek tvořených řádky a sloupci. Uživatelé mohou definovat, manipulovat, řídit a dotazovat se na data pomocí strukturovaného dotazovacího jazyka, známého pod zkratkou SQL. Název systému vznikl spojením slov „My“, jméno dcery spoluzakladatele, a SQL, zkratka Structured Query Language, což je programovací jazyk, který pomáhá přistupovat k datům v relační databázi a spravovat je.

MySQL databáze byla původně uvedena na trh již v roce 1995. Od té doby prošla několika změnami vlastníka/správce, než jí v roce 2010 koupila společnost Oracle Corporation, která ji dosud vlastní. Ačkoli je nyní ve vedení společnosti Oracle, MySQL je stále open source software, což znamená, že je možné ho volně používat a upravovat.

V rámci implementace je použita jako primární uložiště dat. Více informací o systému je dostupné v této dokumentaci [25].

Zdroje: [25], [26].

3.2.8 JDBC

Ovladač JDBC využívá rozhraní JDBC (Java Database Connectivity) API vyvinuté společností Sun Microsystems, která je nyní součástí společnosti Oracle, a poskytuje standardizovaný způsob přístupu k datům pomocí programovacího jazyka Java. Pomocí JDBC může aplikace přistupovat k různým druhů databází a běžet na jakékoli platformě s virtuálním strojem Java. Pro přístup k různým databázovým systémům není nutné psát samostatné

aplikace. Použití JDBC umožňuje napsat jednu aplikaci, která může posílat příkazy SQL různým zdrojům dat.

Podnikové aplikace vytvořené pomocí technologie Java EE potřebují komunikovat s databázemi pro ukládání informací specifických pro danou aplikaci. Interakce s databází tedy vyžaduje efektivní připojení k databázi, kterého lze dosáhnout pomocí ovladače ODBC (Open database connectivity). Tento ovladač se používá s rozhraním JDBC k interakci nebo komunikaci s různými druhy databází, jako jsou databáze Oracle, MS Access, MySQL, MariaDB a Microsoft SQL server.

V implementaci je použita technologie v modulu Data collector. Více informací o knihovně je dostupné v této dokumentaci [27].

Zdroje: [27].

3.2.9 Liquibase

Liquibase je open source knihovna pro verzování změn v databázi. Podporuje většinu známých databází a pro definici struktury databáze lze použít různé souborové formáty. Nejpopulárnější funkcí Liquibase je její schopnost vrátit změny zpět do určitého bodu a tím odpadá potřeba si pamatovat, jaká byla poslední změna/skript, která byla na konkrétní instanci DB provedena.

Liquibase používá skripty označované jako changeset jakožto jednotku změn provedených v databázi. Changesety mohou být v různých formátech včetně XML, JSON, YAML a SQL. Během vývojového cyklu aplikace a v průběhu jejího životního cyklu probíhají ve struktuře DB různé úpravy a vylepšení, vytváří se tedy další changesety. Z tohoto důvodu existuje hlavní soubor, takzvaný changelog, který obsahuje seznam všech changesetů. Současně jsou v databázi specializované tabulky, ve kterých jsou uchovávaný záznamy za každý aplikovaný changeset.

Je doporučována integrace systému Liquibase do běžně využívaného systému verzování kódu (například gitu). Za pomocí platformy pro kontinuální integraci lze synchronizovat verze databáze s verzí aplikace.

V rámci implementace je Liquibase využíván pro verzování primárního schématu databáze a výchozích dat. Více informací o nástroji je dostupné v této dokumentaci [29].

Zdroje: [28], [29].

3.2.10 Pi4j

Pi4j je knihovna, která poskytuje Java vývojářům funkcionálně bohaté, výkonné, ale zároveň na použití jednoduché, objektově orientované API pro práci s I/O platformy RaspberryPi. Dosahuje toho abstrakcí nad nízko-úrovňovým nativním přístupem tak, aby se programátoři mohli soustředit na implementaci business logiky svých aplikací.

Projekt Pi4j byl zahájen v roce 2012, tedy ve stejném roce jako byl představen první počítač Raspberry Pi. Knihovna abstrahuje nad běžně používanými I/O rozhraními včetně GPIO, I2C, SPI, PWM a sériovou komunikací.

V rámci řešení je Pi4j použito pro implementaci obsluhy a nízko-úrovňové komunikace se zařízeními a dalšími prvky. Více informací o knihovně je dostupné v této dokumentaci [30].

Zdroje: [30].

4 NÁVRH A IMPLEMENTACE SYSTÉMU

Praktická část se zaměřuje na implementaci obecného, autonomního systému, který by měl být schopný vyhodnocovat vstupy, ať už uživatelské, získané monitorováním okolního prostředí, nebo periodicky se opakující. Tyto vstupy by měl být schopen analyzovat a na základě uživatelských konfigurací by měl být schopen reagovat a řídit okolní prvky, které interagují s okolním prostředím. Navrhovaný systém by měl být komplexní řešení obsahující několik jak softwarových, tak hardwarových modulů. Systém bude konfigurován na míru určitému řešení, nicméně bude disponovat vysokou obecností a konfigurovatelností tak, aby byl flexibilní pro různé typy řešení a požadavků.

4.1 Aktuální stav

K bližšímu nastínění řešené problematiky, tedy toho, co bude systém vlastně řešit a k jakým vylepšením dojde, bude popsán aktuální stav.

V aktuálním řešení se vyskytuje několik prvků, které jsou schopny interagovat a ovlivňovat okolní prostředí. Jmenovitě jde o jednotku tepelného čerpadla pro topení a chlazení a na ní napojený systém provětrávání se zpětnou rekuperací tepla. Tato zařízení jsou aktuálně řízena buď manuálně, popřípadě s využitím časových plánů.

Jednotka tepelného čerpadla je řízena nástěnným ovladačem a zároveň je napojena na WiFi modul, který slouží jako rozhraní pro vzdálený přístup. Přes tyto dva prvky je možno nastavit určitou autonomnost v podobě jednoduchých časových plánů. Jejich funkčnost si lze představit jako funkci běžného termostatu. Bez specializovaného softwaru není možné tento systém na dálku monitorovat ani nijak řídit. Také možnosti autonomního režimu jsou značně limitovány. Vyhodnocování teploty probíhá v místě, kde se řídicí prvky nachází, v aktuálním řešení se nachází v technické místnosti. Vzhledem k tomu naměřené hodnoty nereflektují hodnoty, které by chtěl uživatel monitorovat a reagovat na ně například z místností, ve kterých tráví většinu času.

Rekuperační jednotka je řízena napřímo pomocí relé modulů. Je relativně jednoduchá na řízení, lze na ní nastavit pouze režim zapnuto/vypnuto a síla, s jakou pracuje. U tohoto zařízení se vyskytují stejné problémy, jako u předchozích zařízení, a to absence možnosti vzdáleného monitorování, autonomního a vzdáleného řízení.

Jak bylo zmíněno, ani jedno ze zařízení není aktuálně možno vzdáleně monitorovat, či řídit. Nelze také efektivně synchronizovat součinnost všech zařízení a není ani možné nastavit složitější autonomní řízení, ve smyslu složitějších reakcí na stav prostředí a konfigurací propracovaných časových plánů. Kromě vzdáleného monitorování zařízení, aktuálně také není možné vzdáleně monitorovat prostředí jako takové.

4.2 Návrh aplikace

Motivace implementace systému je změnit stav popsaný v předchozí kapitole tak, aby byly primárně odbourány zmíněné nedostatky a došlo k zjednodušení řízení stávajících komponent. Cílem je potom vytvořit komplexní systém, který by poskytoval jednotné rozhraní pro řízení všech spravovaných zařízení a v podstatě by vystupoval jakožto implementace chytré domácnosti. V této podkapitole bude krátce tento systém a jeho části popsány, jednotlivé části budou v dalších kapitolách ještě rozebrány podrobněji. Systém se jako takový bude skládat ze tří modulů: senzorové části, Data manageru a Data collectoru.

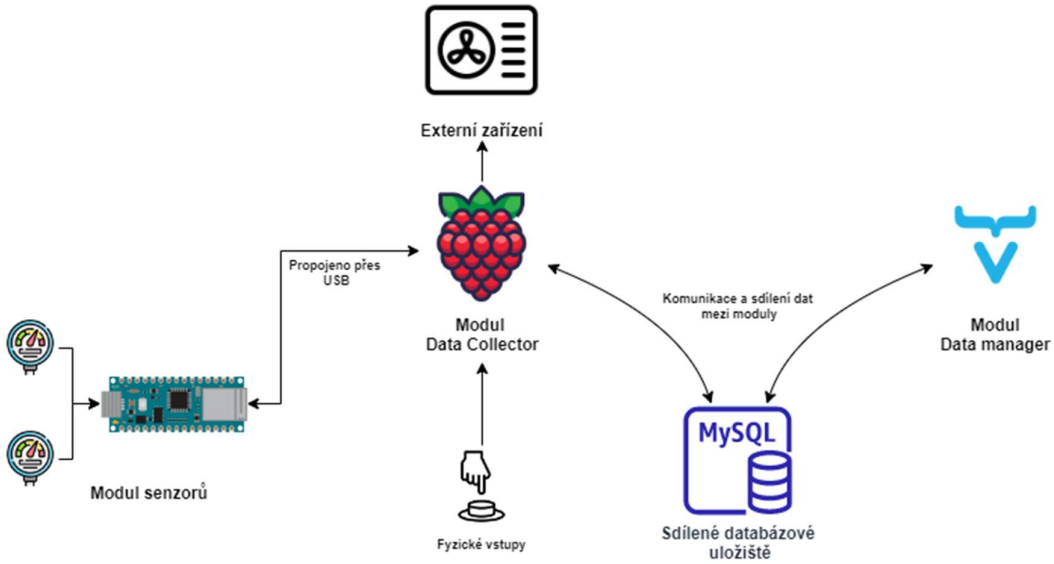
První z nich je senzorová část, tedy modul, který obsahuje senzory, zařízení (která je obsluhují a zajišťují sběr dat z okolního prostředí) a prvky (které zajišťují nízko-úrovňovou infrastrukturu pro bezdrátovou komunikaci). Tato infrastruktura je využita pro posílaní dat z okolního prostředí do centrální jednotky a pro řízení externích zařízení, která budou interagovat s okolním prostředím. Tento modul je napojen přímo na modul Data collector.

Dalším modulem je Data manager, s nímž přijde uživatel nejvíce do kontaktu. Tento modul slouží jak ke vzdálenému monitoringu spravovaných zařízení a okolního prostředí systému, tak ke konfiguraci plánů, které systém využívá k řízení okolních prvků. K dispozici jsou limitní plány, časové plány a několik typů manuálních plánů. Modul také obsluhuje konfiguraci takzvaných eventů, tj. událostí a jim příslušných akcí, které jsou vlastně konfigurace toho, jakým způsobem se bude reagovat, pokud bude některý z plánu aktivován. Další, co se dá v modulu pozorovat je aktuální stav stavového prostoru, tedy toho, co je aktuálně na výstupech systému nastavené, přehled registrovaných senzorů a další konfigurace. Tento modul je tvořen tak, aby se dal veřejně vystavit a byl tak dostupný odkudkoliv bude uživatel potřebovat. Všechny zmíněné části budou podrobněji popsány v kapitole 4.2.3.

Posledním, ale zároveň nejdůležitějším modulem je Data collector. Tento modul by se dal považovat za ústřední část a v podstatě mozek celého systému. Modul registruje senzorové moduly a přijímá data z okolního prostředí, která mu jsou od nich zaslány. Nad nimi provádí

agregační operace a následně agregovaná data ukládá do databáze. Modul analyzuje nakonfigurované plány a rozhoduje, které z nich budou aktivovány. V neposlední řadě také zajišťuje reakci na mechanické prvky k němu připojené, komunikaci s externími zařízeními, retenci dat a další funkcionality, které budou podrobněji popsány v rámci kapitoly 4.2.2.

Celý systém používá centrální databázové uložiště dat, které slouží jak k ukládání samotných dat, tak k zajištění komunikace a předávání informací mezi moduly.



Obrázek 1 Diagram obsahující koncepci celkového systému

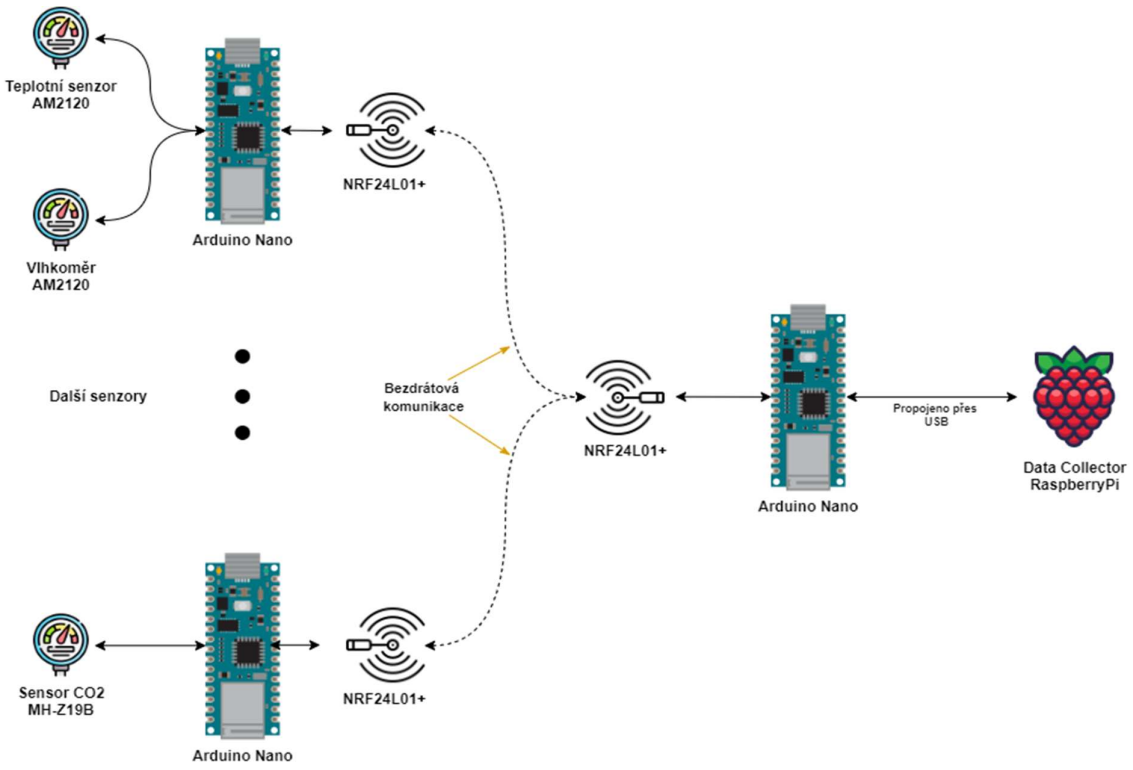
4.2.1 Senzorový modul

V této části proběhne bližší seznámení s návrhem senzorového modulu, který byl obecně přestaven v předchozí kapitole. Samotný modul by se dal rozdělit na dvě části: na část měřící hodnoty z okolního prostředí a na část, které poskytuje komunikační infrastrukturu. Pro obě části jsou nosnými prvky vývojové desky Arduino Nano.

V aktuálním návrhu by měla první část, měřící hodnoty okolního prostředí, schopna měřit teplotu, vlhkost a obsah oxidu uhličitého. Pro měření hodnot je v případě teploty a vlhkosti využit senzor AM2120, který je schopen měřit obě veličiny. V případě oxidu uhličitého je využit senzor MH-Z19B. Oba tyto senzory jsou blíže představeny v předchozích kapitolách 3.1.1 a 3.1.2. Tyto senzory jsou připojovány přímo na Arduino Nano. V aktuální implementaci jsou využita dvě Arduina Nano, pro každý senzor jedno. Nicméně software implementovaný pro tyto prvky zajišťuje, že oba senzory lze připojit zároveň k jednomu Arduinu. Ke každému

z Arduin je také připojen prvek NRF24L01+, ten bude popsán v další části. Přesný popis implementace softwaru, připojení prvku a schéma je popsáno v implementační kapitole 4.4.1.

Druhá část zajišťuje komunikační infrastrukturu, řeší sběr dat naměřených v první části a jejich odesílání do modulu Data collector. Hlavním elementem této části je Arduino Nano, ke kterému je připojen pouze jeden prvek, a to NRF24L01+. Prvek NRF24L01+ byl zmíněn už v předchozí části a je využíván k zajištění bezdrátové komunikace. Samotné Arduino je pak napřímo, pomocí USB rozhraní, připojeno na zařízení Raspberry Pi, které obsahuje modul Data collector. Funkce softwaru na Arduino Nano zajišťuje navazování bezdrátové komunikace s ostatními Arduiny, které obsluhují senzory. Z nich periodicky, dle konfigurace, zjišťuje naměřené hodnoty a ty zasílá přes USB rozhraní do modulu Data collector.



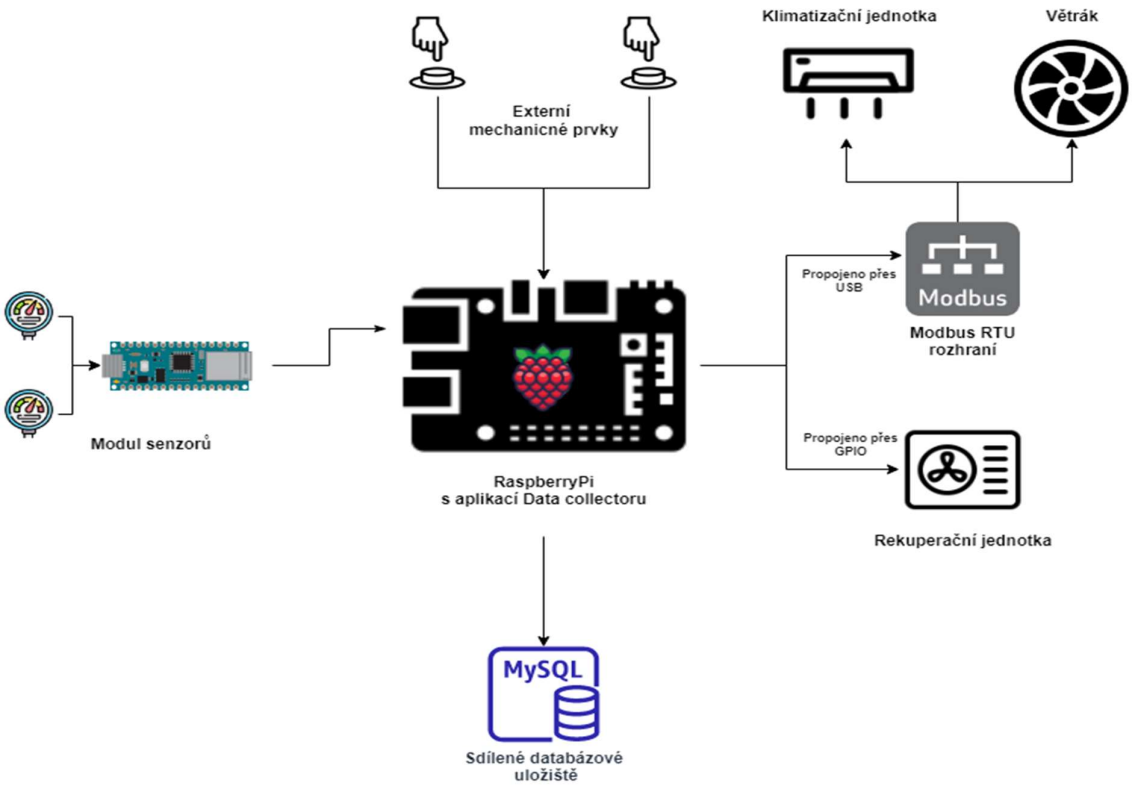
Obrázek 2 Diagram obsahující koncepci senzorového modulu

V rámci aktuálního návrhu jsou celkem využita tři Arduino Nano. Dvě jsou využívána pro obsluhu senzorů a jedno jakožto ústřední prvek, které ostatní ovládá a shromažďuje od nich data. Aktuální implementace však umožňuje mít zařízení pro obsluhu senzorů až pět. Software byl ovšem navrhován tak, aby šel v budoucnu upravit a těchto prvků bylo možno využít ještě více. K tomu je potřeba doimplementovat další část, tzv. repeater, který by umožnil síť dále členit. Tato část by byla prostředníkem mezi zařízeními obsluhující senzory a ústředním

prvkem. Jeden prvek napojený na ústřední část by tedy mohl obsluhovat dalších pět prvků obsluhujících senzory. Nicméně tato část nebyla pro aktuální řešení třeba a není implementována.

