Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Řízení teploty v domě na základě dat ze senzorů

Bakalářská práce

Autor: Miloš Olmr
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové duben 2018

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 26.4.2018 Miloš Olmr

Poděkování:

Rád bych poděkoval vedoucímu bakalářské práce doc. Ing. Filipu Malému, Ph.D. za cenné rady, věcné připomínky a vstřícnost během zpracovávání této práce. Dále děkuji své rodině za veškerou podporu a obzvlášť sestřenici Janě za korekturu.

Anotace

Bakalářská práce se zaměřuje na návrh a realizaci Java EE aplikace pro regulaci teploty v domě s využitím digitálních senzorů DS18B20 připojených k Raspberry Pi (nebo obdobnému zařízení). Součástí je rovněž webové rozhraní poskytující přehled (zahrnující výstup v podobě grafů) s možností regulaci ovlivňovat. Aplikace je navržena modulárně podle architektury Java EE tak, aby byla jednoduše rozšiřitelná a umožňovala nasazení podle různých scénářů. Samotný proces regulace obstarává termoregulátor fungující na principu dvoubodového regulátoru, který je modulem aplikace, a pro výstup využívá rozhraní GPIO. Veškerá získávaná data ze senzorů jsou uchovávána v relační databázi za účelem budoucího vyhodnocování.

Annotation

Title: In-house temperature control based on sensor data

This bachelor thesis focuses on design and creation of Java EE application for in-house temperature control using digital sensors DS18B20 connected to Raspberry Pi (or similar device). This also includes web interface providing overview (embracing output in charts) and allowing to influence regulation. This application is designed as modular based on Java EE architecture to provide easy scalability and deployment based on different scenarios. Process of regulation itself is performed using on-off regulation method by thermo regulator, which is module of application and it uses GPIO interface as output. All collected data from sensors are stored in relational database for future evaluation.

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Dnešní doba je dobou „chytrých“ (smart) zařízení. Relativně moderním pojmem je „Internet věcí“ (Internet of Things, zkráceně IoT). Jedná se o propojení fyzických zařízení, které spolu komunikují. Pokud se jedná o takové propojení zařízení v rámci domu, tak mluvíme o „chytré“ domácnosti (smart home). Najdou se lidé, kteří si vytváří vlastní řešení „chytré“ domácnosti za pomocí miniaturního jednodeskového počítače Raspberry Pi. Nabízí se řada softwarů pro domácí automatizaci, které lze využít. Tato práce se zaměřuje na vytvoření softwaru pro řízení vytápění (teploty) v domě.

Cílem práce je navrhnout a realizovat Java EE aplikaci pro regulaci teploty v domě s využitím senzorů připojených k Raspberry Pi (nebo obdobnému zařízení) včetně webového rozhraní pro ovládání. Téma bylo zvoleno kvůli vlastnímu zájmu o tuto oblast a potřebě po dokonalejším řízení vytápění v rodinném domě. Věřím ale, že toto řešení regulace vytápění bude užitečné i dalším.

Metodika, podle které se v této práci postupuje, je částí vývojového procesu (využívaného při vývoji informačních systémů) s názvem Vodopádový model. Nejprve se provede analýza problému, jejíž výstupem jsou požadavky na budoucí systém. Následně se navrhne systém (návrh zahrnuje volbu technologií). Podle návrhu pak proběhne implementace. Dalším částem, které jsou testování, nasazení a údržba, se tato práce nevěnuje. Analýze předchází část Teoretické podklady poskytující základní poznatky z oblasti regulace ve vytápění, které jsou později aplikovány.

2 Teoretické podklady

Účelem této kapitoly je představit základní teorie z oblasti regulace ve vytápění, na kterých se v této bakalářské práci staví. Většina informací je čerpána z práce Jiřího Doubravy a kolektivu s názvem Regulace ve vytápění.

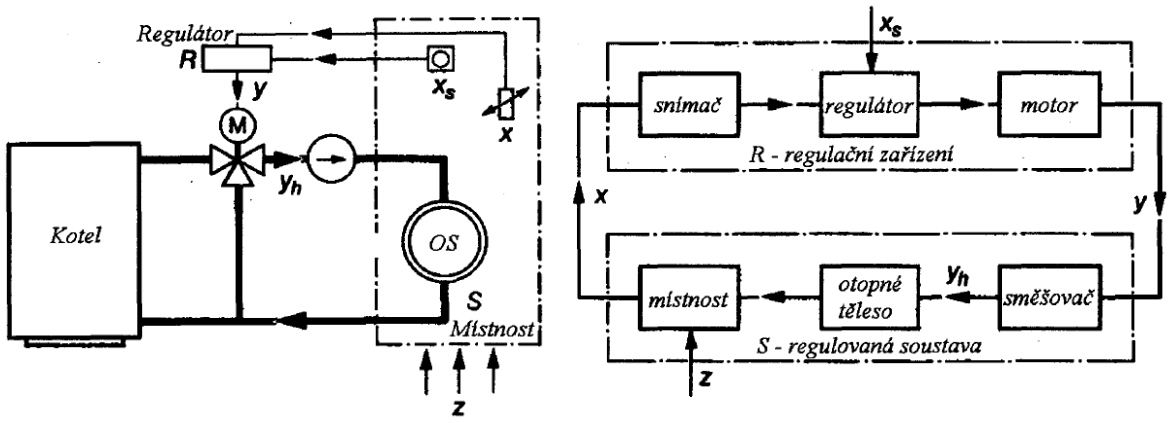
2.1 Základní pojmy regulace ve vytápění

Regulace je proces, při kterém se snažíme udržovat určitou *regulovanou veličinu* na předem zvolené konstantní hodnotě. Část zařízení, ve kterém se uskutečňuje regulace, se nazývá *regulovaná soustava* (1, s. 14). V našem případě bude regulovanou veličinou teplota v domě.

„Nejdokonalejší regulační soustavou se všemi prvky regulace je člověk sám, který dokáže rychle nebo pomalu reagovat na vnější podněty, aniž by musel zvlášť nastavovat tzv. „konstanty regulátoru“.“ (2). Ovšem v dnešní zrychlené době moc lidí na ruční regulaci nemá čas, proto se začalo automatizovat. Samočinná regulace je prováděna pomocí tzv. *regulátorů*.

Bašta (3, s. 161–162) definuje další pojmy z oblasti regulace:

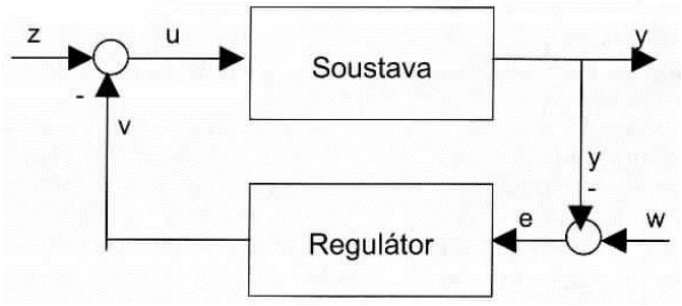
- *požadovaná hodnota* – je zvolená hodnota, na které se regulátor snaží udržet regulovanou veličinu,
- *poruchová veličina* – veličina působící zvenku na regulovanou soustavu a způsobující odchylku od požadované hodnoty,
- *akční veličina* – výstupní hodnota regulátoru, kterou ovlivňuje regulovaná veličina,
- *skutečná hodnota* – naměřená hodnota v určitém čase,
- *regulační odchylka* – rozdíl mezi požadovanou hodnotou a skutečnou hodnotou,
- *akční člen* – část regulačního obvodu, která je ovlivňována akční veličinou.



Obr. 1: Princip regulace teploty v místnosti pomocí centrálního vytápění (3, s. 163)

2.2 Regulační obvod

Regulace je formou řízení se zpětnou vazbou, ve kterém regulátor představuje řídící subsystém a regulovaná soustava je subsystémem řízeným. Právě díky zpětné vazbě získáváme v systému regulace uzavřený obvod.



Obr. 2: Uzavřený regulační obvod se zpětnou vazbou (1, s. 15)

V regulačním obvodu probíhá regulační pochod, kterým rozumíme proces od okamžiku, kdy dojde ke vzniku regulační odchylky, až do jejího odstranění. K odchylce může dojít vlivem poruchové veličiny, potom hovoříme o reakci na poruchu, nebo změnou požadované hodnoty. Regulační pochod lze zaznamenávat graficky jako časovou závislost regulované veličiny (1, s. 15).

2.3 Snímače

Snímače se starají o převádění určité fyzikální hodnoty na signál vhodný k přenosu informace o této veličině a k dalšímu zpracování třeba právě regulátory. Pod pojmem *čidlo* rozumíme vlastní měřící část, za to snímač je již konkrétní přístroj. Při volbě čidla si musíme dát pozor na údaje jako: časová konstanta přenosu,

nelinearita, přesnost čidla, vhodnost do určitého prostředí a typ připojení. Takové teplotní snímače umístěné v jímkách mívají časovou konstantu až několik minut (1, s. 26–27).

Rozlišujeme několik druhů snímačů: teploty, vlhkosti, tlaku, průtoku a kvality ovzduší. Každý druh snímače se pak dále ještě dělí podle principu měření. Volbou vhodného snímače, jeho správným umístěním a pravidelnou údržbou, dosáhneme kvalitní regulace (2).

Obr. 3: Ukázka různých druhů snímačů (4)

2.3.1 Umístění snímačů

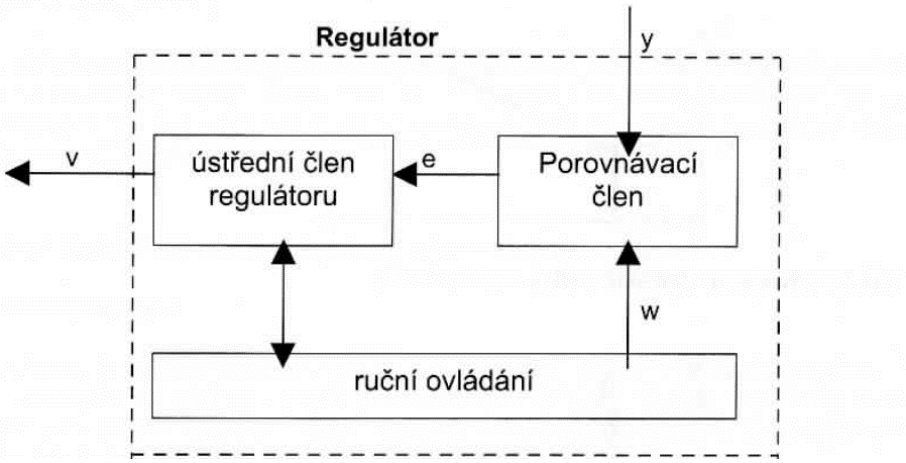
Volba vhodného umístění pro snímač je klíčový moment, protože v případě špatného umístění bude snímač vracet nepřesné nebo dokonce naprosto jiné hodnoty, než které by měl snímač správně naměřit. Nejdůležitější je správné umístění teplotních čidel.

Teplotní snímače musí být umístěny ve výšce přibližně 1,5 m nad zemí a minimálně 50 cm od rohu místnosti. Pod snímačem se v žádném případě nesmí nacházet zdroj tepla. Rozhodně se čidla nesmí umisťovat poblíž dveří a oken, protože v těchto místech může docházet k ovlivnění vzduchem s jinou teplotou. Teplotní snímače nesmějí být umístěny v místech, kam mohou dosáhnout sluneční paprsky. Rovněž se nesmí připevňovat na venkovní stěnu, protože by mohlo v zimních měsících docházet k nadměrnému ochlazování a v letních na jižní straně naopak k nadměrnému ohřívání. Teplotní čidla nepatří do míst, kde nedochází k cirkulaci vzduchu (1, s. 40).

Co se týče umístění venkovního teplotního snímače, pro něj také existuje několik pravidel. Měl by být situován na severní straně domu a nesmí být ovlivněn přímým slunečním zářením. Výjimku tvoří pouze prosklené fasády a převažující okenní plochy (1, s. 40).

2.4 Regulátory

Regulátorem se v regulační technice nazývá veškerá přístrojová technika, která je připojená k technologickému zařízení za účelem jeho regulace (2). Podle fyzikálního principu se regulátory rozlišují na mechanické, pneumatické, hydraulické a elektrické. Podle funkce pak na spojité a nespojité regulátory. Spojité regulátory jsou technicky dokonalejší, ovšem nespojitou regulací lze zjednodušit konstrukci a vyrábět regulátory levněji (1, s. 16–17).