4.2.2 Data collector

V této části je z blízka představen modul Data collector, jeho části a funkcionality. Tento modul lze považovat za mozek celého systému, jelikož na základě uživatelských konfigurací řídí všechny ostatní prvky. Nejpodstatnější částí modulu je software běžící na Raspberry Pi, k němu je pak připojen modul senzorů, externí mechanické prvky a zařízení, která jsou modulem řízeny. Zařízení je také připojeno k databázovému uložišti dat, do kterého ukládá hodnoty a pomocí kterého je zajištěna komunikace s modulem Data manager.



Obrázek 3 Diagram obsahující koncepci modulu Data collector

Nejprve bude popsán samotný software a služby, které poskytuje. Služby by se daly rozdělit podle toho, co obsluhují do tří kategorií: na služby obsluhující modul senzorů, služby vyhodnocující uživatelské plány a reakce na ně a služby řídící komunikaci s ostatními prvky a zařízeními. Dále budou představeny externí prvky a obsluhované zařízení.

Služby obsluhující modul senzorů existují dvě. První služba zajišťuje zjištění a navázání komunikace s modulem senzorů. Po úspěšném navázaní komunikace přichází na řadu služba druhá, ta shromažďuje obdržená data z modulu senzorů, agreguje je a zajišťuje uložení do databáze.

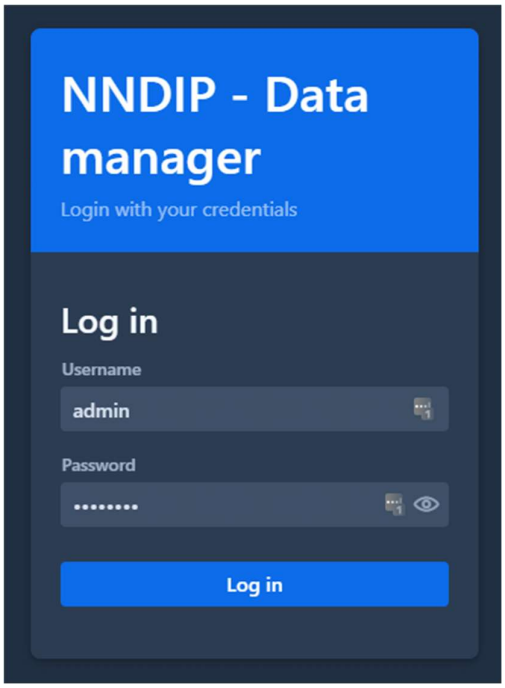
Služby vyhodnocující uživatelské plány a připravující odpovídající reakce jsou v systému také dvě. První periodicky prochází povolené plány a dle událostí v systému vyhodnocuje, zda jsou plány aktivní, či nikoliv. Všechny plány, které jsou vyhodnoceny jako aktivní jsou předány do druhé služby. Tato služba je nejdůležitější z pohledu řízení systému. Na základě priorit plánů získaných z předchozí služby a dalších konfigurací vyhodnotí, jaké stavy se nastaví do spravovaných registrů, respektive jaké výstupy budou zapsány do externích zařízení. Bližší seznámení s plány proběhne v následující kapitole, podrobnosti o jejich vyhodnocování a vyhodnocování hodnot, které budou zapsány do registrů, budou v implementační části.

Poslední druh služeb zajišťuje komunikaci s externími prvky. Jeden poddruh z tohoto typu služeb zajišťuje komunikaci s prvky poskytujícími vstupy do tohoto modulu a se spravovanými zařízeními, která řídí nebo jakkoliv interagují s okolním prostředím. Implementace těchto služeb poskytuje primárně rozhraní pro vstupy zařízení Raspberry Pi, resp. jeho GPIO a pro protokol Modbus RTU. Rozhraní pro GPIO zajišťuje zpracování jak vstupních hodnot, tak hodnot výstupních. Vstupní hodnoty v popisované implementaci produkují mechanické prvky připojené do systému. Výstupní hodnoty jsou pak v řešení využívány pro řízení rekuperační jednotky. Rozhraní pro Modbus RTU slouží v implementaci pouze k zapisování výstupních hodnot, tedy k řízení externích prvků. V řešení jde o klimatizační jednotku a na ní připojený systém provětrávání. Druhý poddruh služeb se věnuje komunikaci s databází. Převážně jde o typickou repositářovou vrstvu, která zajišťuje získávání, ukládání a modifikaci spravovaných datových entit vyskytujících se v softwaru modulu. Tyto služby využívají technologie JDBC s možností opětovného připojení při odpojení od databáze. Tento modul by měl být tedy schopen pracovat i v offline režimu, bez nutnosti získání dat z databáze. Tato funkcionalita je ovšem aktuálně experimentální a vyskytují se v ní menší problémy, které je nutno doladit. S tímto typem služeb je provázána ještě služba zajišťující retenci dat. Vzhledem k velkému objemu dat, který jsou senzory schopny vyprodukovat, by mohlo dojít k zaplnění databáze a k nemožnosti systému korektně pracovat. Z toho důvodu tato služba periodicky, dle konfigurace uživatele, promazává senzorová data.

4.2.3 Data manager

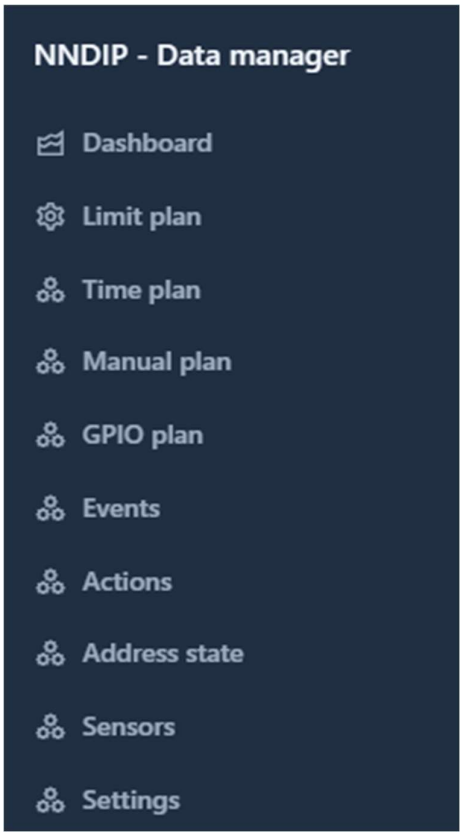
Data manager je z pohledu uživatele nejdůležitější modul. Umožňuje vzdálenou kontrolu aktuálního stavu monitorovaného prostředí a spravovaných zařízení, konfiguraci autonomního režimu a plánů, které řídí reakci na události vznikající v systému. Celý modul je implementován tak, aby ho bylo možno veřejně vystavit v cloudové službě a byl bezpečný na používání. Sdílení dat, respektive konfigurací, tohoto modulu s modulem Data collector je zajištěno pomocí sdíleného databázového uložiště.

Modul je cílen na vystavení v cloudové službě, je tedy nutné, aby byl přístup řízen a umožněn jen autorizovaným osobám. Pokud by nebyl zabezpečen, mohl by do něj přistupovat kdokoliv a zjišťovat si údaje o řízeném systému, v horším případě škodit a přenastavit konfigurace. Z tohoto důvodu je modul zabezpečen standardní správou uživatelů, kdy je každý, kdo chce do modulu přistoupit nucen se přihlásit. Přihlašovací okno lze vidět na Obrázek 4.



Obrázek 4 Přihlašovací okno modulu Data manager

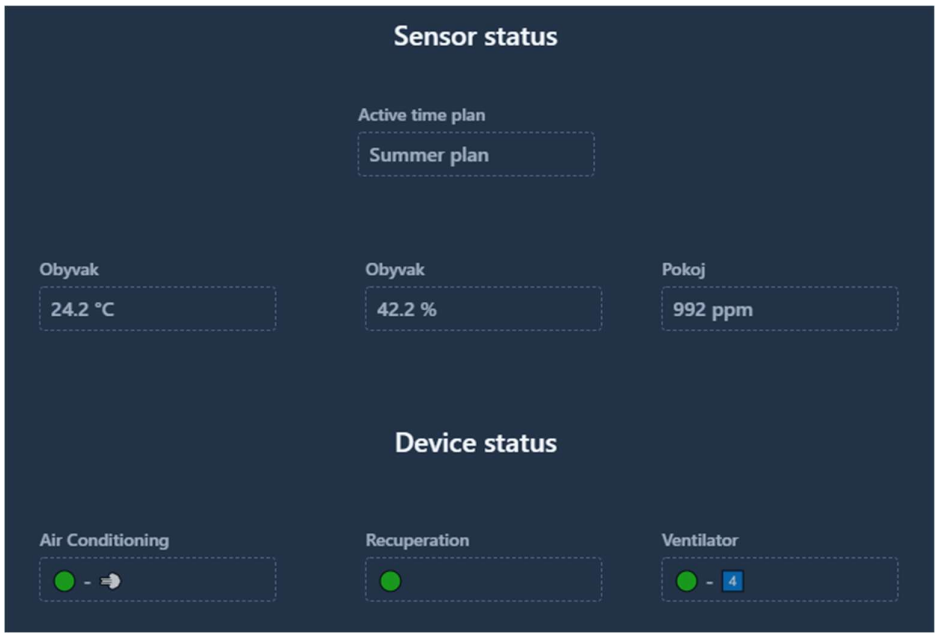
Po přihlášení získá uživatel přístup k funkcionalitám modulu. Jejich seznam lze vidět na Obrázek 5. Tyto funkcionality lze rozdělit do čtyř skupin. V první části lze pozorovat stav prostředí a obsluhovaných prvků, v druhé části se nastavují plány reagující na události v systému, v třetí části se definující eventy společně s jejich elementy a poslední část slouží ke konfiguraci samotného modulu.



Obrázek 5 Přehled funkcionalit modulu Data manager

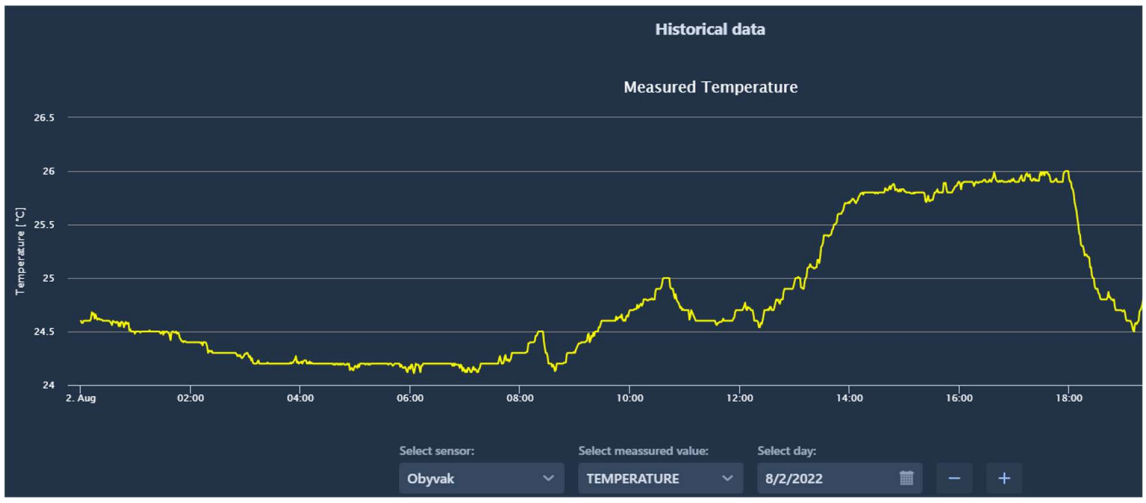
První skupina funkcionalit, věnující se monitorováním stavu prostředí a obsluhovaných prvků, je obsažena na stránce Dashboard a Address state. Stránka Dashboard je úvodní stránka, která se uživateli zobrazí při přístupu do modulu. Kvůli tomu, že se v této stránce vyskytují dynamicky se měnící data, jsou data periodicky načítána a tím se dosahuje zobrazení stále aktuálních dat. Tato stránka obsahuje informace o tom, jaký je aktuálně aktivní limitní plán,

přehled vybraných senzorů a jejich naměřených dat a informace o tom, v jakém režimu se aktuálně nacházejí spravovaná zařízení (viz Obrázek 6).



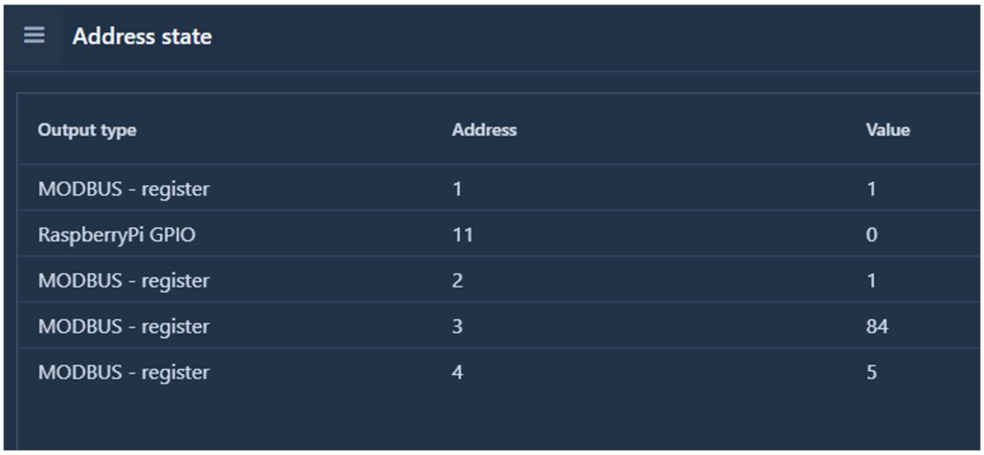
Obrázek 6 První část stránky Dashboard

Dále se na stránce Dashboard nachází grafické znázornění naměřených hodnot z prostředí a jejich vývoj v čase. Zde je možné vybrat jakýkoliv senzor, dále jakou hodnotu chce uživatel pozorovat a den, v kterém byly hodnoty naměřeny (viz Obrázek 7).



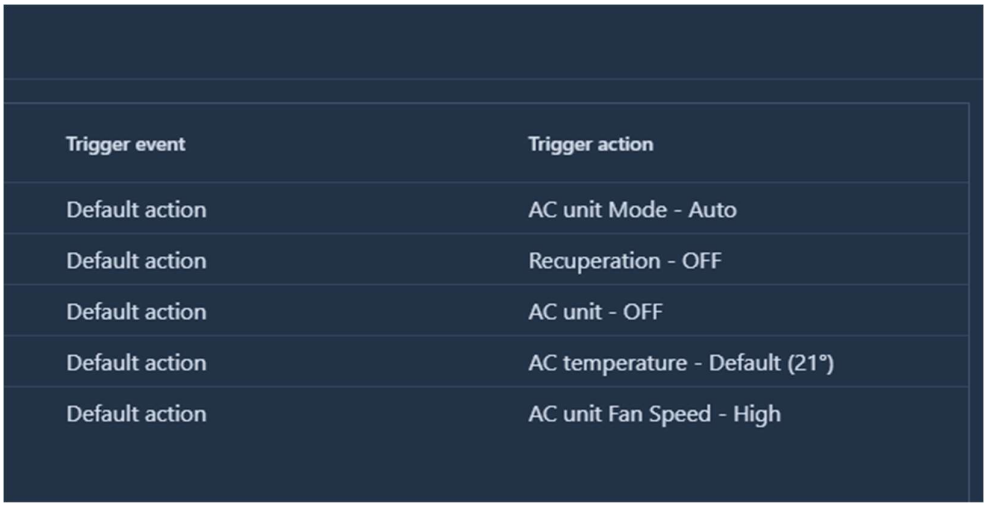
Obrázek 7 Druhá část stránky Dashboard

Druhá stránka využitá k monitorování systému, nazývající se Address state, slouží k podrobnému monitorování všech aktuálně spravovaných výstupů. Převážně tedy k monitorování výstupů na rozhraní GPIO a rozhraní Modbus RTU.



Obrázek 8 Levá část přehledu Address state

V tomto přehledu se nachází tabulka, ve které je pro každý spravovaný výstup jeden záznam, který obsahuje několik atributů. První atribut je typ výstupu, ten může být buď typu GPIO Raspberry nebo Modbus, který se dělí ještě na typ „coil“ a „register“. Dalším atributem je pak adresa výstupu v daném rozhraní a hned za ním následuje atribut s hodnotou, která se na danou adresu zapíše. Nyní zmíněné atributy lze vidět na Obrázek 8. Tabulka obsahuje ještě další dva atributy viz Obrázek 9. První z nich obsahuje informaci o tom, jaký event nastavil hodnotu na danou adresu. Druhý z nich pak informuje o tom, jaká akce z eventu zmíněnou hodnotu nastavila.



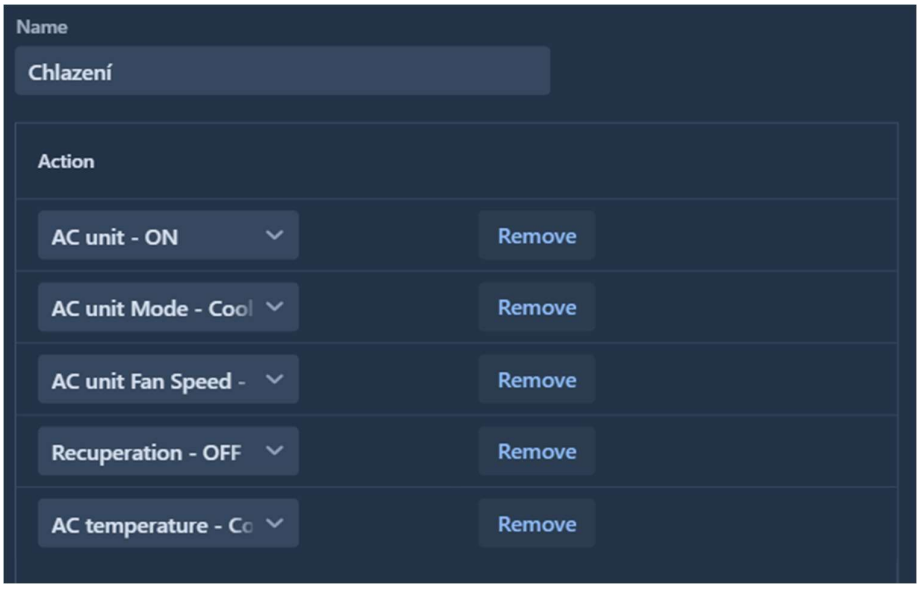
Obrázek 9 Pravá část přehledu Address state

Další skupina obsahuje stránky využité ke konfiguraci eventů a s nimi souvisejících akcí. Akce jsou jednotky definic výstupu, které obsahují informace o tom, na jaký výstup se zapíše definovaná hodnota. Event pak není nic jiného než skupina, která sdružuje jednotlivé akce do logických částí.

Name	Output type	Address	Value	Is default?
AC unit - ON	MODBUS - register	2	2	✗
AC unit - OFF	MODBUS - register	2	1	✓
AC unit Mode - Auto	MODBUS - register	1	1	✓
AC unit Mode - Cool	MODBUS - register	1	2	✗
AC unit Mode - Dry	MODBUS - register	1	3	✗
AC unit Mode - Heat	MODBUS - register	1	4	✗
AC unit Mode - Fan	MODBUS - register	1	5	✗

Obrázek 10 Přehled tabulky akcí na stránce Actions

Pro představu event chlazení sdružuje pět akcí, kde jedna z nich zapne klimatizaci, další z nich ji uvede do režimu chlazení, další nastaví systém provětrávání, další vypne rekuperaci a poslední nastaví požadovanou teplotu. Všechny tyto akce pak logicky utváří skupinu akcí, které jsou potřeba vykonat, aby byl systém uveden do režimu chlazení.