Obr. 4: Obecné funkční schéma regulátoru (1, s. 17)

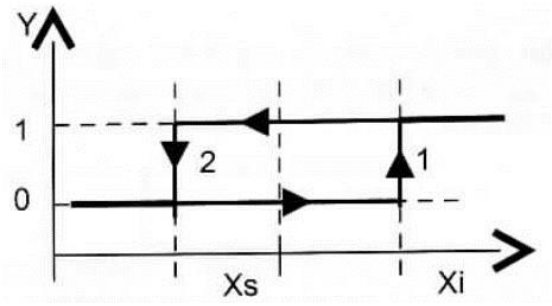
Na obr. 4 lze vidět, že „porovnávací člen určuje hodnotu regulační odchylky e odečtením měřené hodnoty y regulované veličiny od hodnoty řídící veličiny w podle vztahu e=w-y.“ Hodnotu regulační odchylky e vypočtenou porovnávacím členem pak dále zpracovává ústřední člen regulátoru, jehož výstupem je akční veličina v. Ústřední člen regulátorů může být tvořen až třemi částmi: proporcionální složkou, integrační složkou a derivační složkou (1, s. 17).

2.4.1 Dvoubodový regulátor

Dvoubodový regulátor je druh nespojitého regulátoru, ve kterém akční veličina může nabývat pouze dvou hodnot: nulové a jmenovité. Regulátor tohoto typu může

upravovat regulovanou veličinu jedině ve směru zvětšování. Použití je možné pouze v regulovaných soustavách, kde po nastavení akční veličiny na nulovou hodnotu dochází k samovolnému snižování regulované veličiny (1, s. 19). Typickým příkladem dvoubodového regulátoru je termostat.

Akční veličina tedy kmitá mezi jmenovitou a nulovou hodnotou. Regulátor musí být vybaven mechanismem, který zajistí určitou diferenci kolem požadované hodnoty, jinak by kmitání bylo příliš rychlé a mohlo by dojít až ke zničení akčního členu. Nadměrně časté změny by také zbytečně zatěžovaly zbytek systému (1, s. 19).



Obr. 5: Regulační pochod uvnitř dvoubodového regulátoru (1, s. 20)

Princip regulace dvoubodovým regulátorem je znázorněn na obr. 5. Osa Xi představuje regulační odchylku a osa Y akční veličinu. Bod Xs je požadovaná hodnota. Pokud regulační odchylka vzroste a její hodnota se bude nacházet v bodě vzdáleném o danou diferenci od bodu Xs, pak dojde ke změně akční veličiny podle šipky 1. Nyní dochází k nápravě odchylky. V momentě, kdy regulační odchylka dostatečně poklesne, dojde ke změně podle šipky 2. Na grafu z obr. 5 je vidět, že dochází ke změně, pouze pokud se regulační odchylka dostatečně vychýlí, jinak si akční veličina zachovává předchozí hodnotu (1, s. 19).

3 Analýza

Tato kapitola se zabývá analýzou problému, kterým je řízení teploty v rodinném domě. V regulačním obvodu (viz kapitola 2.2 Regulační obvod) je klíčovým prvkem regulátor, který zajišťuje požadované řízení. Regulátor má nějaký vstup a výstup. Vstupem jsou (v tomto případě) teplotní data ze snímačů a výstupem je akční veličina, která ovládá akční člen (většinou topný kotel). U regulátoru je dobré, pokud také disponuje určitým typem ovládání, aby si uživatel mohl snadno nastavit např. požadovanou teplotu. V následujících podkapitolách se detailněji rozeberou klíčové části požadovaného regulátoru.

3.1 Sběr dat

Vstupem regulátoru jsou teplotní data získávaná ze senzorů. Termostat (běžný regulátor) má většinou pouze jeden zabudovaný snímač. Tím je termostat schopný regulovat teplotu na základě teplotních dat pouze z jedné místnosti. Tato práce se zaměřuje na návrh a realizaci regulátoru, který bude umět regulovat teplotu na základě dat z několika připojených vnitřních a venkovních senzorů.

Vzhledem k tomu, že regulátor je zařízení, ke kterému se připojují snímače, tak by bylo vhodné, aby uměl data ze snímačů také zobrazit uživateli (kromě využití při regulaci). Uživateli se mohou tato data hodit k různým účelům (např. při nastavování termostatických ventilů v místnostech). Pokud bude regulátor sloužit také pro zobrazování dat ze senzorů, bylo by vhodné, aby si uživatel mohl vybrat, které senzory se budou využívat při regulaci a které nikoliv.

Předpokládá se vytvoření regulátoru ze zařízení typu Raspberry Pi (Raspberry Pi, Orange Pi nebo Banana Pi) s operačním systémem GNU/Linux. K těmto zařízením je nejčastěji využívaným teplotním snímačem DS18B20 od společnosti Maxim Integrated, jehož využití se bude předpokládat i v této práci.

3.1.1 Senzor DS18B20

Jedná se o digitální teplotní snímač s číslicovým výstupem ve stupních Celsia. Senzor má nastavitelnou přesnost od 9 bitů do 12 bitů a komunikuje prostřednictvím 1-Wire sběrnice. Je schopný měřit teplotu v rozsahu od -55 °C do +125 °C (5, s. 1). Pro jednoznačnou identifikaci senzorů má každý senzor od výroby ve své paměti uložený

unikátní 64 bitový sériový kód. Tento kód sestává z 8 bitů pro označení „rodiny senzorů“ (hodnota: 28), 48 bitů pro samotné sériové číslo a dalších 8 bitů pro CRC součet (5, s. 8). Příkladem takového kódu bez CRC je: 28-0316B3BCDDFF.

Obr. 6: Zapouzdřený teplotní senzor DS18B20 (zdroj: vlastní fotografie)

Pokud je senzor správně připojen a jsou načteny moduly jádra s ovladači pro 1-Wire sběrnici, pak je možné v příkazové řádce přečíst aktuálně naměřenou hodnotu. Nejprve je vhodné zjistit počet připojených zařízení. To se provede pomocí příkazu: „`cat /sys/devices/w1_bus_master1/w1_master_slave_count`“. Případný seznam senzorů lze získat pomocí příkazu: „`ls -l /sys/bus/w1/devices/`“. Samotné získání teplotního údaje je možné provést pomocí zadaní příkazu: „`cat /sys/bus/w1/devices/28-XXXXXXXXXXXX/w1_slave`“. Z výstupu tohoto příkazu lze kromě naměřené hodnoty zjistit i výsledek kontrolního CRC součtu (6).

3.2 Regulace teploty

Důležitým prvkem regulátoru je jeho algoritmus. Podle vnitřního algoritmu regulátor rozhoduje o tom, kdy se bude topit a jak dlouho se bude topit. Algoritmus dostává na vstupu teplotní data společně s aktuálním nastavením a výstupem je akční veličina. Mezi vstupem a výstupem se nachází porovnávací člen a ústřední člen regulátoru (viz obr. 4 v kapitole 2.4 Regulátory). V této práci se bude navrhovat a implementovat nejčastější typ regulátoru – termostat (dvoubodový regulátor).

Jelikož v této práci uvažujeme možnost více připojených vnitřních senzorů, které budou využity při regulaci teploty, musí se vstup předzpracovat tak, aby výsledkem byla jediná hodnota, která se bude dále vyhodnocovat porovnávacím

členem. Způsobem, jak tento problém řešit, může být aritmetický průměr. Žádanou funkcionalitou by mohla být možnost nastavit senzorům určitou váhu podle důležitosti jednotlivých místností (např. senzor v obývacím pokoji bude mít vyšší váhu než senzor na chodbě, protože v obývacím pokoji se tráví více času) a počítat vážený aritmetický průměr.

Porovnávací člen v algoritmu vypočte odchylku. Ústřední člen regulátoru představuje v algoritmu vyhodnocení podmínek. Nejprve se vyhodnotí odchylka, která se porovná s diferencí. Stanovení diference řeší problém s příliš častou změnou akční veličiny, který byl popsán v kapitole 2.4.1 Dvoubodový regulátor. Následuje porovnání aktuální teploty (skutečné hodnoty) s požadovanou teplotou. Pokud je skutečná hodnota nižší než požadovaná a zároveň je hodnota akční veličiny menší než 1, pak se akční veličina nastaví na hodnotu 1. Jestliže je skutečná hodnota vyšší než požadovaná, akční veličina nastaví na hodnotu 0 (pokud její předchozí hodnota byla větší nebo rovno 1). Následující zdrojový kód obsahuje algoritmus ústředního členu znázorněný pomocí pseudokódu.

```
if ( odchylka >= diference/2 ) {
     if ( teplota < pozadovanaTeplota && akcniVelicina < 1 ) {
            akcniVelicina = 1;
     }
     else {
            if ( akcniVelicina >= 1 ) {
                  akcniVelicina = 0;
            }
     }
}
```

Zdrojový kód 1: Algoritmus ústředního členu dvoubodového regulátoru

3.2.1 Výstup

Zařízení typu Raspberry Pi je vybaveno rozhraním GPIO. Toto rozhraní obsahuje několik kontaktů („pinů“), které se dají nastavit buďto jako vstupní nebo výstupní (7). Tyto výstupní „piny“ lze využít pro řízení jiných elektronických zařízení jako třeba topný kotel (akční člen).

3.3 Ovládání

Nabízí se několik způsobů ovládání regulátoru (několik rozhraní): fyzické (tlačítka a displej), pomocí aplikace, nebo webové. Právě poslední zmíněná možnost má oproti ostatním výhodu, protože umožňuje ovládání regulátoru odkudkoliv, z jakéhokoliv zařízení (zařízení s nainstalovaným webovým prohlížečem) a bez nutnosti instalace dalšího software.

3.3.1 Webové rozhraní

Bylo by vhodné, aby uživatelské rozhraní obsahovalo: nastavení regulátoru, správu připojených senzorů a aktuální informace o stavu. Nastavení regulátoru by mělo zahrnovat: požadovanou teplotu, diferenci a hranici venkovní teploty. Užitečným rozšířením by byla možnost definování požadovaných teplot pro různé denní doby, jak to umožňují běžné digitální termostaty. Správa senzorů představuje: zobrazení aktuálně připojených senzorů, pojmenování senzorů a vybrání pro zahrnutí při regulaci teploty (včetně nastavení váhy senzoru). Aktuální informace o stavu by měly obsahovat minimálně informaci o tom, zda se aktuálně topí a nejnovější teplotní údaje ze senzorů.

3.4 Závěr analýzy

Cílem analýzy bylo rozebrat problém řízení teploty v domě na základě dat ze senzorů, přijít s řešeními a definovat požadavky na budoucí systém. Na základě souhrnu požadavků je potřeba navrhnout systém, který bude zastávat regulátor v regulačním obvodu. Výslednou aplikaci bude možné nasadit na zařízení typu Raspberry Pi s připojenými teplotními senzory prostřednictvím 1-Wire sběrnice a připojeným akčním členem na rozhraní GPIO. Aplikace musí fungovat na operačním systému GNU/Linux.

Systém bude sám sbírat data z připojených teplotních senzorů a ukládat je pro pozdější vyhodnocování. Bude provádět regulaci teploty na základě dat řízením akčního členu a také bude poskytovat webové rozhraní pro možnost ovládání.

4 Návrh systému

Kapitola se věnuje návrhu systému na základě provedené analýzy a vyvozených požadavků. Nejprve se zvolí technologie. Pak následuje popis architektury systému, následně i detailní popis jednotlivých částí.

4.1 Technologie

Jedním z požadavků je ovládání přes webové rozhraní. Dalším nárokem je funkčnost na operačním systému GNU/Linux. Technologie, která tyto požadavky splňuje, a navíc nabízí mnoho dalšího (včetně škálovatelnosti), je Java Enterprise Edition (Java EE). Pro lepší práci s daty je dobré využít systém řízení báze dat. Rozšířenou technologií je MySQL.

4.1.1 Java EE

Java Enterprise Edition je sada specifikací rozšiřující standardní verzi Java, mezi které patří specifikace pro vývoj: webových rozhraní, webových služeb, vrstvy modelu, perzistentní vrstvy a klientů aplikací. Java EE aplikace se nasazuje na aplikační server (např. GlassFish nebo JBoss). Pokud se jedná pouze o webovou aplikaci, je možné ji nasadit do webového „kontejneru“ (např. Apache Tomcat). Java EE poskytuje řešení pro transakční zpracování, škálovatelnost, bezpečnost, paralelní zpracování a správu komponent (8).