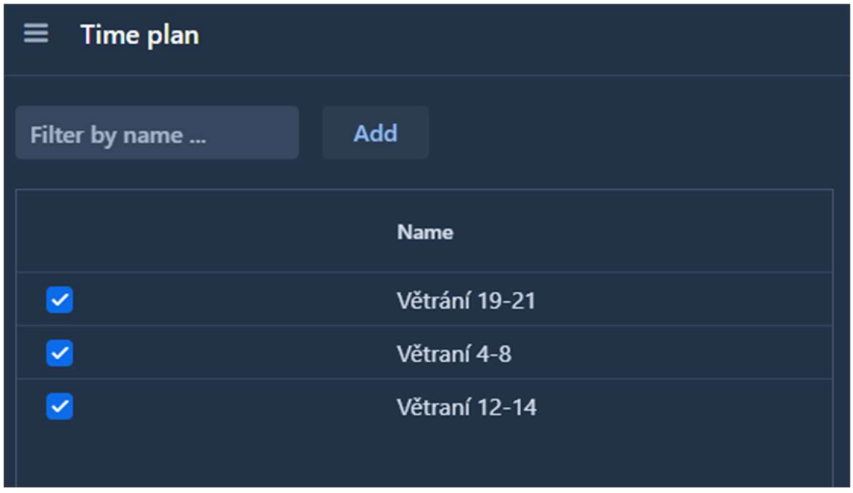


Obrázek 11 Ukázka definice eventu Chlazení

Třetí skupina funkcionalit obsahuje nástroje, které slouží ke konfiguraci různých typů plánů, ty pak řídí reakce systému na různé události. V systému se nachází čtyři typy plánů a v modulu lze nalézt jim příslušné stránky, ve kterých je lze konfigurovat. Každý plán reaguje na různé typy událostí, ale mají společné, že každý z nich má přiřazenou prioritu a event, tedy souhrn akcí (více popsáno v předchozích odstavcích). Takto přiřazené eventy, respektive jim přiřazené akce, jsou pak zpracovávány v modulu Data collector a pokud jsou vyhodnoceny jako aktivní, tak jsou jejich hodnoty zapsány na odpovídající výstupy. Stránky jednotlivých plánů se od sebe moc vizuálně neliší, jejich přehledové části jsou podobny zobrazení na Obrázek 12. Jedinou výjimkou je limitní plán (viz Obrázek 43).

První z nich je Limit plan, tento typ plánu svou funkcí simuluje funkci termostatu. Jsou v něm definovány tři hraniční hodnoty značící příliš vysokou teplotu, příliš nízkou teplotu a příliš vysokou hodnotu oxidu uhličitého v prostředí. Každá z těchto hraničních hodnot definuje jeden limitní plán, má tedy přiřazený event. Při vyhodnocování plánu jsou pak hraniční hodnoty porovnávány se skutečnými naměřenými hodnotami a pokud překročí definované hranice, tak je daný plán brán jako aktivní.

Druhý typ plánu je Time plan neboli časový plán. Tento typ plánu funguje v nakonfigurovaných časových oknech. Lze v něm tedy nastavit čas od kdy do kdy bude daný plán považovaný za aktivní. V aktuálním řešení se používá například pro každodenní provětrávání, které je požadováno vždy ve stejný čas.



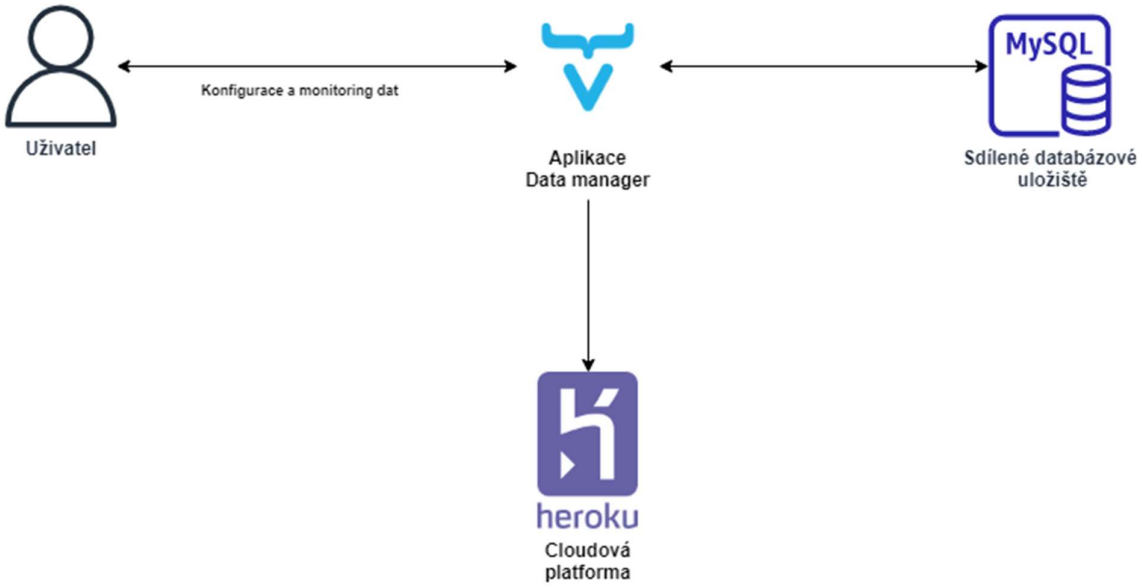
Obrázek 12 Zobrazení tabulky časových plánů na stránce Time plan

Dalším typem plánu je Manual plan. To je typ manuálního plánu, který může uživatel ručně spouštět a vypínat přímo z modulu Data manager. Tomuto typu je velice podobný i poslední

typ, GPIO plan. Tento typ plánu by se dal také považovat za typ manuálního plánu, ovšem v trochu jiném smyslu. Tento plán je navázán na vstupy definované na rozhraní GPIO, v případě tohoto řešení jsou na vstupy Raspberry Pi napojená tlačítka a spínače, pomocí kterých lze plány manuálně spouštět. Rozdíl mezi Manual plan a GPIO plan je takový, že první využívá manuální spouštění v softwaru modulu a druhý využívá hardwarové prvky. GPIO plán pak obsahuje ještě dva podtypy plánů, respektive dvě různá chování, vykonávána při obdržení signálu z GPIO rozhraní. První z nich nazvaný Manual, funguje jako klasický spínač, druhý typ je nazvaný Time, funguje tak, že při obdržení signálu je plán aktivní po dobu definovanou v plánu.

Poslední skupina funkcionalit slouží k modifikaci samotného modulu Data manager. Nachází se na stránce Settings, kde lze nastavovat, jaké senzory a jaký druh jejich hodnot budou zobrazovány na dashboardu. Dále se zde pro spravované zařízení nastavují adresy a typ rozhraní, ze kterých se berou hodnoty a na základě těch se pak určuje stav jim odpovídajících zařízení. Tyto stavy jsou pak také zobrazeny v Dashboardu.

Jak bylo popsáno, modul Data manager je kompletně separován od ostatních modulů systému. Ve většině případů se nenachází ani ve stejné síti, je tedy nutno řešit, jakým způsobem sdílet data mezi moduly. Tento problém byl vyřešen používáním sdíleného databázového uložiště.



Obrázek 13 Diagram obsahující koncepci modulu Data manager

4.2.4 Funkční požadavky

- FP 1 – Požadavky týkající se senzorového modulu
    - FP 1.1 – Modul musí být schopný měřit teplotu
    - FP 1.2 – Modul musí být schopný měřit vlhkost
    - FP 1.3 – Modul musí být schopný měřit obsah oxidu uhličitého
    - FP 1.4 – Modul musí být schopný zajistit bezdrátovou komunikaci mezi jednotlivými částmi modulu
- FP 2 – Požadavky týkající se modulu Data collector
    - FP 2.1 – Požadavky týkající se senzorů a zpracování senzorových dat
        - FP 2.1.1 – Modul musí být schopný rozpoznat a dynamicky připojovat senzorové moduly
        - FP 2.1.2 – Modul musí mít možnost nastavit periodu s jakou bude kontrolováno připojení nových senzorových modulů
        - FP 2.1.3 – Modul musí být schopný zpracovávat data obdržená ze senzorů a ty následně ukládat
        - FP 2.1.4 – Modul musí mít možnost nastavit periodu s jakou budou data obdržená ze senzorů zpracovávána
    - FP 2.2 – Požadavky týkající se vyhodnocování plánů
        - FP 2.2.1 – Modul musí být schopný vyhodnotit jaké plány jsou aktivní
        - FP 2.2.2 – Modul musí být schopný rozpoznat, který z aktivních plánů má největší prioritu, brát v potaz další konfigurace a na základě výsledku rozhodnout, které akce budou vykonány
        - FP 2.2.3 – Pro potřeby vyhodnocování limitních plánů, musí mít modul možnost nakonfigurovat čas, po který budou data ze senzoru považována za aktuální
        - FP 2.2.4 – Modul musí mít možnost nakonfigurovat defaultní limitní plány
        - FP 2.2.5 – Modul musí mít možnost nakonfigurovat defaultní akce

        - FP 2.2.6 – Modul musí mít možnost nakonfigurovat pravidla pro vzájemné vylučování akcí
        - FP 2.2.7 – Modul musí mít možnost nastavit periodu s jakou budou plány vyhodnocovány
    - FP 2.3 – Požadavky týkající se externích vstupů a výstupů
        - FP 2.3.1 – Modul musí být schopný komunikovat přes Raspberry GPIO rozhraní
        - FP 2.3.2 – Modul musí být schopný registrovat listenery na Raspberry GPIO vstupy
        - FP 2.3.3 – Modul musí mít možnost nastavit periodu s jakou budou zjišťovány nové konfigurace listenerů a ty budou následně registrovány
        - FP 2.3.4 – Modul musí být schopný komunikovat přes Modbus RTU rozhraní
        - FP 2.3.5 – Modul musí mít možnost nakonfigurovat parametry potřebné pro připojení Modbus RTU rozhraní
        - FP 2.3.6 – Modul musí ukládat informace o tom jaké hodnoty a kterým eventem (resp. akcí), byly uloženy na adresy externích zařízení
    - FP 2.4 – Požadavky týkající se retence senzorových dat
        - FP 2.4.1 – Modul musí být schopný promazávat senzorová data
        - FP 2.4.2 – Modul musí mít možnost nakonfigurovat periodu s jakou budou data promazávána
        - FP 2.4.3 – Modul musí mít možnost nakonfigurovat expiraci dat, po které budou odstraněna
- FP 3 – Požadavky týkající se modulu Data manager
    - FP 3.1 – Požadavky týkající se stránky Dashboard
        - FP 3.1.1 – Modul musí umožňovat sledování aktuální hodnoty vybraných senzorů
        - FP 3.1.2 – Modul musí umožňovat nastavit jaké senzory a jaký jejich typ dat bude viditelný na Dashboardu

        - FP 3.1.3 – Modul musí umožňovat nakonfigurovat čas, po který budou data ze senzoru považována za aktuální
        - FP 3.1.4 – Modul musí umožňovat sledování aktuálního stavu vybraných spravovaných zařízení
        - FP 3.1.5 – Modul musí umožňovat nastavení vstupů podle jejichž hodnot se bude odvozovat stav spravovaných zařízení
        - FP 3.1.6 – Modul musí informovat o aktuálně používaném období časového plánu
        - FP 3.1.7 – Modul musí umožňovat sledování historických dat ze senzorů v grafické podobě
        - FP 3.1.8 – Modul musí poskytovat automatické obnovování zobrazovaných hodnot
        - FP 3.1.9 – Modul musí mít možnost nastavit periodu s jakou budou zobrazované hodnoty obnovovány
    - FP 3.2 – Obecné požadavky týkající se konfigurace plánů
        - FP 3.2.1 – Modul musí poskytovat funkce pro přehled, tvorbu, modifikaci a mazání konfigurací všech typů plánů
        - FP 3.2.2 – Modul musí umožňovat povolení a zakázání jednotlivých vykonávání plánů
        - FP 3.2.3 – Modul musí umožnit nastavit plánům prioritu
        - FP 3.2.4 – Modul musí umožnit nastavit plánům event, který bude navázán na aktivaci plánu
    - FP 3.3 – Speciální požadavky týkající se konfigurace typu plánů Limit plan
        - FP 3.3.1 – Modul musí umožnit konfigurovat reakci systému, která bude spuštěna při zjištění hodnoty teploty vyšší než nastavená hranice
        - FP 3.3.2 – Modul musí umožnit konfigurovat reakci systému, která bude spuštěna při zjištění hodnoty teploty nižší než nastavená hranice
        - FP 3.3.3 – Modul musí umožnit konfigurovat reakci systému, která bude spuštěna při zjištění obsahu oxidu uhličitého vyššího než nastavená hranice

        - FP 3.3.4 – Pro výše zmíněné funkcionality musí systém poskytovat přepínač mezi zimním a letním obdobím
    - FP 3.4 – Speciální požadavky týkající se konfigurace typu plánů Time plan
        - FP 3.4.1 – Modul musí umožňovat nastavení časového rozsahu, ve kterém bude tento plán vykonáván
        - FP 3.4.2 – Výše zmíněný časový rozsah musí být aplikovatelný i přes půlnoc (např. 22:00–02:00)
    - FP 3.5 – Speciální požadavky týkající se konfigurace typu plánů Manual plan
        - FP 3.5.1 – Modul musí umožňovat spouštět tento typ plánu přímo ze svého prostředí
    - FP 3.6 – Speciální požadavky týkající se konfigurace typu plánů GPIO plan
        - FP 3.6.1 – Modul musí umožňovat konfiguraci adresy GPIO a jeho defaultního napětí
        - FP 3.6.2 – Modul musí umožňovat konfiguraci subtypu tohoto plánu, tzv. Manual, který se bude chovat jako klasický přepínač
        - FP 3.6.3 – Modul musí umožňovat konfiguraci subtypu tohoto plánu, tzv. Time, který bude po sepnutí aktivní po nastavenou dobu
    - FP 3.7 – Požadavky týkající se konfigurace eventů
        - FP 3.7.1 – Modul musí poskytovat funkce pro přehled, tvorbu, modifikaci a mazání konfigurací eventů
    - FP 3.8 – Požadavky týkající se konfigurace akcí
        - FP 3.8.1 – Modul musí poskytovat funkce pro přehled, tvorbu, modifikaci a mazání konfigurací akcí
    - FP 3.9 – Požadavky týkající se monitorování hodnot zapsaných na výstupních zařízeních
        - FP 3.9.1 – Modul musí umožňovat monitorování výstupních adres spravovaných zařízení a hodnot na ně zapsané
        - FP 3.9.2 – Modul musí umožňovat monitorování toho, jaký event a jeho akce zapsaly do výše zmíněných adres

- FP 3.10 – Požadavky týkající se spravovaných senzorů
  - FP 3.10.1 – Modul musí poskytovat funkce pro přehled, tvorbu, modifikaci a mazání spravovaných senzorů

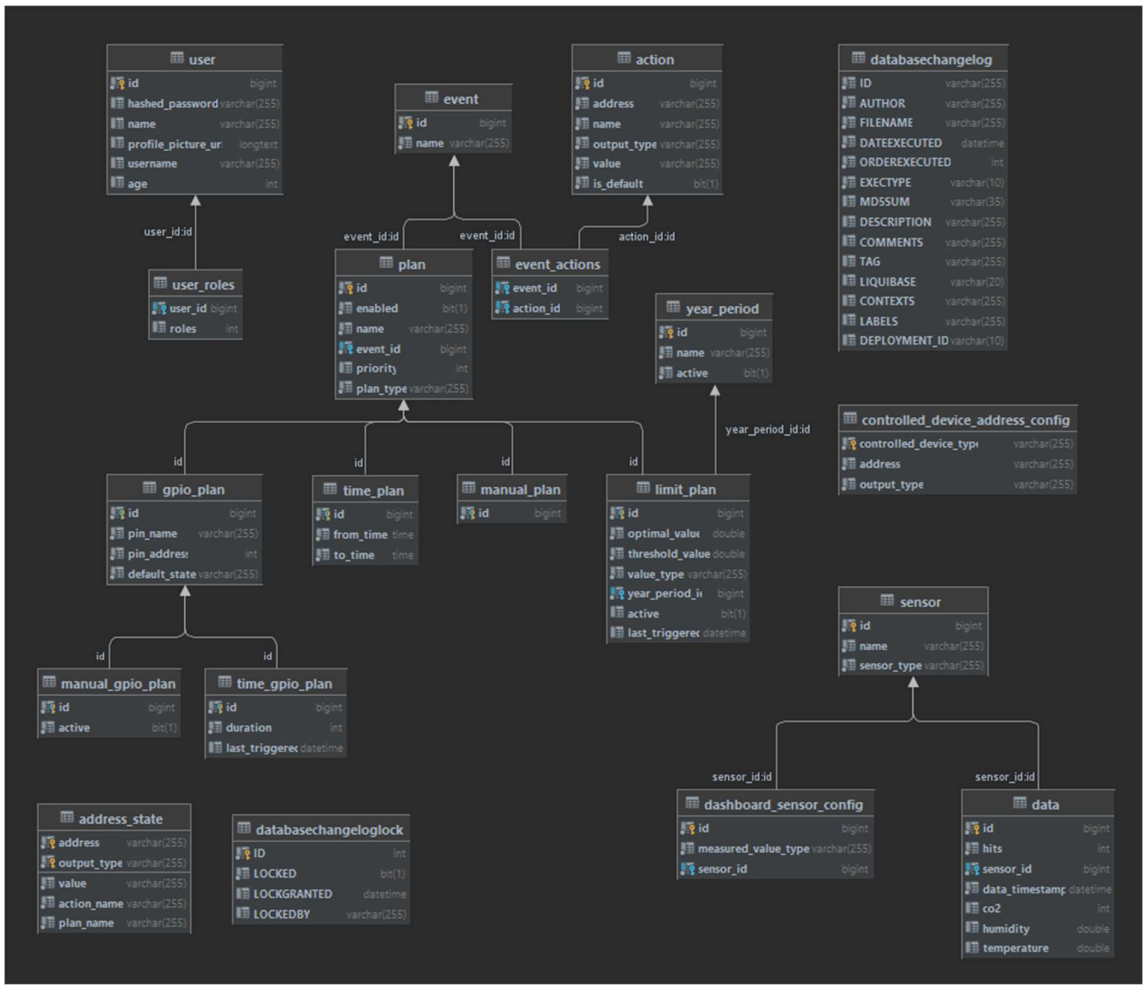
4.2.5 Nefunkční požadavky

- NFP 1 – Požadavky týkající se senzorového modulu
  - NFP 1.1 – Odchylka naměřené teploty vzduchu od skutečné by neměla být více než +/- 0.5 stupně Celsia
  - NFP 1.2 – Odchylka naměřené vlhkosti vzduchu od skutečné by neměla být více než +/- 3% relativní vlhkosti
  - NFP 1.3 – Odchylka naměřeného obsahu oxidu uhličitého v ovzduší od skutečné by neměla být více než +/- 50 ppm
  - NFP 1.4 – Dosah bezdrátové komunikace mezi jednotlivými prvky v běžné domácnosti by měl být minimálně 20 metrů
  - NFP 1.5 – Navrhovaný software by měl být cílený na platformu Arduino
  - NFP 1.6 – Navrhovaný software by měl být implementován v jazyce C++
  - NFP 1.7 – Modul by měl podporovat činnost minimálně tří Arduin s připojenými senzory
- NFP 2 – Požadavky týkající se modulu Data collector
  - NFP 2.1 – Navrhovaný software cílí na platformu Raspberry Pi
  - NFP 2.2 – Navrhovaný software je implementován v jazyce Java
  - NFP 2.3 – Navrhovaný software využívá frameworku Spring boot
  - NFP 2.4 – Využívaný hardware a software je kompatibilní pro komunikaci přes protokol Modbus
  - NFP 2.5 – Navrhovaný software využívá datového uložiště MySQL
- NFP 3 – Požadavky týkající se modulu Data manager
  - NFP 3.1 – Navrhovaný software je cílený na cloudovou platformu Heroku
  - NFP 3.2 – Navrhovaný software je implementován v jazyce Java

- NFP 3.3 – Navrhovaný software využívá frameworku Vaadin
- NFP 3.4 – Navrhovaný software využívá frameworku Spring
- NFP 3.5 – Navrhovaný software je kontejnerizován pomocí Dockeru
- NFP 3.6 – Grafické rozhraní modulu je responzivní
- NFP 3.7 – Navrhovaný software využívá datového uložiště MySQL
- NFP 3.8 – Databázové schéma využívá verzovacího nástroje Liquibase

4.3 Návrh databázového schématu

V této kapitole bude blíže představeno databázové schéma a popsán jeho způsob využití. V řešení je využita databáze MySQL a podle toho byly voleny kompatibilní datové typy. Jak již bylo zmíněno, databáze je využívána jako prostředek ke sdílení dat mezi modulem Data collector a Data manager. Zároveň ho i oba moduly využívají jakožto primární uložiště dat, tedy každý z nich v něm má své tabulky. I z tohoto důvodu se ve schématu nachází větší počet tabulek, přesněji řečeno dvacet. Tabulky by se daly rozdělit do šesti skupin podle typu dat, která obsahují. Ty budou v následujících podkapitolách podrobněji popsány.