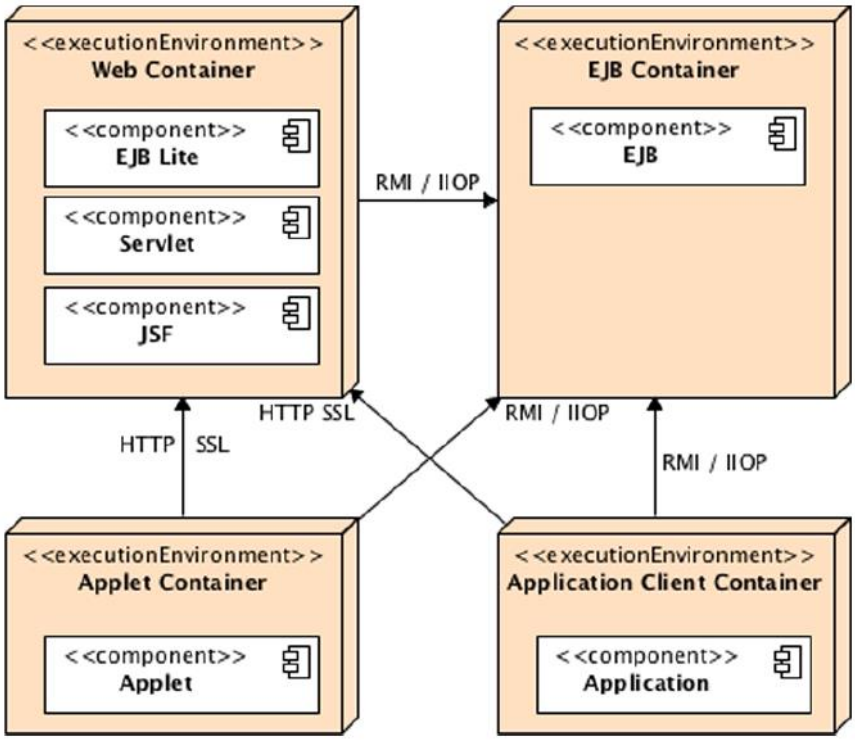
4.1.2 MySQL

MySQL je multiplatformní systém řízení báze dat s otevřeným zdrojovým kódem fungující na principu relačního databázového modelu. MySQL je implementováno v jazycích C/C++ a využívá dotazovací jazyk SQL (9). Pro práci s databázovým systémem v prostředí Java se využívá rozhraní JDBC (Java Database Connectivity). MySQL toto rozhraní podporuje a poskytuje JDBC ovladač (MySQL Connector/J).

4.2 Architektura

Architektura Java EE je množina specifikací, které jsou implementovány různými „kontejnery“. Kontejnery jsou prostředí, které poskytují určité služby

komponentám, které se v nich nachází. EJB kontejner může obsahovat komponenty EJB (Enterprise Java Bean). Jedná se o model (logiku) aplikace. Nad modelem může být postaveno webové rozhraní a GUI (grafické uživatelské rozhraní) v podobě klienta aplikace nebo appletu. Webové rozhraní se umisťuje do webového „kontejneru“, applet a klient aplikace běží ve svém vlastním. Komponenty z těchto kontejnerů spolu mohou komunikovat. Veškeré komponenty aplikace z jednoho „kontejneru“ tvoří modul aplikace. Java EE aplikace se skládá z modulů pro jednotlivé kontejnery. Moduly se nemusí nutně nacházet v kontejnerech pouze na jednom aplikačním serveru, ale mohou se nacházet na více aplikačních serverech a komunikovat spolu prostřednictvím sítě (10, s. 2-7). Tato architektura je znázorněna na obr. 7.



Obr. 7: Architektura Java EE (10, s. 2)

Systém pro řízení teploty v domě bude navržen na této architektuře. Základem bude modul jádra pro EJB kontejner. Modul jádra bude zprostředkovávat základní služby včetně sběru dat ze senzorů. Webové rozhraní bude modul pro webový kontejner. Teplotní regulátor bude klientem aplikace, který bude využívat služby EJB modulu vytápění.

Aby bylo dosaženo co největší rozšiřitelnosti, je potřeba zajistit, aby bylo možné sbírat teplotní data z více zařízení. Při současném návrhu je to možné pouze

ze zařízení, na kterém se nachází modul jádra. Přístup ke vzdálenému zařízení s připojenými senzory lze zajistit využitím SSH. Pomocí SSH lze po autentizaci spouštět příkazy na vzdáleném zařízení a získávat odpovědi (11). Takto navržený systém lze provozovat celý na jednom zařízení nebo jako síť několika zařízení.

4.2.1 Varianta jediného zařízení

Jedná se o nejjednodušší způsob, jak bude systém možné nasadit. Aplikační server (včetně aplikace), databázový systém MySQL a teplotní regulátor (modul klienta aplikace) na jednom zařízení typu Raspberry Pi, ke kterému jsou připojeny teplotní senzory a relé ovládající topný kotel.

4.2.2 Síť zařízení

Komplexnější variantou, ve které jsou jednotlivé prvky rozděleny mezi více zařízení propojených počítačovou sítí, je vytvoření sítě zařízení. Aplikační server včetně databázového systému může být nasazen na serverový stroj s dostatkem zdrojů, teplotní regulátor bude ovládat akční člen prostřednictvím jednoho zařízení typu Raspberry Pi a teplotní senzory mohou být připojeny k dalším zařízením. Veškerá zařízení propojená místní sítí LAN.

4.3 Jádro

Modul jádra bude poskytovat základní služby a bude obsahovat základní entity modelu aplikace. Mezi významné služby patří především: `GatheringService`, `TemperatureService` a `ModuleService`. Ostatní nejsou tolik zajímavé a v této práci nebudou detailně rozebírány. Pouze provádějí základní CRUD operace nad příslušnými entitami. Konkrétně jde o tyto služby: `GatheringUnitService`, `RecordService`, `RoleService`, `SensorService`, `SettingService` a `UserService`.

Každá služba bude definována svým rozhraním, které bude typu `@Local`. Rozhraní s anotací `@Local`, lze využívat pro lokální volání metod (v rámci jednoho JVM). Je možné pak z jednoho modulu aplikace („kontejneru“) využívat služby, které nabízí jiný modul aplikace (10, s. 235).

Základními entitami budou následující třídy: `Gather`, `GatheringUnit`, `LoginDetails`, `Module`, `ModuleSensor`, `Record`, `Role`, `Sensor`, `Setting`, `Temperature`

a `User`. Instance těchto tříd bude možné uložit do relační databáze – z toho vyplývá objektově relační mapování. Odpovídající tabulky v relační databázi pro jednotlivé třídy budou nazvány obdobně jako třídy, pouze v množném čísle (např. tabulka pro třídu `Sensor` se bude jmenovat `Sensors`). Veškeré tyto třídy budou mít implementovány metody: `equals()`, `hashCode()` a `toString()`. Obecně se doporučuje tyto metody překrýt (12, s. 182). Také budou implementovat rozhraní `Serializable`, jelikož by se mohlo do budoucna chtít instance těchto tříd serializovat a je dobré s tím počítat. Detailní popis těchto tříd bude v kapitole 4.3.4 Návrhový model tříd.

4.3.1 Služba GatheringService

Velice podstatnou službou v jádře bude `GatheringService`. Tato služba bude zodpovědná za sběr dat ze senzorů v pravidelných intervalech. Nebude vystavovat žádné rozhraní (`@Local` nebo `@Remote`). Bude mít pouze tzv. „no-interface“ pohled (anotaci `@LocalBean`). Tuto službu tedy nebude možné využívat z žádného jiného modulu aplikace (10, s. 235-237).

Sběru dat v pravidelných intervalech bude docíleno využitím služby Timer Service platformy Java EE. Ta umožňuje zaregistrování určité metody pro opakované volání v nastaveném intervalu. Zaregistrování je možné buďto automaticky pomocí anotace `@Schedule` nebo manuálně v kódu aplikace (10, s. 268-273). Interval sběru dat bude standardně 5 minut. Toto nastavení bude možné změnit v jednom z konfiguračních souborů aplikace („deployment descriptoru“).

Metoda vykonávající získávání a ukládání dat se bude nazývat `gatherDataFromGatheringUnits`. Při získávání dat bude využívat dalšího prvku s názvem `GUConnectivity` („Gathering Unit Connectivity“). Tento prvek sestává z rozhraní, které obecně definuje metody pro připojení ke sběrné jednotce a získání dat ze senzorů. Podle parametrů připojení ke sběrné jednotce lze pak volit konkrétní implementace `GUConnectivity` pro získávání dat z různých zařízení různým způsobem. Aktuálně bude k dispozici pouze jediná implementace umožňující připojení k zařízení typu Raspberry Pi s operačním systémem GNU/Linux pomocí SSH.

4.3.2 Služba TemperatureService

Další důležitou službou bude služba `TemperatureService` pro práci s teplotními daty. Bude ji využívat například právě `GatheringService` pro ukládání získaných dat. K tomu bude sloužit metoda `addTemperature`, která bude vyžadovat předání: teplotního údaje, senzoru (který tuto hodnotu naměřil) a sběru, ke kterému se tento údaj váže. Dále bude nabízet metody pro získávání teplotních dat různého druhu. Konkrétně metody sloužící pro získání poslední (aktuální) přečtené hodnoty ze senzoru, předchozí hodnoty, údajů za posledních 24 hodin, údajů za poslední týden, údajů za poslední rok, údajů za dobu definovanou uživatelem, průměrné teploty za určité období, nejvyšší a nejnižší teploty za určité období. Tyto metody pro získávání dat vždy vyžadují předání senzoru, ke kterému mají údaje patřit. Pouze některé metody vyžadují navíc předání data začátku a konce intervalu.

4.3.3 Služba ModuleService

Architektura systému je modulární. Jádro bude poskytovat základní služby. Nejdůležitější povinností bude získávání a uchovávání teplotních dat a zprostředkování těchto dat ostatním modulům, které s nimi budou dále pracovat. Modulem jsou EJB komponenty pro EJB „kontejner“. Každý modul bude muset být v systému zaregistrován vytvořením instance třídy `Module`. Je dobré vytvořit tzv. `Singleton`, který při startu systému zkontroluje přítomnost instance třídy `Module` a případně ji přidá.

Služba `ModuleService` bude poskytovat metody pro CRUD operace s moduly. Pro vytvoření instance třídy `Module` bude zapotřebí poskytnout metodě `createModule` základní informace o modulu jako: název, autor, verze a tag. Tag bude jedinečný identifikátor modulu, který bude moct sestávat pouze z písmen anglické abecedy a čísel. Služba `ModuleService` bude nabízet metodu `getByTag`, které se předá označení modulu, jenž chceme získat, a tato metoda nám jej vrátí (pokud existuje). Každý vytvořený modul bude pak mít své vlastní nastavení, záznamy a přiřazené senzory viz následující kapitola 4.3.5 Návrhový model tříd.

4.3.4 Služba RecordService

Jedná se o službu implementující systém záznamů do aplikace. Služba bude umožňovat vytváření různých záznamů, které budou moct být využity jako informativní („log“) záznamy pro uživatele nebo pro další zpracování různými moduly. Rozhraní bude definovat metody pro vytváření a získávání záznamů. Pro vytváření záznamů bude sloužit metoda `createRecord`. Metoda vyžaduje předání minimálně tří argumentů, mezi které patří: datum a čas záznamu, obsah záznamu a stav. Dalším volitelným parametrem je předání modulu nebo uživatele, ke kterému se záznam vztahuje.

Získávání záznamů se bude provádět pomocí jedné ze dvou metod, které rozhraní předepisuje. Každá metoda bude navíc třikrát „přetížená“ pro získávání záznamů týkajících se buďto pouze systému, uživatelů, anebo modulů. První metodou bude `getLastNRecordsWithValue`, která vyžaduje předání dvou argumentů: textu, jenž se má v záznamu vyskytovat, a počtu (kolik záznamů má metoda získat). Druhou metodou bude `getRecordsWithValueInInterval`, která bude místo počtu vyžadovat předání data a času začátku a konce intervalu, ze kterého se mají záznamy získávat. „Přetížené“ varianty těchto metod budou mít navíc buďto parametr typu `User` nebo `Module`.

4.3.5 Návrhový model tříd

První třídou modelu je třída `Gather` reprezentující sběr dat ze senzorů. Tato třída je ve vztahu s třídou `Temperature`. Bude obsahovat dva atributy s názvy: `id` a `gatherTime`. Atribut `id` bude celočíselného typu a jedinečným identifikátorem sběru. Atribut `gatherTime` bude datového typu `Date`, nesmí být `null` (má anotaci `@NotNull`) a představuje datum a čas provedeného sběru dat. Mohl by být jedinečným identifikátorem, ale na základě „best practice“ se zavádí umělý jedinečný identifikátor.

Další třídou je třída `Temperature` (představující teplotní údaj), která je ve vztahu s dalšími třídami `Gather` a `Sensor`. Jedinečným identifikátorem bude mít celočíselný atribut `id`. Samozřejmostí je atribut `temp` datového typu `float` obsahující samotnou hodnotu. Tento atribut bude mít dále tři validační anotace. První je anotace `@NotNull`, díky které se bude kontrolovat přítomnost hodnoty. Další anotací je `@Min` nastavující minimální hodnotu tohoto atributu na: -50. Poslední anotací je anotace

`@Max`, jenž bude naopak nastavovat maximální hodnotu na: 70. Těmito validačními anotacemi se vyfiltrují případné nesmyslné hodnoty.

Třída `Sensor` je ve vztahu s třídami `Temperature`, `GatheringUnit` a `ModuleSensor`. Bude obsahovat atribut `usc` datového typu `String`, který je jedinečným identifikátorem senzoru. Kromě něj bude obsahovat také umělé `id`. Dále každý senzor bude mít název, který bude v atributu `name` typu `String`. Maximální délka názvu bude 60 znaků (anotace `@Size` s parametrem `max`). Pro každý senzor bude možné vložit poznámku. Toho bude docíleno zavedením atributu `notes`, který bude datového typu `String`. Dalšími atributy třídy `Sensor` bude atribut: `installed` (datum a čas instalace senzoru – datový typ `Date`), `available` (dostupnost senzoru – datový typ `boolean`), `calibration` (kalibrace senzoru – typ `float`), `interval` (interval sběru dat – celočíselný typ `int`) a `lastGatherTime` (datum a čas posledního sběru ze senzoru – datový typ `Date`).

Sběrnou jednotku (zařízení s připojenými senzory) reprezentuje třída `GatheringUnit`. Tato třída je ve vztahu s třídou `Sensor` a `LoginDetails`. Bude obsahovat atributy: `id`, `name`, `ip`, `description`, `available` a `lastGatherTime`. Atribut `id` bude celočíselným umělým jedinečným identifikátorem. `name` bude pojmenováním sběrné jednotky se stejným pravidlem jako u třídy `Sensor`. `ip` bude atribut datového typu `String` o maximální délce 39 znaků, který bude obsahovat IP adresu sběrné jednotky. Atribut `description` bude obdobným atributem jako `notes` ve třídě `Sensor`. Zbývající atributy `available` a `lastGatherTime` budou stejné jako stejnojmenné atributy ve tříde `Sensor`.

Další třídou je třída `LoginDetails`. Jedná se o třídu obsahující atributy související s připojením ke sběrné jednotce. Kromě `id` bude obsahovat atributy: `username`, `password` a `protocol`. `username` bude přihlašovací jméno (datový typ `String` o maximální délce 40 znaků) a `password` bude pro změnu obsahovat přihlašovací heslo (typ `String` s maximální délkou 32 znaků). Atribut `protocol` bude typem `Protocol` (jedná se o „Enumerátor“ související s `GUConnectivity`) a bude obsahovat způsob připojení ke sběrné jednotce.

Kapitola 4.3.3 Služba ModuleService zmiňuje třídu `Module`. Jedná se o třídu, jejíž instance zastupují moduly aplikace. Bude obsahovat atributy: `id`, `name`, `enabled`,

author, tag, version a description. Třída má vztah s třídami ModuleSensor, Setting a Record. Atribut enabled bude datového typu boolean a bude představovat přepínač zapnutí/vypnutí modulu. author bude atributem typu String o maximální délce 40 znaků a bude sloužit pro zaznamenání jména autora modulu. Dalším atributem typu String o maximální délce 20 znaků bude tag. Více informací o tomto atributu obsahuje kapitola popisující službu související s touto třídou. version bude také datového typu String, ale o maximální délce 12 znaků, reprezentující verzi modulu. Zbývající atributy budou mít stejné vlastnosti jako stejnojmenné atributy v předešlých třídách. Pouze name bude o maximální délce jen 40 znaků.

Vazbu „M:N“ mezi třídami Module a Sensor rozvíjí ModuleSensor. Jedná se o třídu, která bude sloužit pro výběr senzorů využívaných v určitém modulu. Navíc bude obsahovat celočíselný atribut priority sloužící pro nastavení priority senzoru v souvislosti s konkrétním modulem.

Instance třídy Setting reprezentují nastavení týkající se buďto celkového systému nebo modulů. Vztah s třídou Module má multiplicitu „0..1“. Kromě umělého jedinečného identifikátoru id třída obsahuje atributy name a value. Atribut name je pojmenování konkrétního nastavení a nemusí být jedinečný. Bude datového typu String a maximální počet znaků, kterých bude moct nabývat, je 32. value je atribut, který bude obsahovat nastavenou hodnotu a stejně jako name, bude datového typu String (s maximální délkou 128 znaků).

Další třídou je třída Record, která byla již zmíněna v kapitole 4.3.4 Služba RecordService. Tato třída má vztah s třídami User a Module (s multiplicitou „0..1“) a obsahuje atributy: id, recordTime, status a value. Atribut recordTime bude datového typu Date a bude obsahovat datum a čas vytvoření záznamu. Atribut status bude typu boolean, který bude moct být využit jako doprovodný údaj k samotnému obsahu záznamu (např. obsahem záznamu bude, že došlo ke změně stavu vytápění a status bude obsahovat aktuální stav). value bude atribut datového typu String, který bude obsahovat samotný obsah záznamu.

Třída User reprezentuje uživatele systému a má vztah s třídami Record a Role. Obsahuje atributy datového typu String: username a password. username představuje přihlašovací jméno uživatele (s maximální délkou 40 znaků). Atribut

`password` bude obsahovat otisk hesla uživatele o délce 64 znaků pomocí „hešovací“ funkce SHA-256 (13).

Poslední třídou v modelu je třída `Role`. Tato třída je ve vztahu s třídou `User`. Reprezentuje uživatelské role. Obsahuje kromě umělého jedinečného identifikátoru `id` také atribut `name`, který představuje název role. `name` je datového typu `String` s maximální délkou 20 znaků.

4.4 Modul vytápění

Systém provádějící samotnou regulaci vytápění bude modulem aplikace. Modul bude sestávat ze dvou prvků. Jedním bude samotný EJB modul do EJB „kontejneru“. Druhým bude termoregulátor, který bude fungovat jako „stand-alone“ klient enterprise aplikace. Modul bude nabízet zejména služby: `HeatingDataService` a `HeatingControlService`. Dále bude nabízet službu `DayPeriodService`, která ovšem nebude v této práci detailněji rozebírána, protože bude pouze zprostředkovávat CRUD operace nad třídou `DayPeriod`.

Modul vytápění zavádí do modelu aplikace dvě nové třídy. Bude se jednat o třídy: `DayPeriod` a `SensorData`. Více informací o těchto třídách se lze dočíst v kapitole 4.4.3 Návrhový model tříd. Modul bude také obsahovat svůj `Singleton HeatingModule`, který při startu aplikace zkontroluje přítomnost modulu vytápění (instance třídy `Module`) a případně jej vytvoří. Dále zkontroluje, zda nastavení modulu obsahuje veškeré nutné položky. Mezi položky nastavení modulu vytápění bude patřit:

- `heatingEnabled` – představuje „vypínač“ vytápění (možné hodnoty jsou: `true/false`),
- `heatingHysteresisDifference` – hodnota diference kolem požadované hodnoty související s algoritmem regulátoru (jedná se o nezáporné reálné číslo),
- `heatingOutdoorTempLimit` – maximální hodnota venkovní teploty, při které zůstane vytápění zapnuté (očekává se reálné číslo),
- `heatingHeatNow` – určuje, zda se má momentálně vytápět bez ohledu na aktuální teplotní data (možné hodnoty jsou: `true/false`),

- `heatingHeatNowDuration` – doba, po kterou se má topit v souvislosti s předešlou položkou (celočíselná hodnota vyjadřující dobu v minutách),
- `heatingOutdoorSensorUSC` – identifikátor venkovního senzoru (řetězec o 15 znacích např. 28-0316b3bcddff).

4.4.1 Služba HeatingDataService

Jedná se o službu modulu vytápění, kterou bude využívat výhradně termoregulátor. Služba bude definována „remote“ rozhraním s anotací `@Remote`. To je velice důležité, protože na základě definované architektury, se klient (termoregulátor) může nacházet v jiné instanci JVM (na jiném zařízení) než aplikační server s aplikací. Klient bude muset ke službě přistupovat prostřednictvím tzv. *Remote Call* (vzdáleného volání) na „remote“ rozhraní pomocí RMI-IIOP (10, s. 235). Implementační třída bude mít přiřazeno mapování na globální JNDI název: „HeatingDataService“. Pod tímto názvem bude pak klient službu na aplikačním serveru vyhledávat.

Mezi metody, které bude služba nabízet, bude patřit šest různých metod. První metodou bude `getSettings`, jejíž návratovou hodnotou bude kolekce obsahující nastavení vytápění. Druhou metodou bude `getData`, která bude vracet „list“ s naposledy získanými daty ze senzorů. Data budou reprezentována třídou `SensorData`. Další metodou bude `getServerTime` vracející aktuální čas na aplikačním serveru jako datový typ `LocalTime`. Termoregulátor nebude muset mít pro správné fungování povinně nastaven čas. Postačí mít synchronizovaný čas pouze na aplikačním serveru. Denní doby s požadovanými teplotami bude možné získat pomocí metody `getDayPeriods`. Metoda bude vracet seznam denních dob reprezentovaných třídou `DayPeriod`. Poslední dvě metody budou sloužit opačným směrem pro předání informací z termoregulátoru směrem do aplikace. První z metod nazvaná `sendHeatingState` umožní informování o změně stavu vytápění (včetně vypočtené skutečné hodnoty). Druhá a poslední nabízená metoda (touto službou) bude `sendCalculatedAvgTemp`, která se bude využívat pro pravidelné předávání vypočtené hodnoty algoritmem regulátoru do aplikace.

4.4.2 Služba HeatingControlService

Služba `HeatingControlService` bude zprostředkovávat metody pro ovládání vytápění a metody pro získávání informací týkající se vytápění. Bude definována „local“ rozhraním, poněvadž využívána bude pouze v rámci aplikace z modulu webového rozhraní. Rozhraní bude definovat patnáct metod. Zde budou uvedeny pouze ty nejzajímavější.

Jednou z metod této služby bude `isCurrentlyHeatingOn`, jenž vrací logickou hodnotu znamenající aktuální stav vytápění (topení zapnuto/vypnuto). Dalšími metodami budou `getSettings` a `saveSettings`, které umožní načtení a případné uložení nastavení vytápění. Služba bude umět vydat poslední regulátorem vypočtenou teplotní hodnotu pomocí metody `getCurrentTemp`. Další metodou bude `calculateHeatingDuration`, která bude umět vypočítat uplynulou dobu od poslední změny stavu vytápění (doba, kterou se topí nebo naopak netopí). Poslední zde uvedenou metodou bude `heatNow`. Jedná se o metodu pro spouštění funkcionality *Topit nyní*. Tato funkcionalita umožní spustit vytápění bez ohledu na teplotní data. Metoda bude mít jediný parametr, a to celočíselnou hodnotu definující dobu topení.

4.4.3 Návrhový model tříd

Denní doby v podobě třídy `DayPeriod` budou první třídou, kterou zavádí modul vytápění. Bude obsahovat atributy: `id`, `startTime`, `endTime` a `temperature`. Celočíselný atribut `id` bude umělý jedinečný identifikátor. Atributy `startTime` a `endTime` budou datového typu `LocalTime`, určující začátek a konec intervalu denní doby. Posledním atributem bude `temperature` (s datovým typem `float`), který bude představovat požadovanou teplotu pro určitou denní dobu.

Další třídou bude `SensorData`, která bude reprezentovat data ze senzorů (včetně typu senzoru a priority), a její instance budou využívány pro předání údajů termoregulátoru. Mezi atributy bude patřit: `sensorType`, `value` a `priority`. Druh údaje (typ zdrojového senzoru) bude definovat atribut `sensorType`, který bude datového typu `Type` (jedná se o „Enumerátor“ ve stejné třídě s možnostmi „INDOOR“ a „OUTDOOR“). Atribut `value` bude datového typu `float` a bude obsahovat samotný teplotní údaj. Zbývající atribut `priority` bude (stejně jako ve třídě `ModuleSensor`) celočíselná hodnota určující prioritu.

Obě zmíněné třídy budou muset povinně implementovat rozhraní `Serializable`, protože jejich instance budou předávány pomocí RMI-IIOP. Další prvky, se kterými budou ve vztahu, již serializovatelné jsou (14, s. 568-569).

4.4.4 Klient termoregulátor

Software termoregulátoru bude obyčejnou Java SE aplikací, která bude umět fungovat ve dvou různých režimech nebo v kombinaci obou dvou. Prvním režimem bude čistě klientský režim („CLIENT_ONLY“), ve kterém regulátor nezbytně potřebuje pro své fungování připojení k aplikačnímu serveru, aby mohl využívat službu `HeatingDataService`. Druhým režimem bude regulátor jako samostatná aplikace („STAND_ALONE“), jenž bude načítat své nastavení z konfiguračního souboru, ve kterém bude mimo jiné definováno, jaký z připojených senzorů se má využít pro regulaci. Tento mód bude využíván hlavně v kombinované formě (ve variantě síť zařízení) jako záloha při výpadku spojení s aplikačním serverem. Kombinovaná forma („MIXED“) bude standardně fungovat v klientském režimu. Pouze při problému s připojením k serveru dojde k přepnutí do nezávislého režimu a po opětovném obnovení spojení k přepnutí zpět.