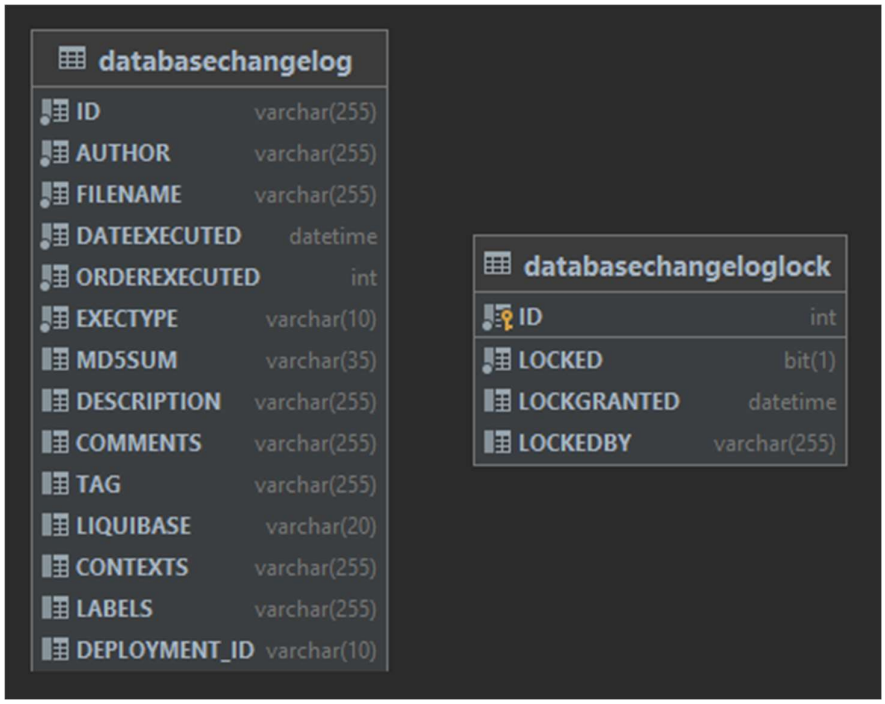


Obrázek 14 Přehledový diagram celého datového modelu

4.3.1 Tabulky nástroje Liquibase

Do první skupiny patří dvě tabulky nástroje Liquibase, nástroje používaného k verzování změn samotného databázového schématu.

První z nich je tabulka „databasechangelog“, tato tabulka obsahuje informace o změnách provedených nad datovým modelem. Záznamy v této tabulce obsahují takové informace jako je jméno autora úprav, jméno changelogu obsahující změny, kdy byly modifikace vykonány, jejich popis a další užitečné informace (viz Obrázek 15). Nástroj podle ní také pozná, jaké změny již byly vykonány a které ještě potřebují doplnit.

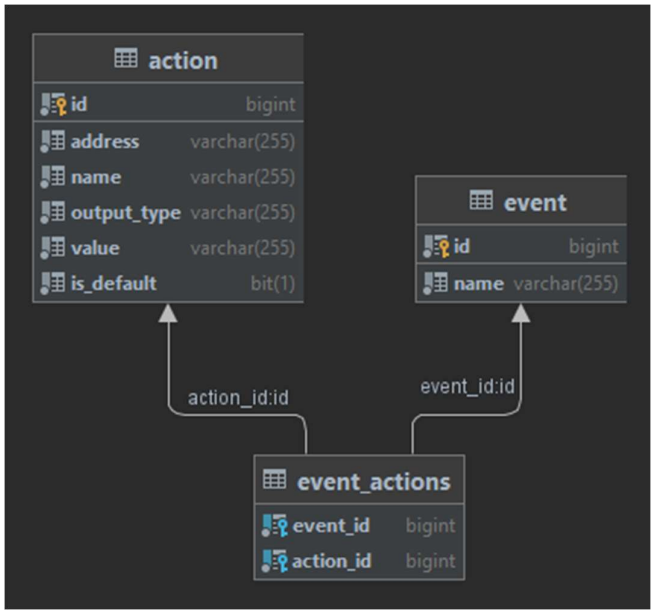


Obrázek 15 Diagram tabulek nástroje Liquibase

Další tabulka s názvem „databasechangeloglock“ slouží pro nástroj jako zámek a má dvě hlavní funkcionality. Při spuštění nástroje se nejprve zkontroluje, zda již není tento zámek alokován, tím je zajištěna exklusivita přístupu a není tedy možné, aby nad databází běžely dvě instance nástroje zároveň. Po úspěšném dokončení běhu nástroje se zámek dealokuje. Pokud nástroj při běhu selže, tak zámek není nikdy uvolněn, a to uživateli značí, že databáze je v nekonzistentním stavu.

4.3.2 Tabulky eventů a akcí

Další skupina tabulek slouží pro uchování eventů a akcí. Jak lze vidět na Obrázek 16, samotné tabulky pro eventy a akce jsou velice jednoduché a pouze reflektují atributy jim odpovídajících entit. Dále se zde nachází vazební tabulka „event_actions", která využívá identifikátorů entit a modeluje vazby mezi nimi. Čtenář by si možná mohl klást otázku, proč není v tabulce akcí přímo reference na event a v jednotlivých záznamech pak není odkaz na event, do kterého patří? Je to z toho důvodu, že vazba mezi entitami je typu M:N. Jeden event může mít tedy několik akcí a zároveň jedna akce může být přidělena několika eventům.



Obrázek 16 Diagram tabulek eventů a akcí

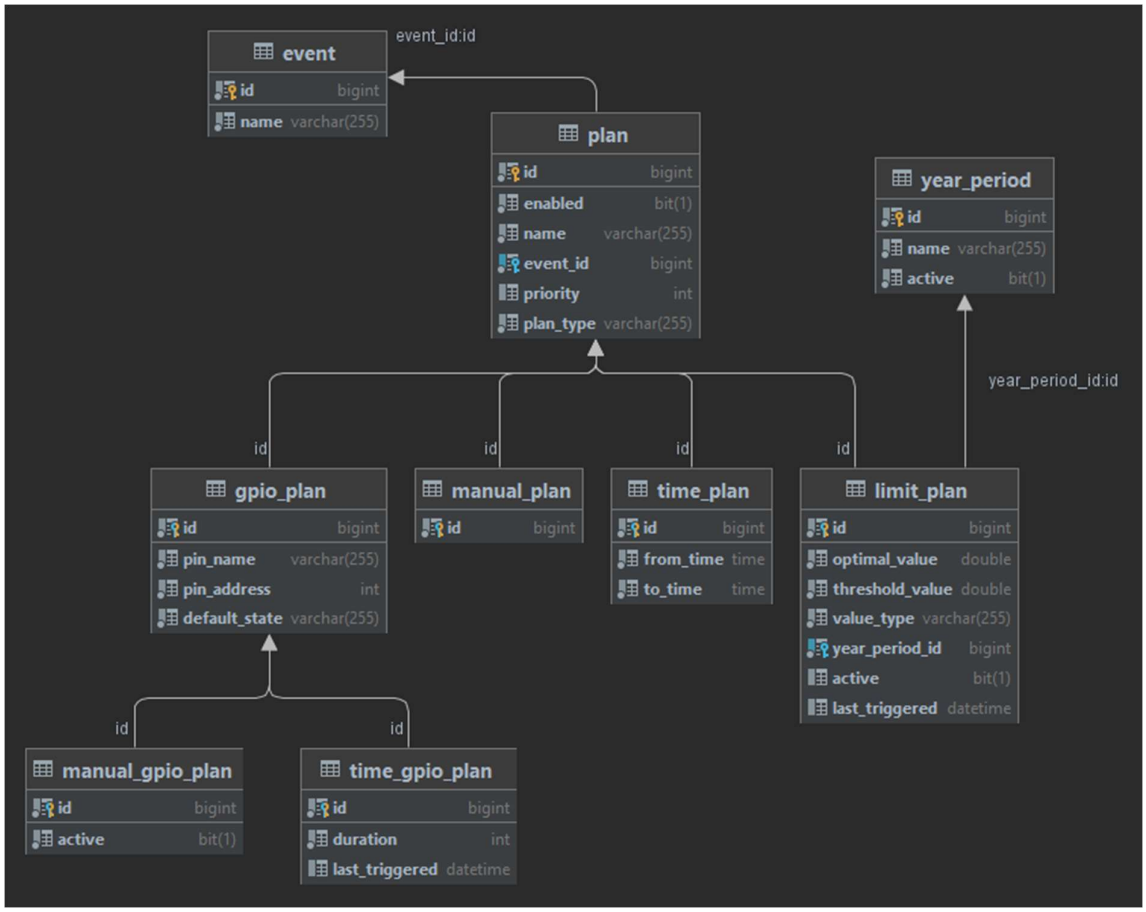
4.3.3 Tabulky plánů

Další skupina tabulek je v porovnání s ostatními největší a z pohledu řešení také nejzajímavější. Jsou v ní zahrnuty definice všech typů plánů a všeho, co k nim patří.

Na části tohoto řešení je nejzajímavější, že jemu odpovídající datové entity využívají abstraktních tříd, obecně řečeno dědičnosti. Je tedy nutné, aby i tato část datového modelu nějakým způsobem reflektovala její využití. Pro zavedení dědičnosti do datových modelů existuje vícero přístupů. První z nich je „Table per class hierarchy", ta funguje tak, že se pro celou hierarchii dědičnosti vytvoří jedna společná tabulka, která obsahuje atributy všech tříd. Další z nich „Table per subclass" funguje tak, že pro každou podtřídu se vytvoří jedna tabulka s pouze jejími atributy a tyto tabulky jsou mezi sebou provázené referencemi. Posledním

z nejpoužívanějších přístupů je „Table per concrete class“, ten vytváří pro každou podtřídu samostatnou tabulku, která obsahuje všechny atributy tedy i ty, které získala od rodičovské třídy.

Jak je patrné z diagramu na Obrázek 17 v datovém modelu se používá přístup „Table per subclass“, jednotlivé tabulky jsou propojeny pomoci id plánu. Tento přístup byl zvolen z toho důvodu, že oproti ostatním přístupům nevytváří v datovém modelu duplikátní sloupce v rámci

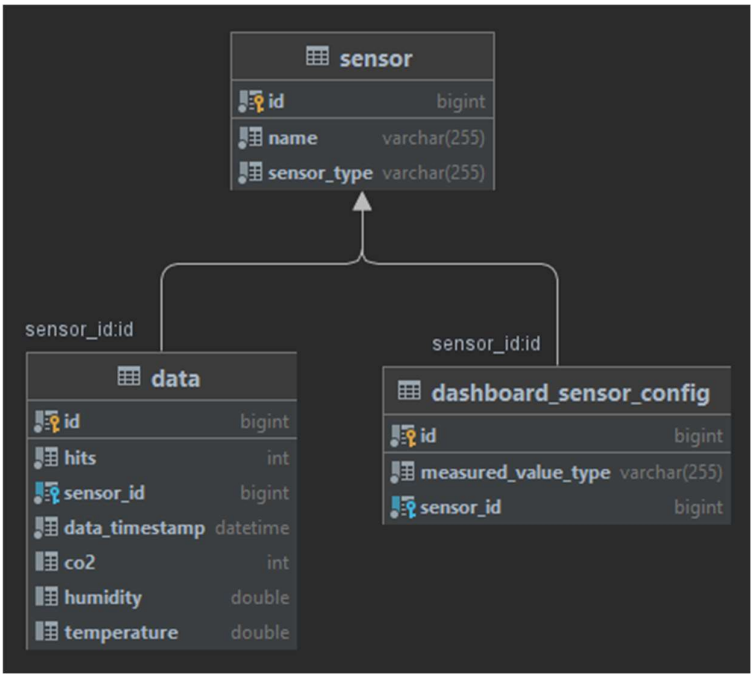


Obrázek 17 Diagram tabulek plánů

hierarchie a zároveň plně dostačuje potřebám obou modulů. Po vysvětlení způsobu přístupu k dědičnosti je orientace v diagramu mnohem jednodušší a po propojení odpovídajících tabulek plánů může být patrné, že obsahují pouze jim odpovídající atributy. V diagramu je také vidět tabulka „year_period”, na kterou má v sobě referenci tabulka „limit_plan”, tato tabulka zastupuje období využívané v limitních plánech a obsahuje informaci o tom, které z období je aktivní. Dále je v diagramu vidět tabulka „event“, která je zde uvedena proto, aby byla patrna reference mezi ní a tabulkou „plan“.

4.3.4 Tabulky senzorů

Následující skupina obsahuje tabulky, které slouží k uchování dat společných se senzory. První z nich, tabulka „senzor“, uchovává informace o senzorech, přesněji jejich název a typ. Další tabulka „data“ uchovává data, která byla naměřena senzorovými moduly a následně zpracována modulem Data collector. Poslední tabulka uchovává konfigurace, které jsou využity při zobrazování senzorů a typu jejich hodnot v modulu Data manager na stránce Dashboard.

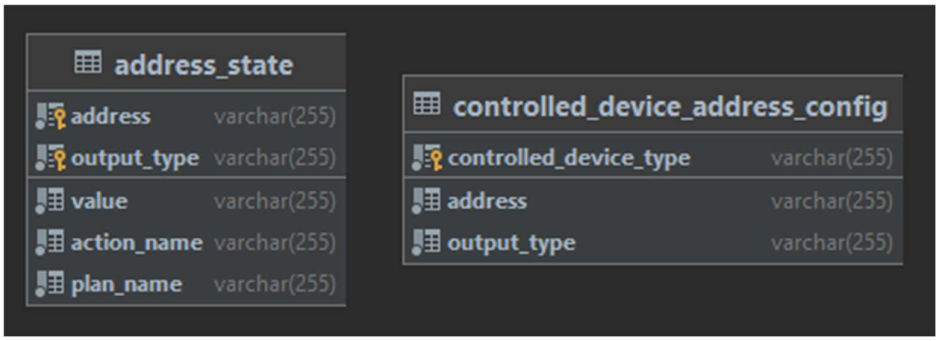


Obrázek 18 Diagram tabulek senzorů

4.3.5 Tabulky adres

Tato skupina tabulek obsahuje data o výstupních adresách, se kterými systém pracuje. První z nich „address_state” uchovává data o spravovaných výstupních adresách, respektive jaké do nich byly zapsané hodnoty a jaké plány a přesněji jejich akce je nastavily. Druhá tabulka je pak spíš konfiguračního charakteru, je využita ke specifikaci výstupů, na kterých jsou připojena

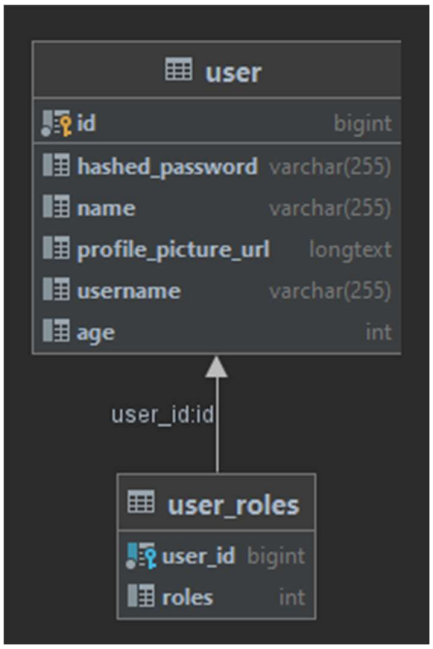
výstupní zařízení. Tato konfigurace je využitá k odvození stavů zařízení, která jsou zobrazována v Dashboardu modulu Data manager.



Obrázek 19 Diagram tabulek adres

4.3.6 Tabulky uživatelských účtů

Poslední skupina obsahuje dvě tabulky, které slouží k uchování dat o uživatelských účtech využívaných v modulu Data manager. První z nich je tabulka „user“, ta reflektuje odpovídající entitu a uchovává její atributy. Druhá tabulka pak obsahuje výčet rolí a jejich vazbu na uživatele.



Obrázek 20 Diagram tabulek uživatelských účtů

4.4 Implementační detaily

V následujících podkapitolách budou blíže popsány implementační detaily jednotlivých modulů. Převážně se budou zaobírat použitými technologiemi, jejich využitím, začleněním do realizace, dále popisem důležitých konceptů, algoritmů, funkcí a v neposlední řadě podrobnějším členěním modulů.

4.4.1 Senzorový modul

Jak bylo obecně naznačeno v kapitole 4.2.1, pojednávající o návrhu aplikace, senzorový modul se dělí na dvě části. První z nich se zaobírá měřením hodnot z okolního prostředí, druhá z nich pak zajišťuje komunikační infrastrukturu a řeší sběr naměřených dat z první části. V této podkapitole bude podrobněji rozebrán software zajišťující činnost v obou částech. Software v obou částech má společné to, že je programovaný v jazyce C++ a je cílen na platformu Arduino, přesněji na model Arduino Nano.

Komunikace mezi oběma částmi je řešena bezdrátově pomocí transceiverů NRF24L01+ a probíhá obousměrně. Druhá část si vyžádá data od zařízení v první částí a ty jí odpoví, zašlou data zpět. Transceivery NRF24L01+ ovšem dokáží pracovat pouze buď v režimu vysílání, nebo naslouchání (podobnost s komunikací typu master-slave). To znamená, že druhá část v režimu vysílání zažádá data od zařízení z první části, to musí být v režimu naslouchání. Ovšem potíže nastanou posléze, druhá část se nyní musí přepnout do režimu naslouchání a zařízení z první části do režimu vysílání, aby mohla data odeslat zpět. Zde nastávají problémy, že zařízení musí být adresována obousměrně, aby komunikace mohla být vůbec možná. Další problém nastává při otázce časování, jak dlouho by měla druhá část naslouchat a čekat na odpověď, než se opět přepne do režimu vysílání. S tímto druhem komunikace je spojeno vícero dalších problémů, které se promítají do obtížnosti implementace.

Naštěstí transceivery NRF24L01+ umožňují zaslání zprávy, u které je očekávána odpověď v podobě 32 B paketu potvrzujícího obdržení zprávy, takzvaného „acknowledge packetu“ (zkráceně ackPacket). Tohoto konceptu se v implementaci využívá tak, že software využívaný v první části vyplní ackPacket naměřenými hodnotami. Komunikace pak vypadá následovně: software druhé části v režimu vysílání zažádá zařízení z první části (ta jsou v režimu naslouchání) o naměřená data. Jako reakce na obdrženou zprávu zašle zařízení z první části ackPacket, který obsahuje naměřené hodnoty. Části si tedy pro potřeby předávaní dat nepotřebují měnit role a odpadají tím problémy popsané výše. Jedno z omezení tohoto typu

komunikace je maximální velikost ackPacketu 32 bytů. Ta ale pro potřeby předání naměřených dat plně dostačuje. Formát dat vyplňovaných do ackPacketu je vidět na Obrázek 21, velikost struktury je 24 bytů.