Termoregulátor se bude skládat z několika prvků. Hlavním bude třída `ThermoRegulator` obsahující metodu `main` a tvořící vstupní bod aplikace. Ta se postará o načtení a kontrolu konfigurace, inicializaci aplikace a spuštění algoritmu regulátoru. Dalším prvkem bude rozhraní `HeatingSwitch` reprezentující přepínač akčního členu (vytápění). Toto rozhraní bude definovat metody (`turnOn` a `turnOff`) pro přepínání mezi dvěma stavy a metody pro zjišťování aktuálního stavu (`isOn` a `isOff`). Implementací tohoto rozhraní bude třída `GpioHeatingSwitch` využívající pro přepínání rozhraní GPIO (na základě požadavků z analýzy). Zavedení rozhraní umožňuje jednoduché nahrazení jedné implementace za jinou a přepínání akčního členu i jiným způsobem. Dále bude existovat rozhraní `Sensor` s metodami `isPresent` a `getValue`, jehož implementací bude třída `LocalLinux1wSensor`, která umožní čtení ze senzoru připojeného přes „1-wire“ sběrnici pod operačním systémem GNU/Linux. Nejdůležitějším prvkem bude ovšem třída `ThermoRegulatorAlg` představující algoritmus regulátoru.

Algoritmus regulátoru bude podle režimu spouštěn buďto jako jednoduchý nebo rozšířený. Jednoduchá verze bude spočívat v pouhém přečtení hodnoty z lokálního senzoru, rozhodnutí o vytápění na základě algoritmu dvoubodového regulátoru (viz. kapitola 3.2 Regulace teploty) a aktuální konfiguraci. Rozšířená verze algoritmu bude využívat služby `HeatingDataService` modulu vytápění. Algoritmus si nejprve načte nejnovější nastavení a zkontroluje, jestli je vytápění zapnuto a není aktivována funkce *Topit nyní* (pokud ne, ověří patřičné přepnutí akčního členu a skončí cyklus). Pokračuje získáním veškerých potřebných dat, zjištěním požadované hodnoty (určením aktuální denní doby), výpočtem skutečné hodnoty, kontrolou venkovní teploty a následuje algoritmus dvoubodového regulátoru. Při rozšířeném algoritmu bude docházet k předávání vypočtené skutečné hodnoty a změny stavu do aplikace na aplikačním serveru.

4.5 Webové rozhraní

Webové rozhraní bude vytvořeno s využitím technologie JavaServer Faces (Java EE) jako další modul aplikace. Bude využita responzivní šablona webových stránek „Matrix Admin“, která v sobě již zahrnuje spoustu frontend technologií jako např. Bootstrap framework nebo jQuery (15). Mezi sekce rozhraní bude patřit úvodní stránka („dashboard“ zobrazující přehled nejdůležitějších údajů), část týkající se senzorů, správa sběrných jednotek a sekce umožňující spravovat modul vytápění.

4.5.1 Úvodní sekce

Mít rychlý přehled nad důležitými údaji bude umožňovat úvodní stránka webového rozhraní. Ta se bude skládat z několika informačních panelů („widgetů“). Prvním bude panel vytápění, který bude informovat o tom, zda je vytápění zapnuto, zda se aktuálně topí (a uplynulý čas), jaká je aktivní denní doba (včetně požadované teploty), jaká je vypočtená skutečná hodnota teploty v domě, zda je aktivována funkce *Topit nyní* a jaká je celková protopená doba za tento a předchozí den. Další panely budou poskytovat informace o senzorech (obdobně jako v sekci *Senzory*). Posledním panelem bude „widget“ obsahující graf, který bude zobrazovat vývoj teplot na určitém senzoru.

4.5.2 Sekce Senzory

Bude se jednat o pohled zobrazující veškeré senzory v systému. Každý senzor bude mít svůj rámeček, ve kterém se bude vyskytovat: název senzoru, aktuální naměřená hodnota, rozdíl (mezi průměrem za poslední hodinu a aktuální hodnotou), průměr za poslední hodinu, dostupnost (ano/ne), datum a čas posledního úspěšného sběru a tlačítko *Detail*, které uživatele přenese na zobrazení senzoru.

Zobrazení senzoru bude pohled, který bude obsahovat sekce: základní údaje, nastavení, teploty a graf. Také se bude v tomto zobrazení vyskytovat tlačítko pro přechod na editaci senzoru. Mezi základními údaji bude: id senzoru, název, usc (unique serial code), datum a čas přidání senzoru do systému, název sběrné jednotky (ke které je senzor připojen) jako odkaz pro přejití na zobrazení jednotky, dostupnost (ano/ne), datum a čas posledního sběru a poznámky. Nastavení bude obsahovat položky: kalibrace a interval. Sekce teploty bude obsahovat údaje: aktuální naměřená hodnota, rozdíl (mezi průměrem za poslední hodinu a aktuální hodnotou), průměr za poslední hodinu, nejvyšší naměřená hodnota během dvaceti čtyř hodin (a čas) a nejnižší naměřená hodnota během dvaceti čtyř hodin (a čas). Graf bude zobrazovat vývoj teplot během čtyřiadvaceti hodin, ale budou k dispozici i možnosti zobrazit vývoj teplot za týden, měsíc a rok, nebo si zvolit vlastní interval.

Editace senzoru bude sestávat z formuláře, který bude obsahovat pole pro změnu názvu senzoru a psaní poznámek k senzoru. Pod formulářem se bude nacházet tlačítko pro uložení změn.

4.5.3 Sekce Sběrné jednotky

Sekce sběrné jednotky bude obsahovat tabulku s výpisem veškerých sběrných jednotek v systému. Tabulka bude obsahovat sloupečky: *ID*, *Název*, *Počet senzorů*, *Dostupnost* (ano/ne), *Poslední sběr* (datum posledního úspěšného sběru) a *Akce*. Poslední sloupec tabulky *Akce* bude obsahovat tlačítka pro možnost zobrazení detailních informací o sběrné jednotce, tlačítko pro editaci údajů jednotky a tlačítko pro odebrání sběrné jednotky. Posledním prvkem v tomto zobrazení bude tlačítko pro přidání nové sběrné jednotky, které bude umístěné nad tabulkou obsahující výpis sběrných jednotek.

Formulář pro přidání nové sběrné jednotky bude obsahovat textová pole pro zadání názvu a IP adresy, menu pro výběr protokolu a další textová pole pro zadání uživatelského jména, hesla a popisu sběrné jednotky. V poslední řadě ještě tlačítko pro odeslání formuláře.

Zobrazení detailních informací o sběrné jednotce bude představovat výpis základních údajů: id, název a popis. Dále informace týkající se připojení: IP adresa, komunikační protokol a uživatelské jméno. V poslední řadě ještě počet připojených senzorů, informace o dostupnosti (ano/ne) a datum posledního úspěšného sběru. Nad těmito informacemi se budou nacházet dvě tlačítka. Jedno tlačítko bude sloužit pro editaci údajů sběrné jednotky a druhé tlačítko provede odebrání sběrné jednotky ze systému.

Formulář editace sběrné jednotky bude obdobou formuláře pro přidání nové sběrné jednotky, akorát bude sloužit (místo přidávání) k editaci již zadaných údajů.

4.5.4 Sekce Vytápění

Webové rozhraní modulu vytápění se bude skládat z několika částí. První částí bude *Panel* obsahující hlavní informace o vytápění jako: informace, zda se aktuálně topí (a jak dlouho se topí/netopí) a vypočtená skutečná teplotní hodnota. Současně bude také obsahovat tlačítko pro spuštění funkcionality *Topit nyní* včetně nabídky pro výběr doby topení. Další částí bude tabulka *Historie* obsahující záznamy o vytápění. Bude obsahovat sloupce: *Datum a čas*, *Událost* (spuštěno/vypnuto) a *Teplota* (na které došlo k přepnutí stavu).

Dále bude existovat tabulka denních dob, která bude obsahovat sloupce: *Čas začátku*, *Čas konce*, *Teplota* (požadovaná teplota v denní dobu) a *Akce*. Sloupec *Akce* bude obsahovat tlačítka pro editaci a smazání denní doby. Pod tabulkou se bude nacházet tlačítko pro přidání. Formuláře pro editaci a přidání denních dob budou obsahovat textová pole pro definování začátku a konce denní doby společně s požadovanou teplotou (pochopitelně včetně tlačítka pro odeslání). Čtvrtou částí bude tabulka senzorů, které byly vybrány pro zahrnutí ve výpočtech skutečné hodnoty. Tabulka bude obsahovat sloupečky: *Senzor* (název senzoru v podobě odkazu na detail senzoru), *Typ* (vnitřní nebo venkovní), *Priorita* a *Akce*. Sloupec *Akce* bude stejný jako u tabulky denních dob. Stejně tak bude pod tabulkou tlačítko pro přidání (výběr)

senzoru. Formulář pro výběr senzoru bude obsahovat nabídku dostupných senzorů, výběr typu senzoru a priority. Formulář pro editaci bude umožňovat pouze změnu typu senzoru a priority.

Poslední částí rozhraní bude formulář *Nastavení* obsahující přepínač zapnutí/vypnutí modulu vytápění a dvě textová pole, která budou sloužit pro definování diference a hranice venkovní teploty.

5 Implementace navrženého řešení

Tato kapitola se věnuje implementační části vývoje. Implementace proběhla na základě návrhu provedeného v předchozí části. Při implementaci se čerpalo převážně z knih „Beginning Java EE 7“ (10), „Učebnice jazyka Java“ (16) a „Java - Bohatství knihoven“ (12), dále z příručky pro vývoj aplikací pro aplikační server GlassFish (17) a z dokumentací Java SE (18) a Java EE (19). Při řešení konkrétních problémů byly dále využity různé internetové informační zdroje. Z důvodu velké obsáhlosti zde budou uvedeny pouze vybrané implementační problémy.

5.1 Jádro

Důležitým úkolem jádra, jak už bylo zmíněno, je sbírání dat ze senzorů připojených ke sběrným jednotkám. Podle návrhu aplikace má toto zajišťovat služba `GatheringService` s metodou `gatherDataFromGatheringUnits`. Tato metoda má anotaci `@Schedule`, aby byla pravidelně volána „kontejnerem“. Implementace viz. následující zdrojový kód.

```
@Schedule(minute = "*/5", hour = "*")
public void gatherDataFromGatheringUnits() {
	final Gather gather = new Gather(new Date());
	final List<GatheringUnit> gatheringUnits = gUnitService.getAll();
	for (final GatheringUnit gu : gatheringUnits) {
		final Map<String, Float> data = getDataFromGatheringUnit(gu);
		if (data != null) {
			processData(data, gather, gu);
			List<Sensor> sensors =
				sensorService.getAllWithoutThisLastGatherTime(
					gather.getGatherTime());
			for (Sensor s : sensors) {
				s.setAvailable(false);
				sensorService.updateSensor(s);
			}
			gu.setLastGatherTime(gather.getGatherTime());
			gu.setAvailable(true);
		}
		else {
			gu.setAvailable(false);
		}
		gUnitService.updateGatheringUnit(gu);
	}
}
```

Zdrojový kód 2: Sběr dat ze senzorů prostřednictvím sběrných jednotek

Ze zdrojového kódu lze vidět, že metoda nejprve vytváří novou instanci sběru (třídy `Gather`) s aktuálním datem a časem. Poté získá seznam sběrných jednotek, který postupně prochází, a pro každou položku zkouší získat data (k tomu slouží metoda `getDataFromGatheringUnit`). Jestliže kolekce `data` není žádná hodnota (není `null`), tak pomocí metody `processData` dojde k dalšímu zpracování dat. Následující zbytek kódu uvnitř podmínky nastavuje stav dostupnosti senzorů a sběrné jednotky. Jinak (v případě, že kolekce `data` je `null`) dojde k označení sběrné jednotky za nedostupnou. V obou případech dojde k aktualizování sběrné jednotky.

Zpracování dat pomocí metody `processData` představuje iteraci přes kolekci dat a pro každý prvek se provede sled operací. Nejprve se ověřuje přítomnost senzoru v systému. Pokud senzor s určitým kódem neexistuje, pak je automaticky přidán. Dále se kontroluje, zda senzor, který se již v systému vyskytuje, nebyl náhodou přepojen k jiné sběrné jednotce. Pokud ano, změna se automaticky uloží do systému. Následně dojde k přidání teplotního údaje metodou `addTemperature`, kterou nabízí služba `TemperatureService`. Závěrem jsou ještě aktualizovány údaje o dostupnosti senzoru.

Metoda `getDataFromGatheringUnit` v podstatě jen vybírá správnou implementaci `GUConnectivity` podle typu připojení ke sběrné jednotce. V této práci byla implementována pouze metoda připojení pomocí SSH. Implementační třída se nazývá `GUConnectivitySSH` a její implementace metody `gatherTemperatures` lze vidět na následující ukázce zdrojového kódu.

```
@Override
public Map<String, Float> gatherTemperatures() throws IOException {
	final Map<String, Float> sensorsAndValues = new HashMap<>();
	final Set<String> sensors = getSensorsSet();
	for (final String usc : sensors) {
		try {
			String value = getSensorValue(usc);
			if (value != null) {
				float temp = Float.parseFloat(value);
				temp = temp / 1000;
				sensorsAndValues.put(usc, temp);
			}
		} catch (CRCCheckFailedException | NumberFormatException e) {
			LOGGER.log(Level.WARNING, e.getLocalizedMessage(), e);
		}
	}
	return sensorsAndValues;
}
```

Zdrojový kód 3: Získávání teplotních údajů

Uvnitř metody dochází k vytváření kolekce senzorů a jejich hodnot. Kolekce je typu `Map`, která obsahuje dvojice: klíč jako unikátní sériový kód senzoru a hodnotu jako naměřený teplotní údaj. Následně se získává množina dostupných senzorů pomocí metody `getSensorsSet` (viz. následující zdrojový kód). Tato množina je procházena a pro každý senzor se zkouší přečíst jeho hodnota voláním metody `getSensorValue`. Pokud je hodnota přítomna, zkusí se převést na reálné číslo a následně i na jednotku stupeň Celsia (formát výstupu ze senzoru jsou milistupně). Jestliže se vše podaří, je kombinace kód senzoru a hodnota vložena do kolekce. Během této části může dojít ke dvěma výjimkám. První je selhání při čtení hodnoty ze senzoru. K ověření bezchybnosti nějakého bloku dat se využívá tzv. cyklický redundantní součet (20). Další výjimkou může být selhání při převodu z řetězce na číslo.

```
private Set<String> getSensorsSet() throws IOException,
                        ConnectionException, TransportException {
       final Set<String> sensors = new HashSet<>();
       final Session sess = ssh.startSession();
       final Command cmd = sess.exec("ls " + W1BUSPATH);
       final String w1 = IOUtils.readFully(cmd.getInputStream()).toString();
       sess.close();
       String[] devices = w1.split("\n");
       for (int i = 0; i < devices.length; i++) {
             if (devices[i].length() == USCLENGTH &&
                 devices[i].startsWith(PREFIX)) {
                   sensors.add(devices[i]);
             }
       }
       return sensors;
}
```

Zdrojový kód 4: Sestavení množiny dostupných senzorů

Zdrojový kód 4 obsahuje metodu pro sestavování množiny dostupných senzorů `getSensorsSet`. Nejprve je vytvořena prázdná množina senzorů. Následně se vytváří nová relace nad nějakým objektem `ssh`. Tento objekt je instancí třídy `SSHClient` z externí knihovny SSHJ, která reprezentuje SSH klienta. SSHJ je SSH Java knihovna implementující verzi SSHv2 (21). Nad otevřenou relací se spouští příkaz, který vrátí seznam senzorů, a jeho výstup se zapíše do proměnné „w1“. Po dokončení se relace uzavře a následuje zpracování výstupu a kontrola. Do množiny senzorů se přidávají pouze kódy zařízení, které jsou teplotním senzorem DS18B20. Probíhá kontrola ověřující délku kódu a přítomnost specifického prefixu.

Následující ukázka zdrojového kódu obsahuje metodu pro čtení hodnot z konkrétních senzorů. Metoda vyžaduje předání kódu senzoru, ze kterého se má číst. Uvnitř probíhá vytvoření relace, spuštění příkazu pro přečtení hodnoty přes rozhraní 1-Wire zařízení, načtení výstupu do proměnné `fullOutput` a závěrečné uzavření relace. V surovém výstupu se hledá řetězec „YES“, který znamená, že ověření kontrolního součtu dopadlo úspěšně. Pokud se nepodaří řetězec nalézt, tak se přečtení hodnoty opakuje a to tolikrát, na kolik je nastaven počet opakování v konstantní proměnné `READRETRYCOUNT`. Pokud ani po opakovaném čtení nedopadlo ověření kontrolního součtu úspěšně, tak se končí výjimkou `CRCCheckFailedException`. Jinak metoda vrací vše, co se nachází za řetězcem „t=“ (bez netisknutelných znaků), protože tam se vyskytuje už jen přečtená hodnota a nic dalšího.

```
private String getSensorValue(final String usc) throws IOException,
                              ConnectionException, TransportException,
                              CRCCheckFailedException {
        String fullOutput = "";
        for (int i = 0; i < READRETRYCOUNT &&
             fullOutput.indexOf("YES") == -1; i++) {
              final Session sess = ssh.startSession();
              final Command cmd = sess.exec("cat " + W1BUSPATH + "/"
                              + usc + W1READINTERFACE);
              fullOutput = IOUtils.readFully(cmd.getInputStream()).toString();
              sess.close();
        }

        if (fullOutput.indexOf("YES") == -1) {
              throw new CRCCheckFailedException(usc, fullOutput);
        }

        return fullOutput.split("t=")[1].trim();
}
```

Zdrojový kód 5: Přečtení hodnoty ze senzoru

Manipulaci s teplotními daty zajišťuje služba `TemperatureService`. Pro přidání teplotního údaje přečteného ze senzoru slouží metoda `addTemperature`, jejíž implementaci lze vidět ve zdrojovém kódu níže. Vytváří se nová instance třídy `Temperature`. Následuje kontrola všech omezení. Jestliže nejsou nalezena žádná porušení, dojde k „perzistování“ nové teploty. Jinak se do logu uloží záznam o překročení nějakého omezení a k „perzistování“ nedojde. Podstatný je objekt `em`, který je instancí třídy `EntityManager`. Jedná se o centrální prvek, který v JPA zodpovídá za organizování veškerých entit (10, s. 108).

```
@Override
public Temperature addTemperature(float temp, final Sensor sensor,
                                  final Gather gather) {
	final Temperature t = new Temperature(temp, gather, sensor);
	final Set<ConstraintViolation<Temperature>> violations =
		validator.validate(t);

	if (violations.isEmpty()) {
		em.persist(t);
	}
	else {
		LOGGER.log(Level.WARNING, "Constraint violation during
			addTemperature. " + violations.toString());
	}

	return t;
}
```

Zdrojový kód 6: Přidání nového teplotního údaje

5.2 Modul vytápění

Důležitým prvkem modulu je služba `HeatingDataService`, která nabízí své služby termoregulátoru a ten je využívá. Velice podstatnou metodou této služby je `getData`, jenž slouží pro získání aktuálních dat získaných z vybraných senzorů za účelem regulace. Zdrojový kód 7 obsahuje ukázku této metody.

```
@Override
public List<SensorData> getData() {
	final List<SensorData> data = new ArrayList<>();
	final Module module = heatingModule.getThisModule();
	if (module != null) {
		final Set<String> outdoorSensors = getOutdoorSensorsUSCs(module);
		for (ModuleSensor ms : module.getModuleSensors()) {
			data.add(new SensorData(
				outdoorSensors.contains(ms.getSensor().getUsc())?
					SensorData.Type.OUTDOOR:
					SensorData.Type.INDOOR,
				ms.getPriority(),
				temperatureService.getCurrentTemperature(
					ms.getSensor()).getTemp()
				)
			);
		}
	}

	return data;
}
```

Zdrojový kód 7: Získání dat pro regulaci

Uvnitř metody dochází nejprve k získání modulu vytápění. Pokud je získání úspěšné, jsou následně procházeny jeho senzory a pro každý senzor je vytvářena

instance `SensorData`, které se v konstruktoru předá typ senzoru, jeho priorita a poslední naměřený údaj.

Další zajímavou metodou, jejíž zdrojový kód je možné vidět níže, je `sendHeatingState`, která provádí zaznamenání aktuálního změněného stavu vytápění včetně vypočtené teploty, na které došlo k této změně. Pro samotné vytvoření záznamu se využívá služba `RecordService`.

```
@Override
public void sendHeatingState(boolean isHeatingOn, float temp) {
	final Module module = heatingModule.getThisModule();
	if (module != null) {
		recordService.createRecord(new Date(), "Heating state changed at
					temp: " + temp, isHeatingOn, module);
	}
}
```

Zdrojový kód 8: Zaznamenání aktuálního změněného stavu vytápění

Mezi nabízené funkcionality služby `HeatingControlService` patří metoda `isCurrentlyHeatingOn`, která vrací logickou hodnotu podle toho, zda je momentálně vytápění zapnuto či vypnuto. K docílení požadované funkcionality probíhá uvnitř metody získání posledního záznamu modulu vytápění, který obsahuje informaci o změně stavu. Pokud byl takový záznam nalezen, pak se vrací jeho přidružená informace o stavu. Jinak se vrací nepravda, protože neexistuje žádný záznam o změně stavu, tudíž vytápění ještě nebylo pravděpodobně zapnuto.

```
@Override
public boolean isCurrentlyHeatingOn() {
	final List<Record> records =
		recordService.getLastNModuleRecordsWithValue(
			getHeatingModule(), "Heating state changed", 1);

	if (!records.isEmpty()) {
		return records.get(0).isStatus();
	}

	return false;
}
```

Zdrojový kód 9: Získání aktuálního stavu vytápění

5.3 Termoregulátor

Termoregulátor má podle návrhu umět pracovat ve dvou režimech. Zdrojový kód 10 obsahuje ukázku přepínání mezi těmito režimy. V čistě klientském režimu

dochází k volání metody `runAsClient` a v režimu samostatné aplikace zase metody `runStandAlone`. Kombinovaná forma využívá pro přepínání mezi stavy logické proměnné `connectionFailed`. V případě výpadku spojení s aplikačním serverem se tato proměnná nastaví na hodnotu `true` a to zapříčiní volání metody `runStandAlone`. Po opětovném úspěšném připojení k serveru se `connectionFailed` nastaví na hodnotu `false` a dolní podmínka opět přestane platit.

```
@Override
public void run() {
	if (conf.getMode()==ThermoRegulatorMode.MIXED||
	    conf.getMode()==ThermoRegulatorMode.CLIENT_ONLY) {
	      try {
			runAsClient();
			if (connectionFailed) {
				connectionFailed = false;
				LOGGER.log(Level.INFO, "Successfully connected to
					application server: "+conf.getServerIp());
			}
	      } catch(ConnectionFailedException e) {
			LOGGER.log(Level.WARNING, e.getMessage(), e);
			if (!connectionFailed) {
				connectionFailed = true;
				return;
			}
	      }
	}
	if (conf.getMode()==ThermoRegulatorMode.MIXED && connectionFailed||
	    conf.getMode()==ThermoRegulatorMode.STAND_ALONE) {
	      runStandAlone();
	}
}
```

Zdrojový kód 10: Přepínání mezi režimy termoregulátoru

Nejpodstatnější částí termoregulátoru je samotný jeho algoritmus. Zdrojový kód 11 obsahuje ukázku nejdůležitějších částí. Nejprve jsou na začátku zpracována data ze senzorů výpočtem skutečné hodnoty z údajů vnitřních senzorů (data venkovních senzorů jsou odkládána do vedlejšího seznamu pro pozdější vyhodnocení). Výpočet jediné skutečné hodnoty je prováděn pomocí váženého aritmetického průměru (jak bylo naznačeno v kapitole 3.2 Regulace teploty). Váhami jsou ve výpočtu priority senzorů, které se sčítají do proměnné `sum`, aby bylo možné jimi později vydělit. Pokud je součet priorit roven nule, tak dochází k výpočtu průměru s váhami rovno jedné. Vypočtená skutečná hodnota je následně předána aplikaci na aplikačním serveru.