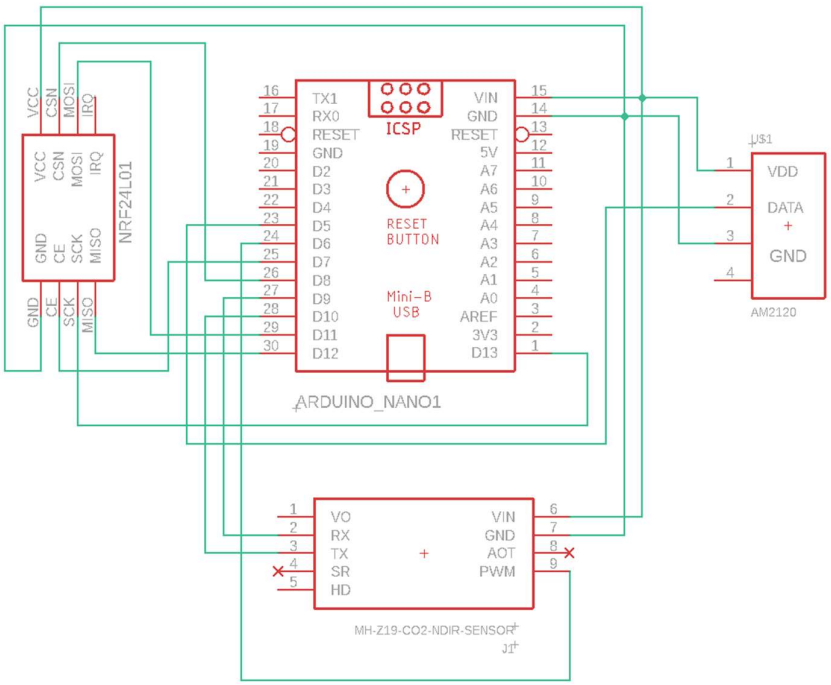
```
struct {
  int id;
  float teplota;
  float vlhkost;
  int co2ppm;
  int co2ppm2;
  int teplota2;
} dataStruct;
```

Obrázek 21 Formát datové struktury pro naměřená data

Software v první části, nazývaný jako Receiver (dle jeho role v bezdrátové komunikaci), je velice jednoduchý. Po spuštění se zahájí komunikace se senzory a na přidělené adrese začne naslouchat na bezdrátové síti. Poté v nekonečné smyčce blokovaný čeká, dokud neobdrží zprávu. Při obdržení zprávy vyčte data ze senzorů, naplní jimi datovou strukturu a tu odešle jakožto ackPacket. Následně datovou strukturu promaže a opět blokovaný čeká, dokud neobdrží zprávu.

Software v druhé části, nazývaný jako Transmiter, je podobně jednoduchý. Po spuštění nakonfiguruje prvek pro bezdrátovou komunikaci. Poté v nekonečné smyčce postupně zasílá zprávy na adresy zařízení, které mu byly nakonfigurovány. Po obdržení dat z ackPacketu zaslaných z Recieveru, tato data naformátuje a odešle je přes sériovou linku na připojený modul Data collector.

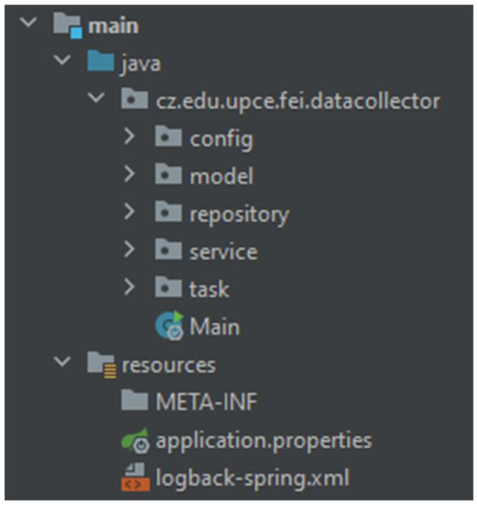
Na Obrázek 22 lze vidět schéma zobrazující, jakým způsobem k Arduino Nano připojit senzor měřící teplotu a vlhkost AM2120, senzor měřící obsah oxidu uhličitého MHZ-19B a transceiver NRF24L01+ používaný k bezdrátové komunikaci.



Obrázek 22 Schéma zapojení prvků k Arduinu Nano

4.4.2 Data collector

Software tohoto modulu je napsán v jazyce Java s využitím frameworku Spring Boot a nástroje Gradle. Framework zajišťuje správu a konfiguraci instancí objektů, zjednodušenou konfiguraci programu pomocí properties souboru, jednoduchou práci s více vlákny a spoustu dalších funkcionalit. Cílený je na platformu Raspberry Pi, přesněji pak na Raspberry Pi 3



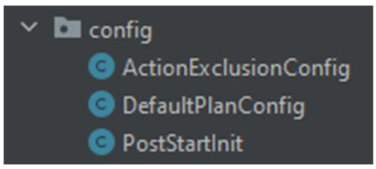
Obrázek 23 Přehled top level struktury balíčků z modulu Data collector

model B. Soubory aplikace jsou před distribucí seskupeny do JAR balíčku, ve kterém je pak aplikace do cílového systému distribuována.

Z Obrázek 23 je patrné, jakým způsobem je aplikace členěna. Složka „java“ obsahuje zdrojové kódy aplikace rozčleněné dle standardních konvencí jazyka Java do balíčků. Ve složce „resources“ jsou pak další dva soubory, z nich soubor „application.properties“ obsahuje uživatelské konfigurace aplikace a soubor „logback-spring.xml“ obsahuje konfigurace logovacího nástroje Logback.

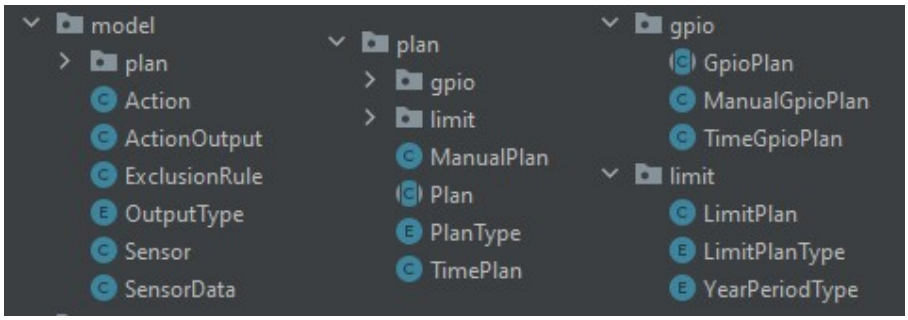
Aplikace je vnitřně členěna na tyto balíčky:

- cz.edu.upce.fei.datacollector – Hlavní balíček obsahující spouštěcí bod aplikace, třídu Main a všechny ostatní balíčky.



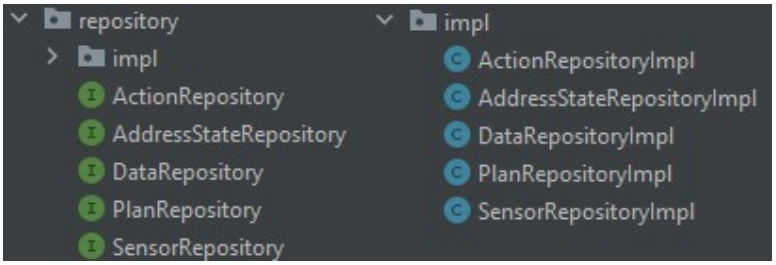
Obrázek 24 Přehled balíčku config z modulu Data collector

- config – Balíček obsahuje třídy, které reflektují uživatelské konfigurace ze souboru „application.properties“ a třídu „PostStartInit“, která provádí inicializační operace při startu aplikace.



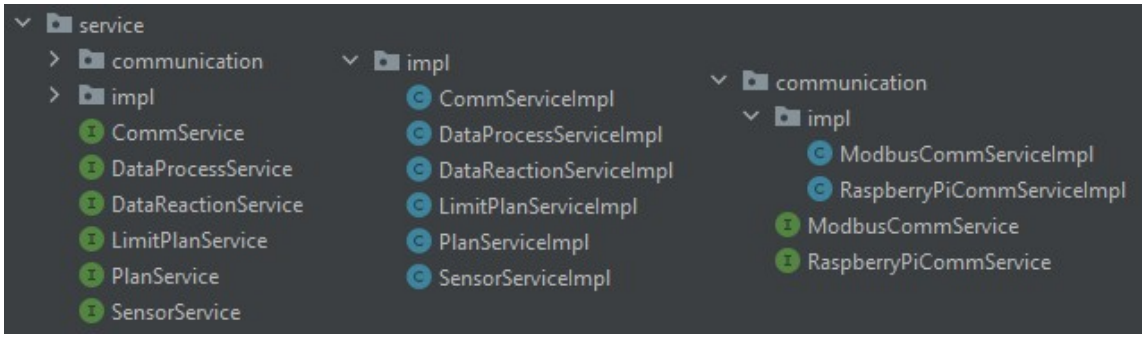
Obrázek 25 Přehled balíčku model z modulu Data collector

- model – Balíček obsahuje definice datových modelů aplikace.



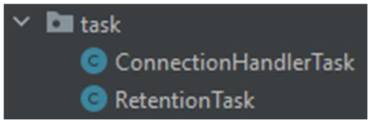
Obrázek 27 Přehled balíčku repository z modulu Data collector

- repository – Balíček obsahuje rozhraní repositářové vrstvy a jejich implementací za pomocí JDBC.



Obrázek 26 Přehled balíčku service z modulu Data collector

- service – Balíček obsahuje rozhraní a implementaci všech služeb vyskytujících se v aplikaci.



Obrázek 28 Přehled balíčku task modulu Data collector

- Task – Balíček obsahuje úlohy, které se periodicky opakují a neobsahují žádnou pokročilou business logiku.

Aplikace vykonává několik typů funkcí, které mohou probíhat paralelně. Přesněji řečeno jde o dynamické připojování/odpojování senzorových modulů, zpracovávání dat obdržených ze senzorů, reagování na aktivní plány, konfigurování GPIO listenerů a zajišťování promazávání senzorových dat. Tyto funkce na sebe mají navázané další části, které budou popsány níže. Z důvodu potřeby paralelismu pracuje aplikace ve vícevláknovém režimu.

První vlákno obsahuje funkci, sloužící k dynamickému připojování a odpojování senzorových modulů. Ta periodicky kontroluje, zda nebyl připojen nový modul senzorů do

USB portu. Pokud objeví nově připojený modul, je zaregistrován do seznamu spravovaných modulů a je na něj navázán listener, který přijaté zprávy zasílá do další části aplikace. Pokud je zjištěno, že byl senzorový modul odpojen, tak je odstraněn jemu přidělený listener a modul je odstraněn ze seznamu spravovaných modulů.

Další vlákno obsahuje službu, která obsluhuje zpracovávání dat obdržených ze senzorových modulů. Tato služba získává zprávy z komunikace navázané funkcí popsané v předchozím odstavci, respektive je provázána s kódem listenerů, které zmíněná funkce registruje. Získané zprávy jsou průběžně ukládány do speciálního bufferu, který služba obsahuje. V intervalech nastavených uživatelem se zprávy z toho bufferu zpracují a buffer se vyprázdní. Zpracování nejprve začíná validací obdržených zpráv a případným filtrováním zpráv v neplatném formátu. Zprávy, které jsou v korektním formátu, jsou následně rozparsovány a jsou uloženy do odpovídajícího datového modelu. V posledním kroku proběhne agregace dat, tedy jejich sdružení dle poskytnutého id senzoru. Výstupem jsou zprůměrovaná data s časovým znaménkem a informací o tom, z kolika zpráv byl průměr vytvořen. Data jsou následně uložena do databáze.

Následující vlákno obsluhuje nejobsáhlejší a funkcí nejdůležitější skupinu služeb. Tato skupina řeší vyhodnocování aktivních plánů, z nich odvodí výstupní hodnoty pro externí zařízení a ty následně propaguje jak do zařízení, tak do databáze. Hlavní službou, která v podstatě celý proces řídí, je služba „DataReactionService“. Přesnější postup je následující: hlavní služba si nejprve načte všechny výstupy, které obsluhuje, z nich si vytvoří strukturu, která bude následně vyplňována hodnotami, které budou na adresy výstupů zapsány.

V dalším kroku je provolána služba, která postupně vyhodnocuje, jaké plány jsou aktivní. Vyhodnocování probíhá tak, že u manuálních plánů je vyhodnocen příznak, který značí, že je plán aktivní. Následuje vyhodnocení časových plánů, u kterých se vyhodnotí, zda je aktuální čas v rozsahu nakonfigurovaném v plánu. U limitních plánů je chování trošku složitější, uvažujme příklad plánu reagujícího na příliš vysokou teplotu. Tento plán má nastaven jako hranici 25 °C, při které bude aktivován, dále má tento plán nastavenou optimální teplotu 23 °C, dokud nebude této teploty dosaženo, bude plán stále aktivní. V praxi tedy systém zapne chlazení při hraniční teplotě a chladí do té doby, dokud není dosaženo optimální teploty. Pokud by bylo využito pouze hraniční teploty, a venku panovaly tropické teploty, tak by systém neustále přecházel mezi stavy, kdy by chladil (teplota 25 °C a více) a stavem, kdy by bylo chlazení vypnuto (teplota 24,9 °C). Tento nežádoucí stav je vyřešen výše zmíněným přístupem. Na tomto příkladě bylo vysvětleno, jak funguje vyhodnocování limitních plánů. Téměř totožné

chování lze pozorovat i ostatních druhů limitních plánů. Poslední vyhodnocované plány jsou GPIO plány, manuální podtyp GPIO plánu se chová úplně stejně jako manuální plány, je tedy vyhodnocován podle nastaveného příznaku. Časový podtyp GPIO plánu se chová tak, že je zaznamenán čas sepnutí mechanického prvku a při vyhodnocování je kontrolováno, zda je aktuální čas v rozmezí od času sepnutí, po čas sepnutí plus čas definovaný v plánu. Plány vyhodnocené jakožto aktivní jsou zaslány zpět do hlavní služby.

Aktivní plány jsou v hlavní službě seřazeny podle priority. Následně jsou jednotlivé plány postupně iterovány a jejich akce jsou postupně vyplňovány do struktury evidující spravované výstupy. Ty jsou vyplněny pouze pokud jim odpovídající výstup nemá přiřazenou hodnotu, nebo pokud vyhovuje pravidlům vzájemného vylučování akcí. Pokud po vyčerpání všech aktivních plánů jsou ještě nějaké výstupy nevyplněné, tak se modul snaží jejich hodnotu odvodit v tomto pořadí z defaultních plánů, defaultních akcí, jejich defaultní hodnoty, a pokud není ani jedno dostupné, tak je nastavena hodnota 0.

V předchozím odstavci byla zmíněna pravidla vzájemného vylučování akcí. Tato pravidla jsou uživatelem konfigurována a řeší to, že se v systému mohou vyskytnout akce, u kterých chce mít uživatel vždy vyšší prioritu než u jiných, nehledě na to, jakou má prioritu plán, ve kterém figurují. Jedno takto definované pravidlo se vždy týká pouze jedné adresy a zajišťuje, aby určité hodnoty nebyly nikdy přepsány jinými (viz **Error! Reference source not found.**).

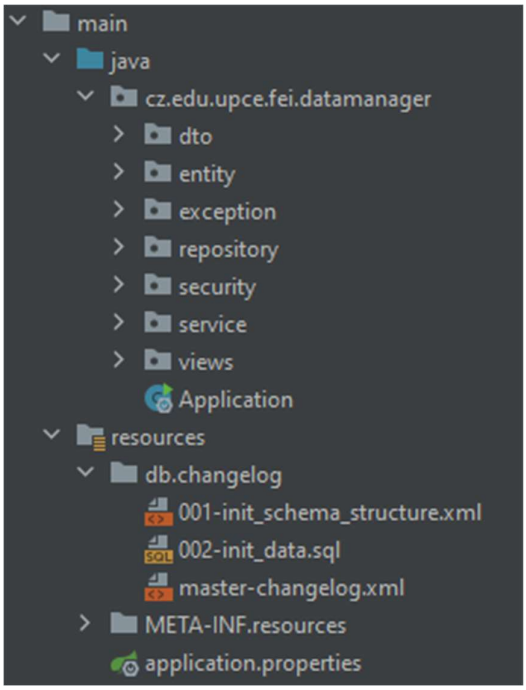
V následujícím kroku jsou získané hodnoty zapsány podle jejich typu, přes specializované rozhraní přímo do zařízení. Tedy buď přes službu implementující komunikační rozhraní GPIO Raspberry Pi nebo rozhraní Modbus RTU. Rozhraní GPIO je implementováno za pomocí knihovny Pi4J a rozhraní Modbus je implementováno pomocí knihovny easymodbus. Po úspěšném zapsání do odpovídajících rozhraní jsou hodnoty ukládány i do databáze.

Další vlákno v aplikaci slouží k registrování listenerů na rozhraní GPIO Raspberry Pi, respektive na jeho vstupy, které jsou nakonfigurovány v rámci GPIO plánů. Tyto listenery pak na těchto vstupech naslouchají, a pokud se na nich vyskytne požadovaný vstup, aktivují odpovídající plán. Toto vlákno se spouští periodicky dle konfigurace uživatele.

Poslední vlákno obsluhuje promazávání senzorových dat. To periodicky, dle konfigurace uživatele, promazává senzorová data, která už jsou podle uživatele příliš stará a z jeho pohledu už nezajímavá.

4.4.3 Data manager

Software modulu Data manager je napsán pomocí programovacího jazyku Java s využitím frameworku Vaadin, frameworku Spring a nástroje Maven. Framework Vaadin poskytuje spoustu výhod při vytváření webových aplikací, a společně s frameworkem Spring se umožňuje soustředit přímo na implementaci funkčních požadavků aplikace bez potřeby složitých konfigurací. Software je cílen pro cloudové využití, přesněji na platformu Heroku, z tohoto důvodu se ve zdrojových kódech nachází Dockerfile, který dokáže aplikaci kontejnerizovat.



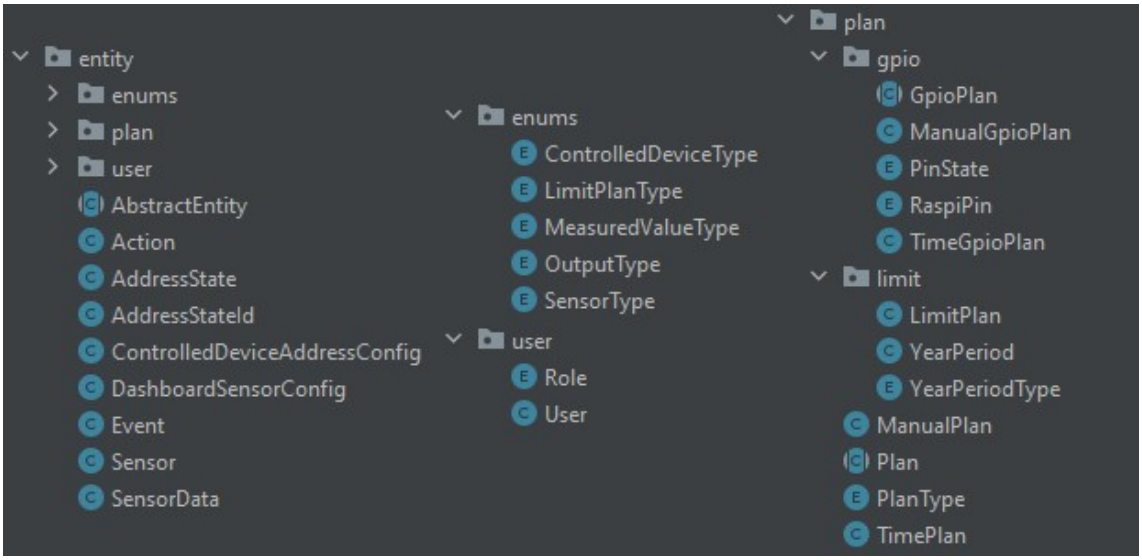
Obrázek 29 Přehled top level struktury balíčků z modulu Data manager

Z Obrázek 29 je patrné, jakým způsobem je aplikace členěna. Lze z něj také poznat, že aplikace ctí třívrstvou architekturu a implementuje komponenty jako jsou repository, service a views. Složka „java“ obsahuje zdrojové kódy aplikace rozčleněné dle standardních konvencí jazyka Java do balíčků. Ve složce „resources“ je soubor „application.properties“ obsahující uživatelské konfigurace aplikace a dále složka „db.changelog“, ve které lze nalézt changelogy. Ty využívá nástroj Liquibase k verzování změn nad databázovým schématem.