Poté dochází ke kontrole, zda údaje z venkovních senzorů nepřekračují nastavenou hranici (maximální venkovní teplotu). Tím končí část předzpracování a následuje výpočet regulační odchylky (porovnávacím členem). Vypočtená odchylka společně s dalšími hodnotami je využita v poslední části algoritmu, kterou je ústřední člen dvoubodového regulátoru. Akční veličinu v tomto případě reprezentuje objekt `heating`, který je typu `HeatingSwitch`. Objekt `conf` je instance třídy `Configuration`, který obsahuje aktuální konfiguraci termoregulátoru. Při spouštění aplikace je načítána základní konfigurace ze souboru a při běhu v klientském režimu se aktualizuje aktuálním nastavením z aplikačního serveru.

```
for (final SensorData sensor : data) {
    if (sensor.getSensorType() == Type.OUTDOOR) {
       outdoorSensors.add(sensor);
       continue;
    }
    weightedAvgTemp += sensor.getPriority() * sensor.getValue();
    sum += sensor.getPriority();
}

if (sum == 0) {//ošetření proti dělení nulou}
weightedAvgTemp /= sum;
dataService.sendCalculatedAvgTemp((float)weightedAvgTemp);

if (!outdoorSensors.isEmpty() &&
         checkOutdoorTempExceeding(outdoorSensors, outdoorTempLimit)) {
    if (heating.isOn()) {
        heating.turnOff();
        dataService.sendHeatingState(false, (float)weightedAvgTemp);
    }
}

float distance = Math.abs((float)(requestedTemp - weightedAvgTemp));

if (distance >= conf.getDifference() / 2) {
    if (weightedAvgTemp < requestedTemp) {
       if (heating.isOff()) {
           heating.turnOn();
           dataService.sendHeatingState(true, (float)weightedAvgTemp);
       }
    } else {
       if (heating.isOn()) {
           heating.turnOff();
           dataService.sendHeatingState(false, (float)weightedAvgTemp);
       }
    }
}
```

Zdrojový kód 11: Ukázka algoritmu termoregulátoru

Regulátor obsahuje jedinou implementaci `HeatingSwitch`, a to pomocí rozhraní GPIO. Ukázku implementace metody `turnOn` je možné vidět na zdrojovém kódu 12. Metoda pouze nastavuje výstupní pin (objekt typu `GpioPinDigitalOutput`) na vysokou nebo nízkou hodnotu v závislosti na konfiguraci, která hodnota představuje sepnutí vytápění. Pro různá zařízení může hodnota na výstupu znamenat něco jiného např. relé v zapojení NO (Normally Opened) vyžaduje pro sepnutí nízkou LOW hodnotu (logickou nulu). Obdobně funguje opačná metoda `turnOff` (22).

```
@Override
public void turnOn() {
	if (isLowEqOn) {
		pin.low();
		LOGGER.log(Level.INFO, "Pin state LOW");
	}
	else {
		pin.high();
		LOGGER.log(Level.INFO, "Pin state HIGH");
	}
}
```

Zdrojový kód 12: Implementace sepnutí vytápění pomocí GPIO rozhraní

Získávání dat z aplikace na aplikačním serveru termoregulátor provádí s využitím služby `HeatingDataService`. Tuto službu je nutné nejprve získat. Způsob se mírně liší podle toho, na kterém aplikačním serveru aplikace poběží (v tomto případě GlassFish). Nejprve se získá aktuální kontext (v tomto případě aplikačního serveru). Následně se služba vyhledá podle JNDI názvu. Jelikož má tento název explicitně nastaven na „HeatingDataService“ pomocí anotace, pro vyhledání stačí předat název služby. Zkontroluje se, zda získaný objekt není `null` a pomocí metody `narrow` třídy `PortableRemoteObject`, jestli je možné ho „přetypovat“ na *remote* rozhraní. Metoda vrací referenci na *remote* rozhraní služby nebo `null`, pokud nastal nějaký problém s připojením k aplikačnímu serveru nebo vyhledáním služby. Před spuštěním je nutné přidat GlassFish knihovny (zejména soubor „gf-client.jar“) do „classpath“ a nastavit „hostname“ a ORB port (17, s. 189-190).

```
@Override
public HeatingDataServiceRemote getHeatingDataService() {
	InitialContext ctx;
	try {
		ctx = new InitialContext();
		Object obj = ctx.lookup("HeatingDataService");

		if (obj != null) {
			return (HeatingDataServiceRemote)
				PortableRemoteObject.narrow(obj,
					HeatingDataServiceRemote.class);
		}
	} catch (NamingException e) {
		LOGGER.log(Level.WARNING, e.getLocalizedMessage(), e);
	}

	return null;
}
```

Zdrojový kód 13: Přistupování ke službě HeatingDataService

5.4 Webové rozhraní

JavaServer Faces je webový framework postavený na architektuře MVC (Model-View-Controller). Webové rozhraní se skládá z pohledů v podobě XHTML souborů v adresáři WebContent, JSF kontrolérů („Managed Beans“) a šablony webových stránek (v adresáři `WebContent/template`). Složku modelu v architektuře MVC obsahují EJB moduly aplikace. Při tvorbě webového rozhraní se navíc čerpaly informace z průvodce JSF „JavaServer Faces (JSF) Tutorial“ (23), dokumentace pro Bootstrap (24), průvodce k jazyku JavaScript na serveru www.w3schools.com (25) a dokumentace k jQuery knihovně pro vykreslování grafů Flot (26).

Nezbytnou funkcionalitou je schopnost zobrazit naměřený údaj. Konkrétně získat a zobrazit aktuální (poslední naměřenou) teplotu ze senzoru. Pro zobrazení (vypsání) hodnoty určité proměnné se v JSF používá tag `h:outputText`. V atributu `value` se pomocí tzv. *Expression Language* uvede výraz, jehož výsledek se má zobrazit. Výstup lze dále formátovat pomocí formátovacích značek. Zdrojový kód 14 obsahuje ukázku zobrazení aktuální teploty ze senzoru ve formátu čísla s jedním desetinným místem, za kterým následuje jednotka stupeň Celsia. Pro získání údaje k zobrazení má kontrolér funkci `getCurrentTemp`. Na zdrojovém kódu 15 je možné vidět, že dochází k využití metody `getCurrentTemperature` služby `TemperatureService`. Metoda vrací referenci na objekt typu `Temperature`, proto následuje ještě volání `getTemp`, které již vrací samotnou teplotní hodnotu jako reálné číslo.

```
<h:outputText
        value="#{sensorController.getCurrentTemp(sensorController.sensor)}"
        style="font-weight: bold;font-size: 14px">
          <f:convertNumber pattern="#0.0°C" />
</h:outputText>
```

Zdrojový kód 14: Zobrazení poslední naměřené hodnoty senzorem

```
public float getCurrentTemp(Sensor sensor) {
      return tempService.getCurrentTemperature(sensor).getTemp();
}
```

Zdrojový kód 15: Získání poslední naměřené hodnoty

Podle návrhu má sekce vytápění obsahovat tabulku se záznamy o vytápění. Implementaci obsahuje zdrojový kód 16. Bylo využito tagu `h:dataTable`, který umožňuje iterování přes kolekci a vykreslování v podobě tabulky. Sloupce tabulky definují párové značky `h:column`, mezi které může přijít definice názvu sloupce pomocí tagu `f:facet` a zvolený obsah, který se má v buňkách sloupce zobrazovat např. výpis libovolného textu nebo zobrazení teploty.

```
<h:dataTable value="#{heatingController.records}" var="r"
        styleClass="table table-bordered">
      <h:column>
            <f:facet name="header">Datum a čas</f:facet>
            <h:outputText value="#{r.recordtime}">
                  <f:convertDateTime
                      pattern="#{mainController.datetimeFormat}" />
            </h:outputText>
      </h:column>
      <h:column>
            <f:facet name="header">Událost</f:facet>
            <h:outputText value="#{r.status?'Spuštěno':'Vypnuto'}" />
      </h:column>
      <h:column>
            <f:facet name="header">Teplota</f:facet>
            <h:outputText
               value="#{heatingController.getTempFromRecordValue(r.value)}">
                  <f:convertNumber pattern="#0.0°C" />
            </h:outputText>
      </h:column>
</h:dataTable>
```

Zdrojový kód 16: Zobrazení historie vytápění v tabulce

Dalším zajímavým implementačním problémem je vykreslování grafů. Zobrazování grafů je prováděno pomocí JavaScript JQuery knihovny Flot. Vykreslování probíhá dovnitř definovaného „div“ bloku. Zdrojový kód 17 obsahuje ukázku takového

bloku, který je označen třídou `chart` a navíc má definovanou datovou vlastnost `points` sloužící pro předání zobrazovaných dat.

```
<div class="chart" data-points="#{
    temperatureDataConverter.getAsJsonArray(
      graphController.getData(sensorController.sensor))}">
</div>
```

Zdrojový kód 17: Blokový element pro vykreslení grafu

Vykreslení grafu uvnitř blokového elementu `<div></div>` provádí následující zdrojový kód 18. Volá se funkce plot, které se předá: objekt (element, uvnitř kterého se má graf vykreslit), data jako dvourozměrné pole čísel (parametr `color` definuje barvu křivky) a nastavení (`mode` představuje typ osy (může být číselná nebo časová) a `tickFormatter` je funkce pro definování formátu popisků osy).

```
var plot = $.plot(obj,
    [ { data: JSON.parse(obj.dataset.points), color: "#ee7951"} ], {
      series: {
            lines: { show: true },
            points: { show: true }
      },
      xaxis: {
            mode: "time"
      },
      yaxis: {
            tickFormatter: function(num, obj){return num+"°C"}
      },
      grid: { hoverable: true, clickable: true }
});
```

Zdrojový kód 18: Vykreslení grafu pomocí JavaScriptu

Požadavkem vycházejícím z návrhu byla možnost zobrazit vývoj teplot za den, týden, měsíc a rok, nebo si zvolit vlastní interval. To bylo splněno implementováním formuláře, jehož zdrojový kód je možné vidět na zdrojovém kódu 19. Formulář obsahuje výběr jedné možnosti (tag `h:selectOneRadio`) z předdefinovaných intervalů nebo je možné zvolit variantu vlastní. Pro možnost vlastní jsou k dispozici dvě textová pole (pomocí značky `h:inputText`), do kterých se zadává datum a čas vymezující určité období. Dále je ještě k dispozici tlačítko *Zobrazit* (tag `h:commandButton`) pro překreslení grafu.

```
<h:form>
    <h:selectOneRadio value="#{graphController.interval}"
      layout="pageDirection">
      <f:selectItem itemValue="1" itemLabel="24 hodin" />
      <f:selectItem itemValue="2" itemLabel="týden" />
      <f:selectItem itemValue="3" itemLabel="měsíc" />
      <f:selectItem itemValue="4" itemLabel="rok" />
      <f:selectItem itemValue="5" itemLabel="vlastní" />
    </h:selectOneRadio>
    Od <h:inputText value="#{graphController.from}">
      <f:convertDateTime pattern="#{mainController.datetimeFormat}" />
    </h:inputText>
    Do <h:inputText value="#{graphController.to}">
      <f:convertDateTime pattern="#{mainController.datetimeFormat}" />
    </h:inputText>
    <h:commandButton action="#{graphController.display}" value="Zobrazit">
      <f:param name="id" value="#{sensorController.sensor.id}" />
    </h:commandButton>
</h:form>
```

Zdrojový kód 19: Formulář pro výběr dat zobrazených grafem

6 Shrnutí výsledků

Cílem této bakalářské práce bylo navrhnout a implementovat Java EE aplikaci pro regulaci teploty v domě s využitím dat ze senzorů připojených k Raspberry Pi (nebo obdobného zařízení) včetně nějakého způsobu ovládání. Během analýzy byla zvolena varianta ovládání pomocí webového rozhraní.

Volba technologií a návrh systému proběhl na základě požadavků vzniklých z analýzy tohoto problému. Následně byla navržena architektura s ohledem na zvolenou technologii. Systém se skládá ze čtyř modulů: modulu jádra, webového rozhraní, modulu vytápění a termoregulátoru.

Modul jádra se stará o sběr dat ze senzorů a poskytuje základní služby ostatním modulům (např. zprostředkování získaných dat). Účel webového rozhraní je, myslím, zřejmý – poskytovat přehled a umožnit uživateli regulaci ovlivňovat. Součástí je i výstup v podobě grafů. Modul vytápění zprostředkovává předzpracovaná data termoregulátoru, poskytuje metody pro získávání různých informací o vytápění a umožňuje ovládání regulace prostřednictvím modulu webového rozhraní. Termoregulátor je klientem provádějící samotnou regulaci na principu dvoubodového regulátoru s využitím dat a nastavení, která mu poskytuje aplikace umístěná na aplikačním serveru. Současně umí pracovat i v samostatném režimu např. pro případ výpadku spojení k serveru.

Nakonec byl takto navržený systém i realizován. Implementace proběhla v prostředí Eclipse a výsledná aplikace je funkční, ovšem z důvodu nedostatečného času pro důkladné otestování nelze zaručit naprostou bezchybnost.

7 Závěr a doporučení

Zvolená architektura umožňuje provozovat systém ve dvou různých variantách: varianta jediného zařízení nebo síť zařízení. Nasazení ve vlastním domě proběhlo ve variantě druhé. Řešení spočívalo ve třech sběrných jednotkách (dvě zařízení Orange Pi a jedno Raspberry Pi, které současně vystupuje v roli regulátoru) a jednom serveru (vyhrazený počítač s nainstalovaným aplikačním serverem GlassFish a databázovým systémem MySQL). První jednotka umístěná do zahradního domku komunikuje bezdrátově prostřednictvím WiFi a disponuje připojenými senzory měřící teplotu vody v bazénu (samozřejmě není využito pro regulaci) a venkovní teploty vzduchu na slunci a ve stínu. Zbývající dvě jednotky jsou nainstalovány v kotelně (nacházející se ve sklepě) a v prvním patře domu. Celkem je k systému připojeno dvanáct teplotních snímačů.