Aplikace je vnitřně členěna na tyto balíčky:

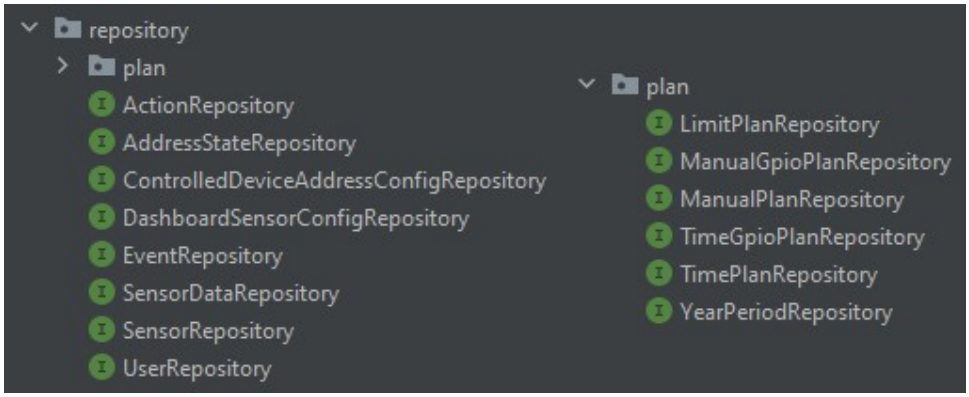
- cz.edu.upce.fei.datamanager – Hlavní balíček obsahující spouštěcí bod aplikace, třídu Application a všechny ostatní balíčky.

- dto – Balíček obsahuje třídy sloužící k přenosu dat mezi vrstvami.



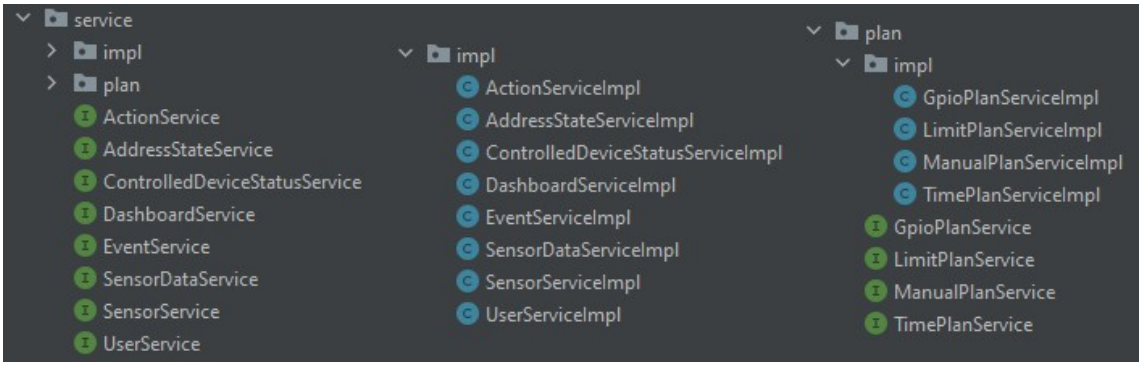
Obrázek 30 Přehled balíčku entity z modulu Data manager

- entity – Balíček obsahuje definice položek datové vrstvy, konfigurovaných jako Hibernate entity.
- exception – Balíček obsahuje business výjimky používané k řízení toku dat.



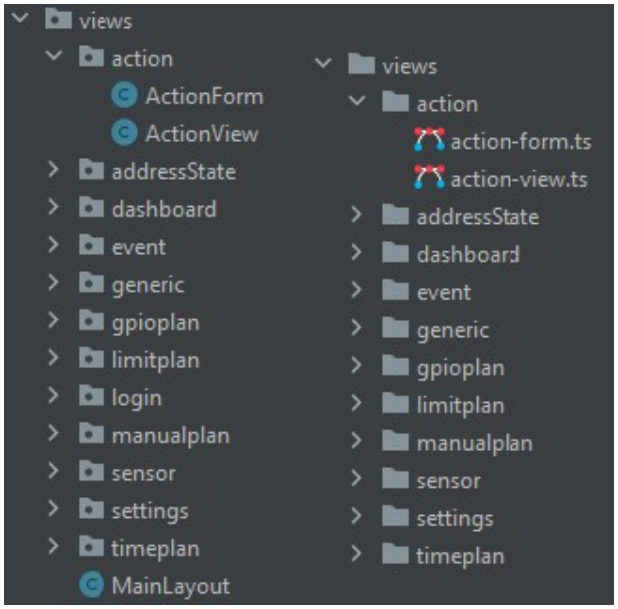
Obrázek 31 Přehled balíčku repository z modulu Data manager

- repository – Balíček obsahuje repositařové rozhraní rozšiřující JPA standard.



Obrázek 33 Přehled balíčku service z modulu Data manager

- service – Balíček obsahující rozhraní a implementace služeb modulu.



Obrázek 32 Přehled balíčku views z modulu Data manager

- views – Vlevo jsou Java balíčky obsahující logiku jednotlivých zobrazení, vpravo jsou pak jim odpovídající balíčky obsahující javascriptový kód definující vzhled jednotlivých zobrazení

Aplikace jako taková není implementačně nijak náročná, jedná se o klasickou webovou aplikaci postavenou nad třívrstvou architekturou. První vrstvou je repositářová vrstva obsahující rozhraní dle JPA standardu, ta zajišťuje přístup k datům a další operace nad databází. Další vrstva je servisní vrstva, obsahující služby zajišťující primárně pouze přenos dat mezi vrstvami a validace dat. Následuje vrstva zobrazovací, která se dělí na dvě části (viz Obrázek 32). Dělí se na Java třídy obsahující logiku určující, jak zobrazení fungují, a na soubory obsahující javascriptový kód definující, jak budou zobrazení vypadat. Soubory

s javascriptovým kódem byly vygenerovány pomocí nástroje Vaadin builder, to je vizuální nástroj pomáhající vytvářet zobrazení pomocí techniky drag and drop. Jedná se tak o značné zjednodušení tvorby grafických rozhraní. Na modul je navázán nástroj Liquibase, to znamená, že při spuštění modulu Data manager se kontroluje verze databázového schématu a v případě nalezení staré verze se aktualizuje.

4.5 Konfigurace

V této kapitole bude popsáno, jaké konfigurace jsou v jednotlivých modulech možné a jaký mají vliv na jejich funkci.

4.5.1 Senzorový modul

Senzorový modul obsahuje dva samostatné programy psané v jazyku C++. Oba programy jsou kompilované a jejich konfiguraci je nutné provádět přímo ve zdrojovém kódu. Pro práci se zdrojovými kódy je doporučeno využít Arduino IDE, které je vytvořeno přímo pro práci s programy cílenými na platformu Arduino.

První z programů nazvaný Transmiter se nachází v souboru s názvem „TxAckPayload.ino“. V tomto programu lze konfigurovat počet a adresy zařízení, se kterými bude program navazovat komunikaci. Na prvním řádku na Obrázek 34 se nachází proměnná, která určuje aktuální počet zařízení. Na následujících řádcích se nachází dvourozměrné pole znaků, obsahující definice adres jednotlivých zařízení (počínaje adresou „00001“). Program je aktuálně nakonfigurován na maximální možnou kapacitu obsluhovaných zařízení. Doporučuje se tedy měnit pouze adresy a ani to není ve většině případů nutné.

```
10  const byte numSlaves = 5;
11  const byte slaveAddress[numSlaves][5] = {
12                              {'0','0','0','0','1'},
13                              {'0','0','0','0','2'},
14                              {'0','0','0','0','3'},
15                              {'0','0','0','0','4'},
16                              {'0','0','0','0','5'}
17                          };
18
```

Obrázek 34 Příklad způsobu konfigurace Transmiteru senzorového modulu

Druhý program se nazývá Receiver a nachází se v souboru „RxAckPayload.ino“. Jde o program, jehož instancí se většinou v systému vyskytuje několik, obsluhuje totiž samotné senzory, a to reflektuje i jeho konfigurace. Na Obrázek 35 lze spatřit první konfigurovanou hodnotu s názvem „sensorId“, ta vyjadřuje, pod jakým identifikátorem bude zařízení vystupovat v rámci systému. Druhou hodnotou je pak adresa, pod kterou bude zařízení vystupovat a naslouchat v rámci bezdrátové komunikace.

```
28
29  // !! Config start - CHANGE ONLY HERE !!
30  #define sensorId 1
31  const byte address[6] = {'0','0','0','0','1'};   // Address of this node
32  // !! CONFIG END !!
33
```

Obrázek 35 Příklad způsobu konfigurace Receiveru senzorového modulu

4.5.2 Data collector

Modul Data collector využívá Java framework Spring boot. Ten umožňuje velmi jednoduchý a rychlý způsob konfigurace pomocí externích konfiguračních souborů, do kterých stačí pouze přidat záznamy a aplikace si pak sama dokáže nakonfigurovat vše potřebné, bez nutnosti modifikace zdrojového kódu. V případě stávajícího řešení se využívá konfigurační soubor s názvem „application.properties“. Všechny způsoby konfigurace budou popsány v následujících odstavcích.

Část konfiguračního souboru na Obrázek 36 obsahuje následující záznamy: „logging.level.root“ – Slouží k nastavení globální úrovně v celé aplikaci (včetně knihoven). „logging.level.root.cz.edu.upce.fei.datacollector.*“ – Slouží k nastavení úrovně logování souborů patřících aplikaci. Následující záznamy slouží ke konfiguraci datového zdroje.

```
1  logging.level.root=WARN
2  logging.level.cz.edu.upce.fei.datacollector.*=INFO
3
4  # DB properties
5  spring.datasource.url=jdbc:mysql://localhost:3306/thesis
6  spring.datasource.username=root
7  spring.datasource.password=password
8  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

Obrázek 36 Konfigurace logování a připojení do DB v modulu Data collector

V následujícím pořadí jde o URL databáze, do které se bude aplikace připojovat, dále jsou zde záznamy obsahující přihlašovací údaje a typ databázového ovladače, který bude využit při připojení.

V další části konfiguračního souboru na Obrázek 37 jsou záznamy, jejichž hodnoty jsou zapsány ve formátu cron záznamů. Tyto záznamy definují periody vykonávání úloh, kterým přísluší. Záznamy jsou přidělené následujícím úlohám:

- „connectionHandlingTask“ – Úloha zajišťující dynamické připojování/odpojování senzorových modulů.
- „dataProcessingTask“ – Úloha zpracovávající data obdržená ze senzorů.
- „planReactionPeriod“ – Úloha zajišťující vyhodnocování a reakce na aktivní plány.
- „refreshGpioListenersPeriod“ – Úloha konfigurující GPIO listenery.
- „dataRetentionInvokePeriod“ – Úloha zajišťující mazání senzorových dat.

```
11  # Schedulers timing
12      # cron for search for new sensor connections
13  connectionHandlingTask=55 */1 * * * ?
14      # cron for received data processing schedule
15  dataProcessingTask=0 */1 * * * ?
16      # cron for plan evaluation and address states change
17  planReactionPeriod=30 */1 * * * ?
18      # cron for updating GPIO listeners
19  refreshGpioListenersPeriod=20 */5 * * * ?
20      # cron for sensor data retention startup
21  dataRetentionInvokePeriod=0 0 * * * ?
```

Obrázek 37 Konfigurace cron záznamů v modulu Data collector

Následující část konfiguračního souboru na Obrázek 38 obsahuje tyto záznamy: První z nich definuje počet minut, po který jsou senzorová data považována za aktuální. To je využito pro potřeby vyhodnocování limitních plánů. Následuje název USB portu, na kterém je připojen

```
24  # Other config values
25  # default value is 5 minutes
26  limitPlan.dataFetching.maxAgeInMinutes=15
27  modbus.usbPort=COM3
28  # default values is 1
29  modbus.targetId=1
30  # sensor data retention period (in days)
31  sensordata.retention.days=30
```

Obrázek 38 Ostatní konfigurace v modulu Data collector

adaptér USB – RTU, který je využíván pro komunikaci přes protokol Modbus. Dalším

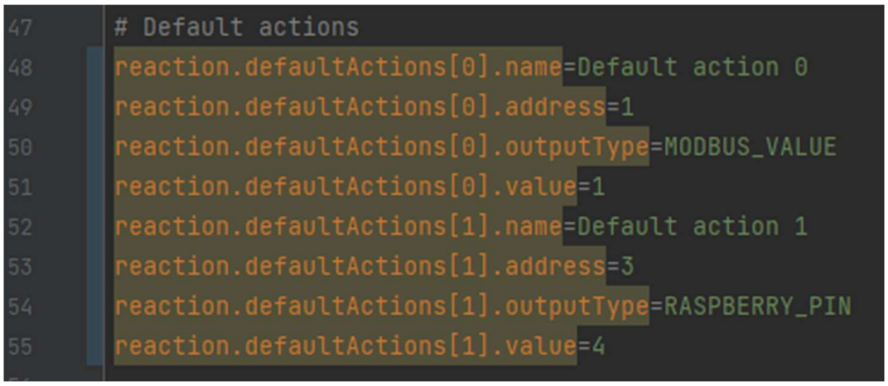
záznamem je identifikátor, pod kterým aplikace vystupuje v rámci komunikace přes protokol Modbus. Poslední záznam v této části obsahuje počet dnů, po který jsou senzorová data uchovávaná v databázi.

Část konfiguračního souboru na Obrázek 39 umožňuje definovat limitní plány přímo v modulu Data collector. Princip je v podstatě úplně stejný jako v případě konfigurace přes modul Data manager popsaný u Obrázek 43. Limitní plán nakonfigurovaný na Obrázek 39 funguje tak, že se aktivuje, pokud bude teplota prostředí menší než 19 °C a bude aktivní tak dlouho, dokud hodnota teploty prostředí nebude 20 °C, nebo vyšší. Pokud bude plán aktivován, jeho snahou bude, aby bylo přes rozhraní Modbus zapsáno na adresu 1 hodnota 1. Způsob, jakým se budou vyhodnocovat hraniční hodnoty, je nastaven v záznamu končícím řetězcem „valueType“. Povolené hodnoty jsou „TEMPERATURE_LOW“ (reaguje na nastavenou spodní hranici teploty), „TEMPERATURE_HIGH“ (reaguje na nastavenou horní hranici teploty) a „CO2“ (reaguje na nastavenou horní hranici oxidu uhličitého). Rozhraní, do kterého se bude zapisovat je nastaveno v záznamu končícím řetězcem „outputType“. Povolené hodnoty jsou „MODBUS_VALUE“ (umožňuje zápis celočíselné hodnoty do rozhraní Modbus), „MODBUS_BOOLEAN“ (umožňuje zápis boolean hodnoty do rozhraní Modbus) a „RASPBERRY_PIN“ (umožňuje zápis celočíselné hodnoty do rozhraní GPIO Raspberry Pi).

```
34  # Default limit plans
35  reaction.defaultLimitPlans[0].name=Default limit plan - Low temperature
36  reaction.defaultLimitPlans[0].enabled=true
37  reaction.defaultLimitPlans[0].planType=LIMIT_PLAN
38  reaction.defaultLimitPlans[0].valueType=TEMPERATURE_LOW
39  reaction.defaultLimitPlans[0].optimalValue=20.0
40  reaction.defaultLimitPlans[0].thresholdValue=19.0
41  reaction.defaultLimitPlans[0].actionList[0].name=Default action 0
42  reaction.defaultLimitPlans[0].actionList[0].address=1
43  reaction.defaultLimitPlans[0].actionList[0].value=1
44  reaction.defaultLimitPlans[0].actionList[0].outputType=MODBUS_VALUE
```

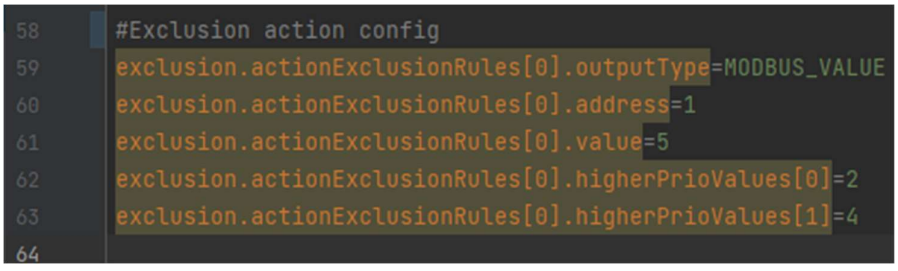
Obrázek 39 Konfigurace defaultních limitních plánů v modulu Data collector

V části konfiguračního souboru na Obrázek 40 se nachází definice defaultních akcí, tedy akcí, které budou vykonány, pokud modul nedokázal odvodit hodnoty z jiných zdrojů. Na obrázku jsou vidět definice dvou defaultních akcí. První z nich zajišťuje zapsání hodnoty 1 na adresu 1 do rozhraní Modbus a druhá z nich zajišťuje zapsání hodnoty 3 na adresu 4 do rozhraní GPIO zařízení Raspberry Pi. Záznam končící řetězcem „outputType“ má stejnou funkci jako v předchozí konfiguraci, má tedy i stejné povolené hodnoty „MODBUS_VALUE“, „MODBUS_BOOLEAN“ a „RASPBERRY_PIN“.



Obrázek 40 Konfigurace defaultních akcí v modulu Data collector

Poslední část konfiguračního souboru na Obrázek 41 zobrazuje nastavení pravidel pro vzájemné vylučování akcí. Konfigurace na obrázku zajišťuje to, že při zapisování přes rozhraní Modbus na adresu 1 mají před hodnotou 5 vždy přednost hodnoty 2 a 4. Tato přednost je brána v potaz, i pokud by hodnota 5 byla v plánu s vyšší prioritou. Záznam končící řetězcem „outputType“ má stejnou funkci jako v předchozích konfiguracích, má tedy i stejné povolené hodnoty „MODBUS_VALUE“, „MODBUS_BOOLEAN“ a „RASPBERRY_PIN“.



Obrázek 41 Konfigurace vzájemně výlučných akcí v modulu Data collector

4.5.3 Data manager

Modul Data manager je webová aplikace napsaná v programovacím jazyce Java, za pomocí frameworku Vaadin a Spring. Konfigurace popisované v této kapitole by se daly rozdělit na dvě části: (i) část, která zajišťuje konfiguraci samotného modulu, (ii) část, která zajišťuje konfiguraci systému jako celku, ta je nastavována přímo ve webové aplikaci.

Stejně jako v případě modulu Data collector, i tento modul využívá externích konfiguračních souborů, které poskytuje framework Spring. Těchto souborů, přesněji řečeno souborů s názvem „application.properties“ je využito ke konfiguraci samotného modulu. Celý konfigurační soubor je vidět na Obrázek 42. První záznam definuje port, na kterém bude aplikační server po spuštění dostupný. Další záznam využívá nástroj Liquibase a obsahuje cestu k souboru, ve kterém se nachází seznam changelogů. Následující záznamy slouží ke konfiguraci připojení do databáze. V tomto pořadí jde o URL datového zdroje, ke kterému se bude aplikace připojovat, poté jsou zde záznamy obsahující přihlašovací údaje, a nakonec typ databázového ovladače, který je využíván při připojení. Poslední dva záznamy jsou využívány pro konfiguraci stránky Dashboard. První z nich určuje maximální povolené stáří senzorových dat, po které budou data brána jako aktuální a budou zobrazena na Dashboardu. Druhý z nich je záznam s hodnotou ve formátu cron záznamu, který slouží k nastavení frekvence obnovování dat na Dashboardu.

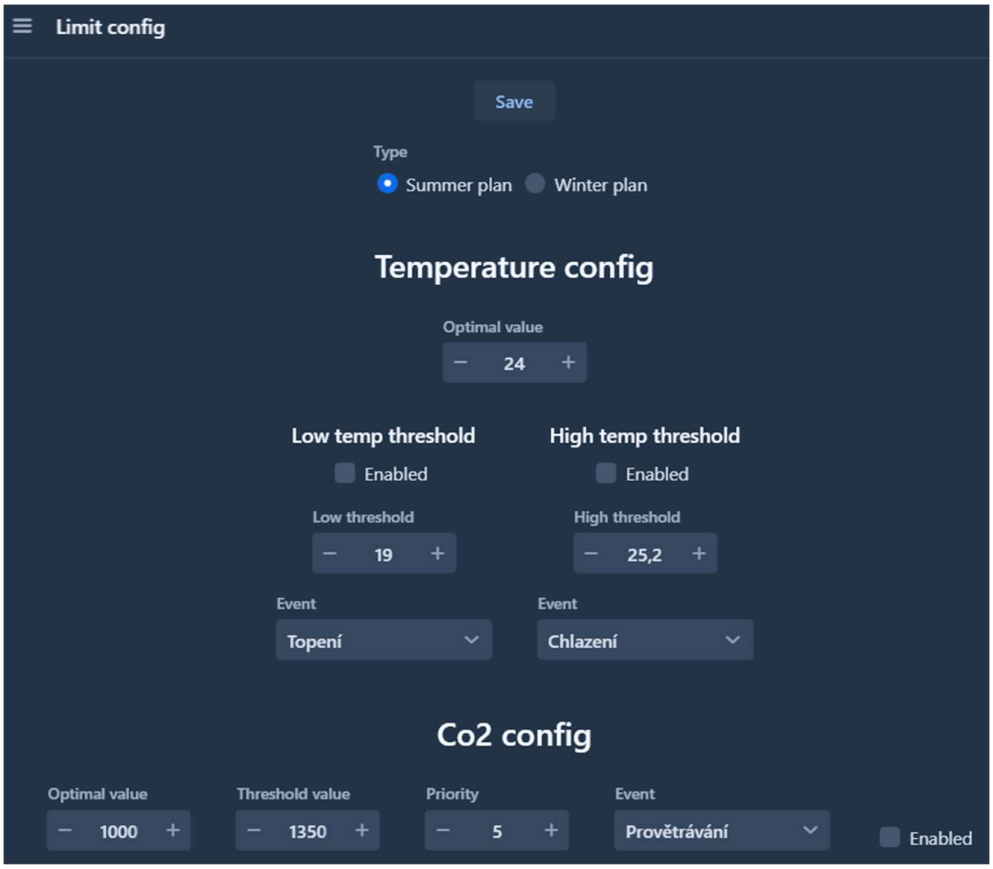
```
11  server.port=${PORT:8090}
12  spring.liquibase.change-log=classpath:db/changelog/master-changelog.xml
13
14  spring.datasource.url=${DB_URL:jdbc:mysql://localhost:3306/thesis}
15  spring.datasource.username=${DB_USER:root}
16  spring.datasource.password=${DB_PASSWORD:password}
17  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
18
19  dashboard.sensorData.maxAgeInMinutes=15
20  dashboard.refreshRate.cron=15 */1 * * * ?
```

Obrázek 42 Příklad konfiguračního souboru modulu Data manager

4.5.3.1 Limitní plány

V následujících kapitolách budou představeny konfigurace ve webové aplikaci modulu Data manager, která slouží ke konfiguraci reakcí systému na okolní prostředí. V této kapitole jde o konfigurace sloužící k nastavení limitních plánů, které se nacházejí na stránce „Limit plan“.

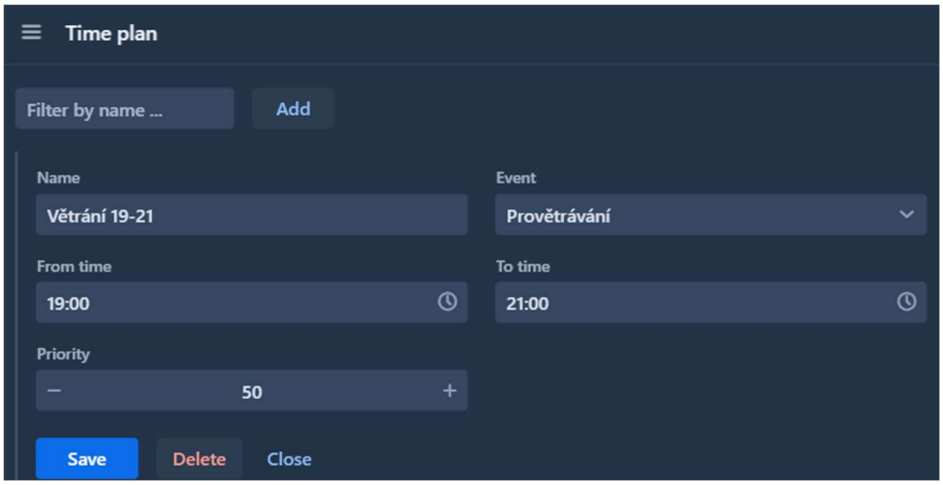
Jak je patrné z Obrázek 43 lze konfigurovat tři typy tohoto plánu, v prostřední části jsou konfigurace reakcí na hodnotu teploty prostředí vyšší a nižší než nastavené hranice. Ve spodní části se pak nachází konfigurace reagující na obsah oxidu uhličitého vyšší než nastavená hraniční hodnota. Všechny typy konfigurací mají nastavitelnou hranici, při které se aktivují, dále event, jakým na aktivaci reagují a zaškrtávací políčko značící, jestli je jejich vyhodnocování povoleno. Pro vyhodnocování plánů reagujících jak na teplotu, tak na hodnoty oxidu uhličitého existuje optimální hodnota. Ta slouží k tomu, že plán je po spuštění v aktivním režimu tak dlouho, dokud není dosaženo této hodnoty. Na obrázku lze ještě vidět přepínač mezi zimním a letním režimem. Ten poskytuje možnost mít plány nakonfigurované pro každé období jiným způsobem.



Obrázek 43 Příklad konfigurace limitního plánu v modulu Data manager

4.5.3.2 Časové plány

Následující popis konfigurace se týká časových plánů. Ty se dají nastavit na stránce „Time plan“. Příklad konfigurace lze nalézt na Obrázek 44. Z něj je viditelné, že u tohoto typu plánu se nastavuje jméno, aby byl plán v přehledu odlišitelný od ostatních. Dále event, kterým bude plán reagovat na aktivaci. Následuje nastavení dvou časů od a do, které určují časový rozsah, ve kterém bude plán aktivní. A nakonec priorita, která slouží k vyhodnocování, které aktivní plány budou využity k řízení systému.

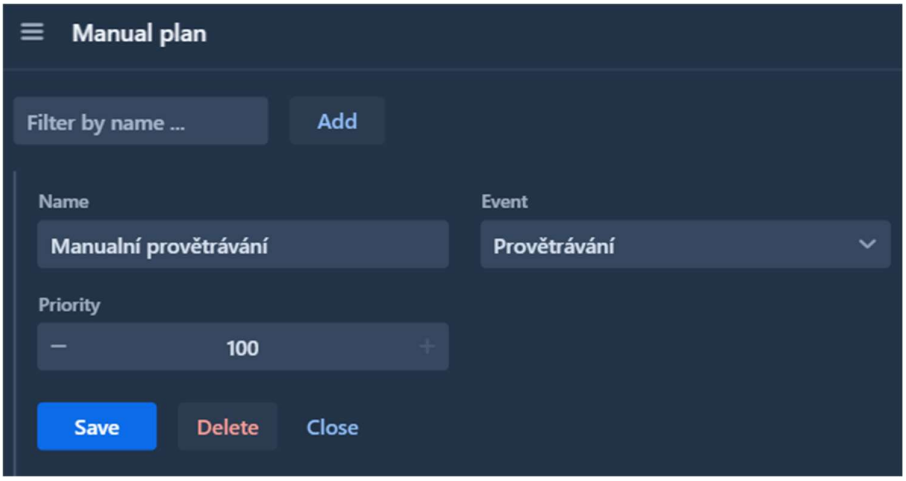


Obrázek 44 Příklad konfigurace časového plánu v modulu Data manager

4.5.3.3 Manuální plány

Dalším typem konfigurace je konfigurace manuálních plánů. Tu lze provádět na stránce „Manual plan“. Příklad nastavení lze vidět na Obrázek 45.

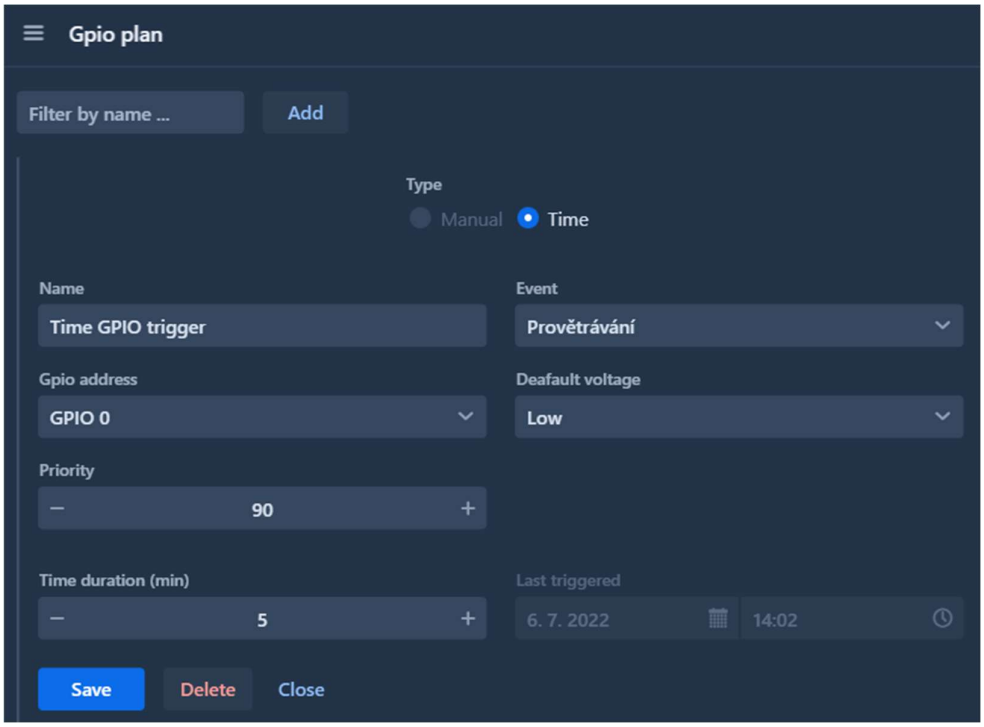
Konfigurace tohoto plánu je velice jednoduchá, sestává se pouze z nastavení jména, pod kterým plán vystupuje v přehledu manuálních plánů, eventu, který je navázán na aktivaci plánu a z nastavení priority, se kterou bude plán v případě aktivace využit. Tento typ plánu je možné spouštět z přehledu manuálních plánů pomocí zaškrtávacího políčka.



Obrázek 45 Příklad konfigurace manuálního plánu v modulu Data manager

4.5.3.4 GPIO plány

Posledním typem plánů jsou GPIO plány, ty se dají nastavit na stránce „GPIO plan“. Tento typ plánu má dva podtypy, první nazvaný Manual, ten funguje jako klasický spínač, a druhý nazvaný Time, který je po obdržení signálu aktivní po nakonfigurovanou dobu. Oba podtypy pracují s hardwarovými součástkami a jsou si konfigurací velmi podobné.



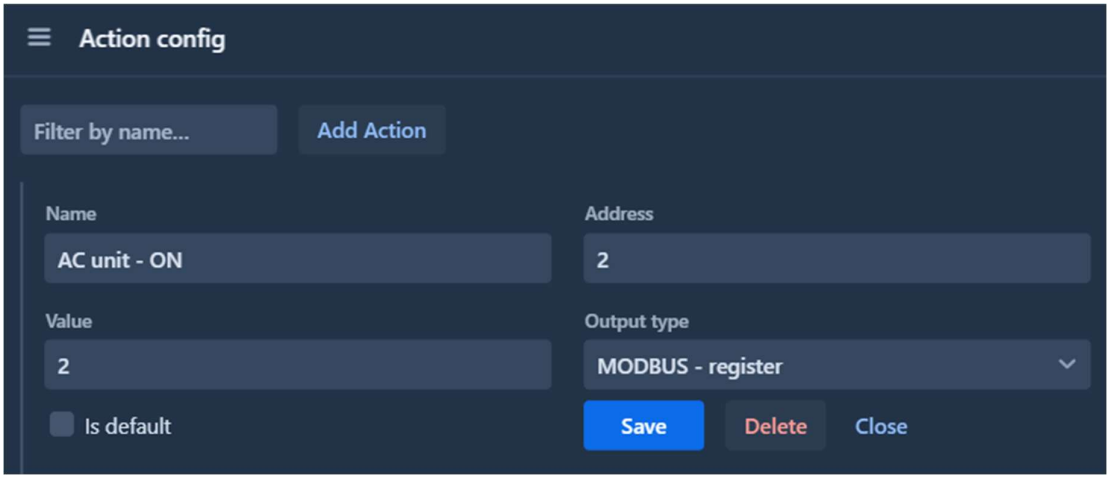
Obrázek 46 Příklad konfigurace GPIO plánu v modulu Data manager

U prvního podtypu Manual se konfiguruje název, využívaný k odlišení od ostatních plánů v přehledu. Dále se nastavuje event, kterým plán reaguje na aktivaci. Následuje nastavení adresy GPIO pinu, který bude plán monitorovat a který způsobuje aktivaci plánu. S nastavením GPIO pinu je úzce spojené nastavení výchozího napětí, ve kterém se pin nachází. To určuje, při jaké úrovni napětí bude plán aktivován. Tedy pokud je výchozí napětí plánu nastaveno na hodnotu Low, aktivuje ho hodnota High. To samé v opačném případě. Nakonec se nastaví priorita plánu, která slouží v modulu Data collector k vyhodnocování aktivních plánů.

Konfigurace druhého podtypu Time lze vidět na Obrázek 46. Jak je patrné, obsahuje všechny konfigurační položky, které obsahuje i první podtyp. V konfiguraci je navíc akorát nastavení času, po který bude daný plán po obdržení signálu aktivní. Mimo to se zde ještě nachází informace o tom, kdy byl naposledy obdržen signál aktivující plán.

4.5.3.5 Akce

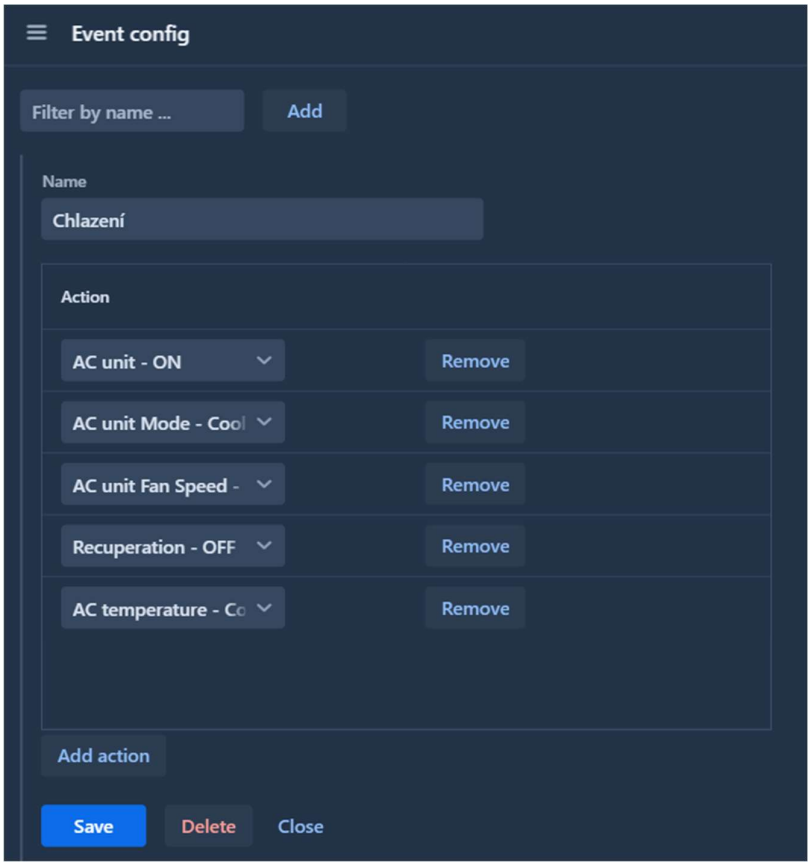
Další konfigurace zajišťuje nastavení akcí, tedy jednotek definic výstupu, které obsahují informace o tom, na jaký výstup se zapíše definovaná hodnota. Z Obrázek 47 je patrné, že jejich konfigurace je relativně jednoduchá. Obsahuje název akce, ten by měl v ideálním případě reprezentovat, s jakým výstupem se pracuje a jaký stav se nastavuje. Při dodržení tohoto doporučení jsou pak jednotlivé akce mnohem jednodušeji dohledatelné v přehledu akcí. Dále se nastavuje typ rozhraní, přes které je zařízení připojeno a jeho adresa v rámci tohoto rozhraní. Poslední částí je nastavení hodnoty, která bude na odpovídající rozhraní a adresu zapsána a nastavení, zda je tato hodnota považována za výchozí.



Obrázek 47 Příklad konfigurace akce v modulu Data manager

4.5.3.6 Eventy

S konfigurací akcí, popsanou v předchozí kapitole, úzce souvisí konfigurace eventů. Ty nejsou nic jiného než skupina sdružující jednotlivé akce do logických celků. Na Obrázek 48 lze vidět konfiguraci eventu, který sdružuje všechny akce, které je potřeba provést, aby byl systém uveden do režimu chlazení. Samotná konfigurace není nijak složitá, stačí vyplnit jméno, podle kterého je event rozlišitelný v přehledu eventů a následovně vybrat libovolný počet akcí, ze kterých se bude event skládat.

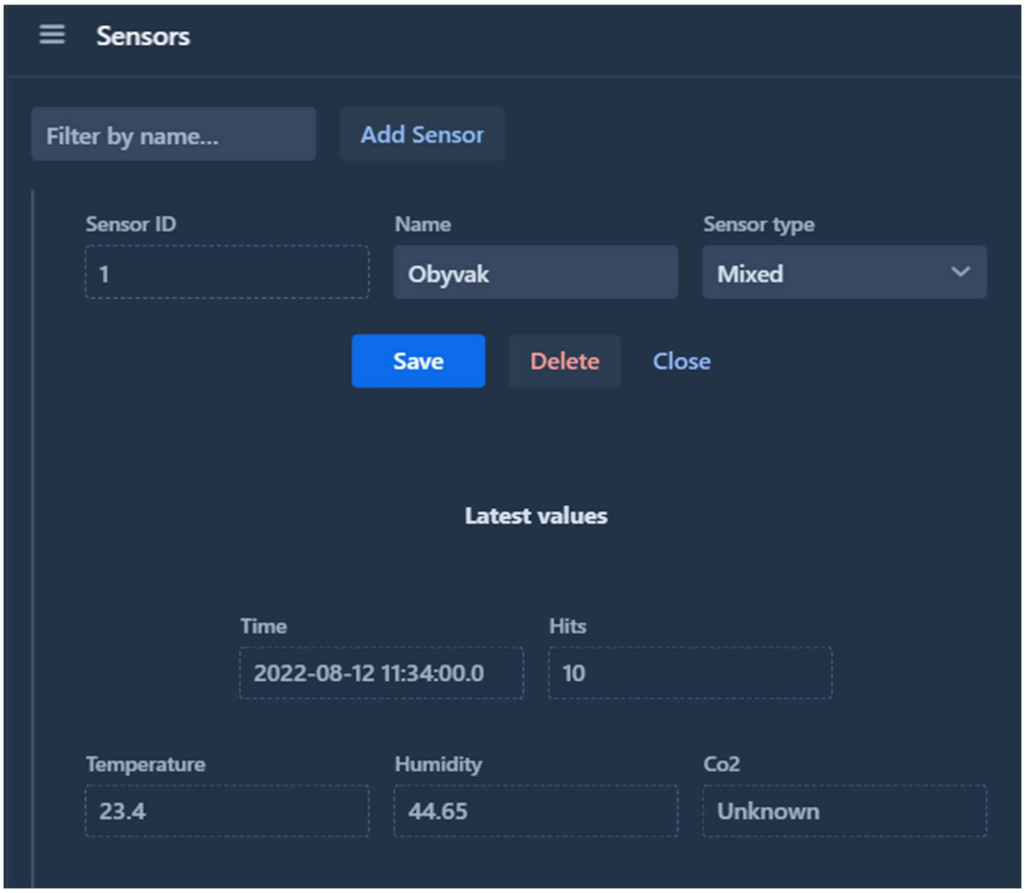


Obrázek 48 Příklad konfigurace eventu v modulu Data manager

4.5.3.7 Senzory

Následující konfigurační část je využitá ke správě senzorů, lze ji nalézt na stránce „Sensor“. Jde o konfiguraci, která není využívaná k řízení systému, ale spíše k potřebám uživatele a konfigurování modulu Data manager. Zajišťuje správu připojených senzorů, respektive informací o nich. Standardně se po připojení senzorového modulu s identifikátorem, který se ještě v systému nevyskytuje, vytvoří nový záznam do seznamu senzorů. Takto nově

vytvořený záznam obsahuje identifikátor senzoru a je opatřen názvem „UNKNOWN“. V této části lze název modifikovat, což je zároveň i doporučeno z toho důvodu, aby uživatel rozpoznal, o jaký senzor se jedná. Dále se zde dá také nastavit typ senzoru. Toto nastavení má vliv pouze na to, jakým způsobem bude jeho hodnoty možné zobrazit na stránce Dashboard. Nemá tedy vliv na to, co senzor reálně měří. Ve spodní části pak lze nalézt poslední data společně s jejich metadaty, která byla z daného senzoru obdržena.

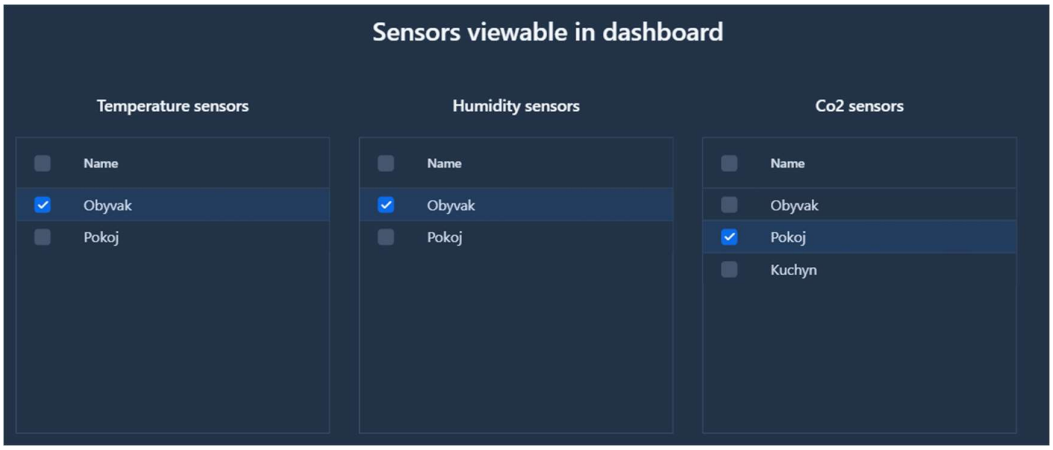


Obrázek 49 Příklad konfigurace senzoru v modulu Data manager

4.5.3.8 Další nastavení

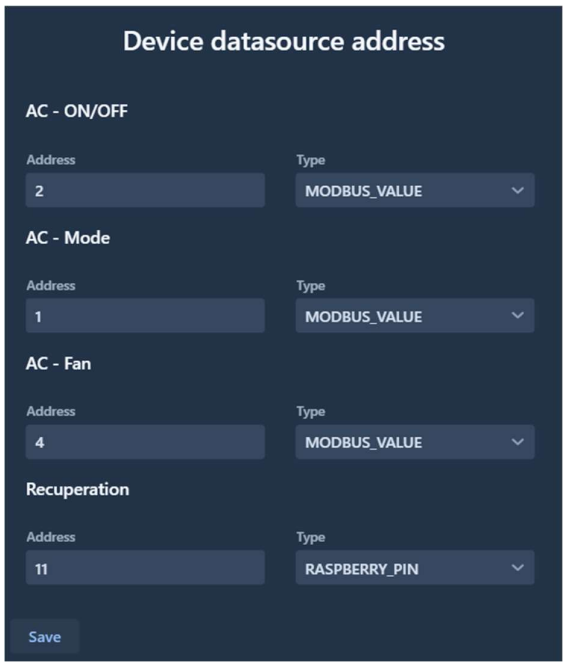
Poslední kategorie konfigurací se nachází na stránce „Settings“. Tyto konfigurace jsou využívány k vizuální modifikaci stránky Dashboard, respektive informací, které se na ní budou zobrazovat.

První, co lze v této sekci nastavit je, jaké senzory, přesněji řečeno, jaký druh senzorových dat bude zobrazen na stránce Dashboard. Senzory jsou v této části vylistovány v daných kategoriích podle toho, o jaký typ senzoru se jedná. Pokud se jedná o typ senzoru „Mixed“, je zobrazen ve všech kategoriích. V ostatních případech je zobrazen pouze v jemu odpovídající kategorii.



Obrázek 50 Příklad konfigurace typu senzorů, zobrazovaných na Dashboardu

V druhé části se pro spravované zařízení nastavují adresy a typ rozhraní, ze kterých se berou hodnoty a na základě těch se pak určuje jejich stav. Stavy těchto zařízení jsou zobrazovány na Dashboardu.



Obrázek 51 Příklad konfigurace adres spravovaných zařízení pro získání jejich stavu

5 NASAZENÍ A TESTOVÁNÍ SYSTÉMU

5.1 Nasazení do Heroku

Modul Data manager je cílený na využívání v cloudu, přesněji na platformu Heroku. Ta je velice jednoduchá na používání a do jisté míry i bezplatná (to má ovšem své omezení).

Z důvodu zjednodušení práce s cloudovou službou a zároveň k možnosti přenositelnosti mezi různými poskytovateli je celou aplikaci možné kontejnerizovat. K tomu slouží soubor Dockerfile, který zajišťuje sestavení celé aplikace, tedy jak její serverové, tak klientské části, a její následné vložení do odpovídajícího základního image. Rovněž zajišťuje všechny další konfigurace potřebné pro správný běh. Výsledkem je image obsahující celý modul.

Pro nasazení na platformu Heroku musí mít uživatel nainstalovaný nástroj Heroku CLI a musí mít otevřený terminál v kořenové složce modulu Data manager. Po splnění těchto požadavků stačí v terminálu provolávat tyto příkazy:

- „heroku login“ – Příkaz sloužící k přihlášení do platformy Heroku.
- „heroku container:login“ – Příkaz sloužící k přihlášení do kontejnerového repositáře platformy Heroku.
- „heroku create“ – Příkaz sloužící k vytvoření projektu, pod kterým bude aplikace vystupovat. Tento krok není povinný, pokud již byla aplikace vytvořena.
- „heroku container:push web“ – Příkaz slouží k provolání souboru Dockerfile, který vytvoří image aplikace. Tento image je následně umístěn do kontejnerového repositáře.
- „heroku container:release web“ – Příkaz slouží k nasazení image aplikace do používaného projektu.

Po provolání těchto příkazů by mělo proběhnout nasazení aplikace a aplikace by měla být dostupná na Heroku. V případě problému s nasazením, nebo nedostupnosti aplikace se hodí znát příkaz „heroku logs“, který zobrazí logy z aplikace.

5.2 Testování systému

Jednotlivé moduly systému, tj. jejich hardwarové, tak softwarové části, za sebou mají několik iterací testování. V počátcích implementace byla testována převážně senzorová část, se zaměřením hlavně na software zajišťující komunikaci se senzory a implementaci bezdrátové komunikace, sloužící jakožto medium pro komunikaci mezi Arduiny. V rámci tohoto testování

byly zjištěny nedostatky bezdrátové komunikace a došlo k optimalizaci a výměně hardwarových prvků.

Po úspěšném uvedení senzorového modulu do provozu, se pro potřeby testování přidal modul Data collector a systém se začal testovat jako ucelenější celek. Převážně se testovala schopnost sbírat data ze senzorů, zpracovávat je v modulu Data collector a následně je ukládat do databáze.

V poslední fázi testování během vývoje byl přidán modul Data manager a systém byl testován jako celek. Tento modul poskytoval systému konfigurace plánu a vlastně tedy určoval, jak bude systém řízen a na co bude reagovat. V rámci této fáze testování nebyly k dispozici výstupní jednotky, které by byly schopny regulovat prostředí a místo nich byly využity simulátory. V rámci této fáze bylo nalezeno a opraveno nejvíce logických chyb, zároveň se systém odladil a připravil na reálný provoz.

Aktuálně je systém nasazen v jednom případě a je využíván k monitorování a řízení domácnosti. Je tedy testován v plném rozsahu uvedeném v návrhu práce. To znamená, že jsou využity všechny jeho moduly a řídí se pomocí nich klimatizační jednotka se systémem provětrávání a rekuperační jednotkou. V rámci této fáze, která stále běží, se odlaďuje komunikace s externími zařízeními a zjišťuje se, jak se systém chová v dlouhodobějším horizontu. Zároveň je systém nasazen ve dvou případech, kdy je modul Data collector využíván pouze k monitorování prostředí a neřídí žádné externí zařízení.

6 ZÁVĚR

Hlavním cílem této práce bylo navrhnout a implementovat do reálného prostředí komplexní systém, skládající se jak z hardwarové, tak softwarové části, pro autonomní řízení klimatického komfortu budovy. To v sobě primárně zahrnuje periodické vyhodnocování teploty a kvality vzduchu (CO2) a na základě těchto hodnot systém ovládá jednotku tepelného čerpadla (topení/chlazení) a jednotku pro centrální větrání se zpětným ziskem tepla (rekuperace) s cílem dosáhnout uživatelem nastavených hodnot. Navržený systém pak přes hlavní webovou aplikaci poskytuje uživateli jednotné rozhraní pro monitorování a správu celého systému.

Písemná část práce se spíše zaměřovala na praktickou stránku diplomové práce, která je dosti obsáhlá a vyžadovala zevrubnější popis. Z tohoto důvodu není teoretická část tolik rozsáhlá. Nejprve v ní došlo k zběžnému seznámení se se samotnou problematikou chytrých budov, jejich obecným fungováním a s jejich charakteristikami. Dále byly v krátkosti představeny nejvýznamnější hardwarové a softwarové technologie, které byly využity v implementaci praktické části.

V počátku praktické části práce proběhla analýza požadavků. Na jejich základě byla vypracována základní koncepce popisující, jak by systém mohl vypadat a z jakých částí se skládat. Návrh aplikace sestává ze tří modulů: (i) modulu obsluhujícího senzory, (ii) modulu zajišťujícího sběr dat a řízení systému a (iii) modulu poskytujícího uživatelské rozhraní, využívané pro monitoring a konfiguraci systému. V další fázi došlo k výběru hardwarových komponent a k implementaci vlastního softwaru pro jednotlivé moduly. Během této fáze docházelo k testování jednotlivých částí, průběžné revizi a modifikacím dosud provedených rozhodnutí. Například bylo zjištěno, že původní hardwarová část, která měla zajišťovat bezdrátovou komunikaci mezi senzory, je pro řešení nevhodná kvůli nízké přenosové vzdálenosti a složité implementaci komunikace mezi více zařízeními a z tohoto důvodu ji bylo nutné vyměnit. Po odladění všech zjištěných nedostatků a dokončení implementací jednotlivých částí bylo pokročeno k poslední fázi vývoje. V této fázi byl systém uveden do provozu jako celek za účelem odladění zbývajících nedostatků a připravení systému do reálného provozu.

V písemné části popisující praktickou část práce, byly zachyceny výše popsané fáze. Čtenář v ní měl možnost se podrobněji seznámit s návrhem systému, který popisuje jak popis navrhovaného řešení a požadavků na systém, tak popis původního stavu. Dále byly představeny implementační detaily, ze kterých mohl čtenář získat podrobné informace o tom, jak jednotlivé

moduly fungují. V závěru jsou pak popsány možnosti konfigurace systému a způsob, jakým je možné nasadit jeho webovou aplikaci do cloudu.

Systém samotný je sice konfigurován na míru určitému řešení, nicméně během vývoje byl kladen důraz na to, aby disponoval vysokou obecností a flexibilností při konfiguraci. Neomezuje se tak pouze na řízení zmíněných zařízení, ale dokáže pracovat s jakýmkoliv zařízením využívajícím standardního průmyslového komunikačního protokolu Modbus, nebo ho je možné řídit pomocí TTL logiky. Díky těmto vlastnostem je systém využitelný pro různé typy řešení a požadavků. Zároveň poskytuje požadované jednotné webové rozhraní pro monitorování a konfiguraci jak spravovaných zařízení, tak autonomního řízení a stavů systému.

Z popisu vyhotoveného systému vyplývá, že požadavky, které na něj byly kladeny, jsou splněny v plném rozsahu. Zároveň byl systém nad rámec práce připraven na další možná rozšíření a je flexibilně využitelný na různé jiné typy řešení a požadavků.

Aktuálně je systém nasazen v reálném prostředí a dál intenzivně testován. Do budoucna se počítá s dalšími rozšířeními a stabilizací systému, a to především na základě výsledků dlouhodobého testování.

POUŽITÁ LITERATURA

[1] CLEMENTS-CROOME, Derek. Intelligent buildings design, management and operation. Londýn, Velká Británie: Thomas Telford Publishing, 2004. ISBN 978-0727732668.

[2] VAADIN TEAM. Book of Vaadin: Vaadin 14 Edition. Vaadin, 2019. ISBN 978-1692121440.

[3] SCHILDT, Herbert. Java 8: Výukový kurz. Brno, Česká republika: Computer Press, 2016. ISBN 978-8025146651.

[4] MATOUŠEK, David. C++: Výukový kurz. Brno, Česká republika: Computer Press, 2018. ISBN 978-8025149065.

[5] POHANKA, Pavel. Internet věcí. Pavel Pohanka [online]. 2017 [cit. 2020-04-16]. Dostupné z: http://www.pavelpohanka.cz/internet-of-things/

[6] AM2120 Technical Manual. V1.0. Guangzhou, China, 2019. Dostupné také z: https://www.micros.com.pl/mediaserver/UPAM2120_0004.pdf

[7] Intelligent Infrared CO2 Module: Model: MH-Z19B. V1.0. Guangzhou, China, 2016. Dostupné také z: https://www.winsen-sensor.com/d/files/infrared-gas-sensor/mh-z19b-co2-ver1_0.pdf

[8] NRF24L01 Single Chip 2.4GHz Transceiver: Product Specification. V2.0. Trondheim, Norway, 2007. Dostupné také z: https://www.sparkfun.com/datasheets/Components/nRF24L01_prelim_prod_spec_1_2.pdf

[9] 433 MHz RF Transmitter and Receiver: Module pinout, features & working. ETechnophiles [online]. [cit. 2022-08-02]. Dostupné z: https://www.etechnophiles.com/433-mhz-rf-transmitter-and-receiver-module-pinout-features-working/

[10] ASHLEY, Emma. What is Arduino Nano?: A Getting Started Guide. DesignSpark RS [online]. 29.06.2021 [cit. 2022-08-02]. Dostupné z: https://www.rs-online.com/designspark/what-is-arduino-nano-a-getting-started-guide

[11] Arduino Nano: Product Reference Manual. 02.08.2022. Via Andrea Appiani 25, 20900 MONZA MB, Italy, 2022. Dostupné také z: https://docs.arduino.cc/static/6c4e545daca62ab320ed3c9dd912d7e4/A000005-datasheet.pdf

[12] What is a Raspberry Pi?. Opensource.com [online]. [cit. 2022-08-02]. Dostupné z: https://opensource.com/resources/raspberry-pi

[13] RASPBERRY PI 3 MODEL B. 2016. Dostupné také z: https://www.terraelectronica.ru/pdf/show?pdf_file=%252Fds%252Fpdf%252FT%252FTechicRP3.pdf

[14] Modbus Application Protocol Specification [online]. Modbus-IDA, 2006 [cit. 2022-08-17]. Dostupné z: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf

[15] What is Modbus and How does it work?. Schneider electric: Life is on [online]. 19.3.2013 [cit. 2022-08-02]. Dostupné z: https://www.se.com/us/en/faqs/FA168406/

[16] Project Lombok [online]. [cit. 2022-08-17]. Dostupné z: https://projectlombok.org/features

[17] KIMBERLIN, Michael. REDUCING BOILERPLATE CODE WITH PROJECT LOMBOK. Object computing: Your outcomes engineered [online]. 12.1.2010 [cit. 2022-08-02]. Dostupné z: https://objectcomputing.com/resources/publications/sett/january-2010-reducing-boilerplate-code-with-project-lombok

[18] HAUER, Philipp. Evaluating Vaadin: Strengths and Weaknesses. Philipp Hauer's Blog [online]. 16.02.2015 [cit. 2022-08-02]. Dostupné z: https://phauer.com/2015/evaluating-vaadin-strengths-weaknesses/

[19] BEHLER, Marco. What is Spring Framework?. MB: Marco Behler [online]. 17.4.2019 [cit. 2022-08-18]. Dostupné z: https://www.marcobehler.com/guides/spring-framework

[20] Spring Framework: Official Documentation [online]. [cit. 2022-08-18]. Dostupné z: https://spring.io/projects/spring-framework/#overview

[21] MULDERS, Michiel. What Is Spring Boot?. Stackify [online]. 16.9.2019 [cit. 2022-08-18]. Dostupné z: https://stackify.com/what-is-spring-boot/

[22] Spring Boot: Official Documentation [online]. [cit. 2022-08-18]. Dostupné z: https://spring.io/projects/spring-boot#overview

[23] Docker: Reference documentation [online]. [cit. 2022-08-18]. Dostupné z: https://docs.docker.com/reference/

[24] CAREY, Scott. What is Docker?: The spark for the container revolution. InfoWorld [online]. 2.8.2021 [cit. 2022-08-18]. Dostupné z: https://www.infoworld.com/article/3204171/what-is-docker-the-spark-for-the-container-revolution.html

[25] MySQL 8.0: Reference Manual [online]. [cit. 2022-08-18]. Dostupné z: https://spring.io/projects/spring-boot#overview

[26] BOYETT, Richard. What is MySQL: MySQL Explained For Beginners. Hostinger: Tutorials [online]. 26.7.2022 [cit. 2022-08-18]. Dostupné z: https://www.hostinger.com/tutorials/what-is-mysql

[27] TYSON, Matthew. What is JDBC?: Introduction to Java Database Connectivity. InfoWorld: Tutorials [online]. 13.5.2022 [cit. 2022-08-18]. Dostupné z: https://www.infoworld.com/article/3388036/what-is-jdbc-introduction-to-java-database-connectivity.html

[28] GRZEGORCZYK, Rafał. What is Liquibase and how to start using it?. Pretius [online]. 22.3.2021 [cit. 2022-08-18]. Dostupné z: https://pretius.com/blog/liquibase-tutorial/

[29] Liquibase: Documentation [online]. [cit. 2022-08-18]. Dostupné z: https://docs.liquibase.com/home.html

[30] Pi4J: Documentation. The Pi4J Project: Java I/O library for Raspberry Pi [online]. [cit. 2022-08-18]. Dostupné z: https://pi4j.com/documentation/