Pro zprovoznění aplikace je nezbytné provést několik nastavení v administraci aplikačního serveru. Nejprve nadefinovat JDBC Resource tak, aby korespondoval s konfigurací samotné aplikace. To je důležité z důvodu, aby měla aplikace přístup do databáze. Dále je potřeba zavést v části Security nový JDBC Realm, který bude umožňovat přihlášení do aplikace. Pokud se termoregulátor nebude nacházet na stejném zařízení společně s aplikačním serverem, je potřeba také správně nastavit IIOP Listener. Termoregulátor je nutné nakonfigurovat v souboru „config.properties“. Pokud má fungovat i v samostatném režimu, je vyžadováno zadání kódu senzoru, který se má používat.

Při tvorbě aplikace byla snaha o navržení systému tak, aby jej bylo možné jednoduše rozšířit. Je možné přidávat další moduly, které budou různě pracovat s daty získávané jádrem. Pouhým vytvořením alternativní implementace `GUConnectivity` je možné začít získávat data i jiným způsobem než pomocí SSH. Implementováním rozhraní `HeatingSwitch` vlastní variantou, lze dosáhnout spínání i jiným způsobem než jen přes rozhraní GPIO. Zajímavým novým modulem by byl analyzátor místností, který by informoval uživatele o přetápění nebo naopak o „nedotápění“, a tím by napomáhal dosažení rovnoměrného vytápění celého domu.

8 Seznam použité literatury

[1] DOUBRAVA, Jiří a kolektiv. *Regulace ve vytápění*. 2., upr. vyd. Praha: Společnost pro techniku prostředí, 2007. Sešit projektanta - pracovní podklady. 181 str. ISBN 978-80-02-01951-0.

[2] BARTEL, Stanislav. Automatické regulace vytápění. In: *TZB-info* [online]. Praha: Topinfo, 2004 [cit. 2017-06-29]. Dostupné z: http://vytapeni.tzb-info.cz/mereni-a-regulace/2079-automaticke-regulace-vytapeni

[3] BAŠTA, Jiří. *Hydraulika a řízení otopných soustav*. Praha: Vydavatelství ČVUT, 2003. 252 str. ISBN 80-01-02808-9.

[4] Snímače. In: *Aplik.sk* [online]. Bratislava: APLIK spol., c2000-2017 [cit. 2017-07-10]. Dostupné z: http://www.aplik.sk/sk/Produkty/Snimace.alej

[5] *DS18B20 Programmable Resolution 1-Wire Digital Thermometer* [online]. Rev 4. USA: Maxim Integrated, c2015 [cit. 2018-01-19]. Dostupné z: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

[6] HOMB, Gunnar Ranøyen. Do It Yourself: SNMP Temperature Monitoring System. In: *Norwegian Creations* [online]. Norway: Norwegian Creations, 2017 [cit. 2017-07-27]. Dostupné z: https://www.norwegiancreations.com/2017/06/do-it-yourself-snmp-temperature-monitoring-system

[7] GPIO: Raspberry Pi Models A and B. *Raspberry Pi* [online]. Cambridge: Raspberry Pi Foundation [cit. 2018-01-29]. Dostupné z: https://www.raspberrypi.org/documentation/usage/gpio/

[8] Java Platform, Enterprise Edition. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-01-30]. Dostupné z: https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition

[9] MySQL. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-01-30]. Dostupné z: https://cs.wikipedia.org/wiki/MySQL

[10] GONCALVES, Antonio. *Beginning Java EE 7*. Berkeley, CA: Apress, 2013. ISBN 978-1-4302-4626-8.

[11] SSH (Secure Shell). In: *SSH Communications Security* [online]. Waltham: SSH Communications Security, 2017 [cit. 2018-01-30]. Dostupné z: https://www.ssh.com/ssh/

[12] HEROUT, Pavel. *Java - bohatství knihoven*. 3. vyd. České Budějovice: Kopp, 2008. ISBN 978-80-7232-368-5.

[13] SHA-2. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-03-13]. Dostupné z: https://en.wikipedia.org/wiki/SHA-2

[14] SRIGANESH, Rima Patel., Gerald. BROSE a Micah. SILVERMAN. *Mastering enterprise JavaBeans 3.0*. Indianapolis, IN: Wiley Pub., c2006. ISBN 978-0-471-78541-5.

[15] Free bootstrap admin template - Matrix admin. *Wrappixel* [online]. 2016 [cit. 2018-03-17]. Dostupné z: https://wrappixel.com/templates/matrix-admin/

[16] HEROUT, Pavel. *Učebnice jazyka Java*. 5., rozš. vyd. České Budějovice: Kopp, 2010. ISBN 978-80-7232-398-2.

[17] *GlassFish Server Open Source Edition: Application Development Guide* [online]. Release 4.0. Redwood City, CA: Oracle Corporation, 2013 [cit. 2018-03-26]. Dostupné z: https://javaee.github.io/glassfish/doc/4.0/application-development-guide.pdf

[18] Java™ Platform, Standard Edition 7 API Specification. *Overview (Java Platform SE 7)* [online]. Redwood City, CA: Oracle Corporation, 2017 [cit. 2018-03-26]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/overview-summary.html

[19] Java(TM) EE 7 Specification APIs. *Overview (Java(TM) EE 7 Specification APIs)* [online]. Redwood City, CA: Oracle Corporation, 2015 [cit. 2018-03-26]. Dostupné z: https://docs.oracle.com/javaee/7/api/overview-summary.html

[20] MESANDER, Ben. Understanding the Cyclic Redundancy Check. In: *Cardinal Peak* [online]. Lafayette, CO: Cardinal Peak, 2013 [cit. 2018-03-29]. Dostupné z: https://cardinalpeak.com/blog/understanding-the-cyclic-redundancy-check/

[21] ERP, Jeroen. SSHJ - SSHv2 library for Java. In: *GitHub* [online]. San Francisco, CA: GitHub, 2017 [cit. 2018-03-29]. Dostupné z: https://github.com/hierynomus/sshj

[22] Simple GPIO Control using Pi4J. In: *The Pi4J Project* [online]. 2016 [cit. 2018-04-04]. Dostupné z: http://pi4j.com/example/control.html

[23] JavaServer Faces (JSF) Tutorial. *Tutorialspoint* [online]. Hyderabad: Tutorials Point, c2018 [cit. 2018-04-06]. Dostupné z: https://www.tutorialspoint.com/jsf/index.htm

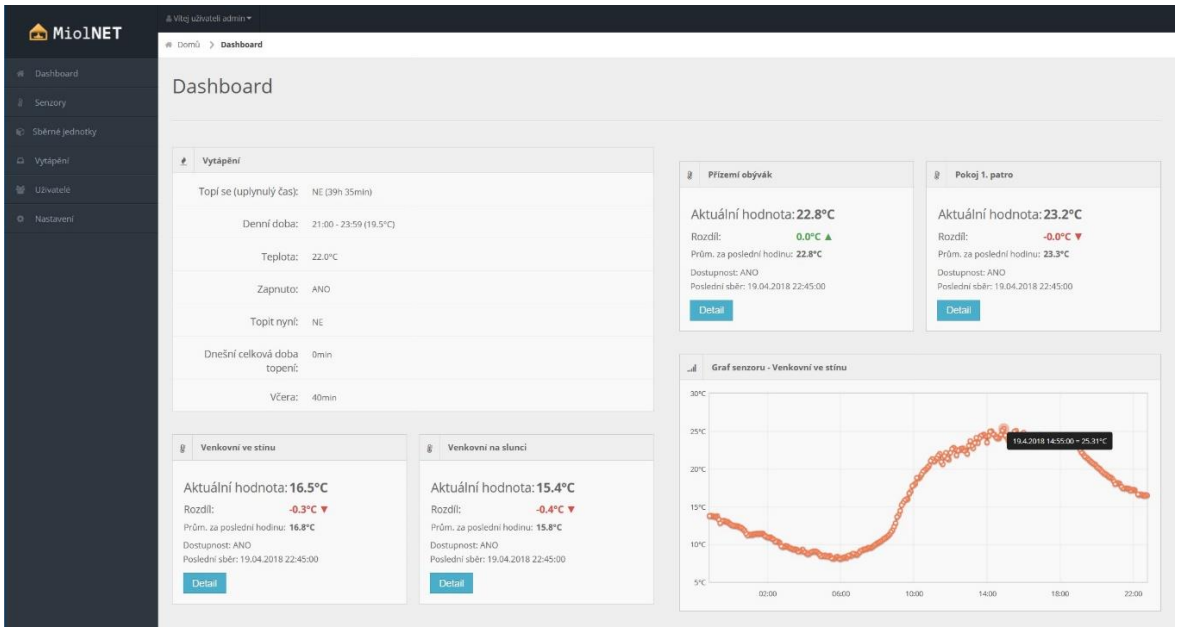
[24] Bootstrap 2.3.2 Documentation. *BootstrapDocs* [online]. [cit. 2018-04-06]. Dostupné z: http://bootstrapdocs.com/v2.3.2/docs/index.html

[25] JavaScript Tutorial. *W3schools.com* [online]. Sandnes: Refsnes Data, c1999-2018 [cit. 2018-04-06]. Dostupné z: https://www.w3schools.com/js/default.asp

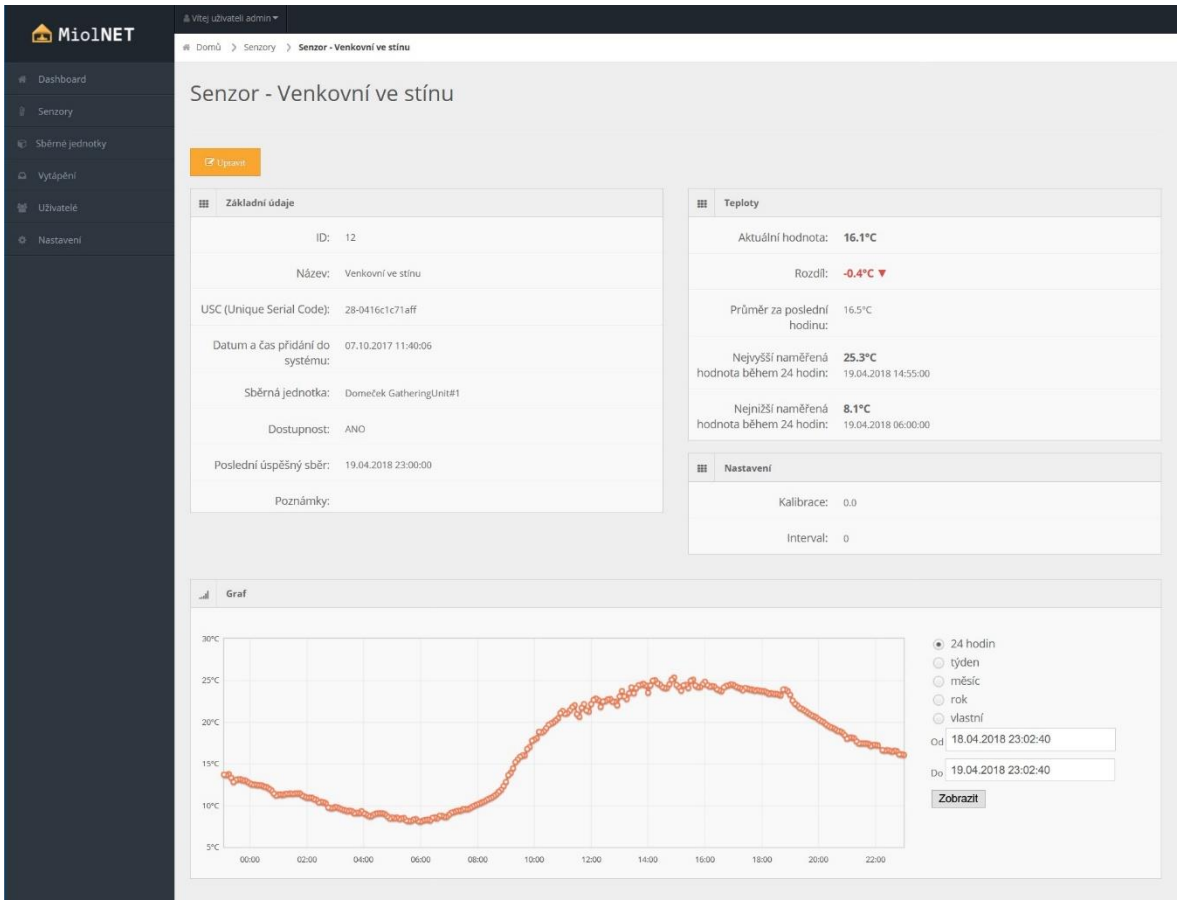
[26] Flot Reference. In: *GitHub* [online]. San Francisco, CA: GitHub, 2014 [cit. 2018-04-06]. Dostupné z: https://github.com/flot/flot/blob/master/API.md

9 Přílohy

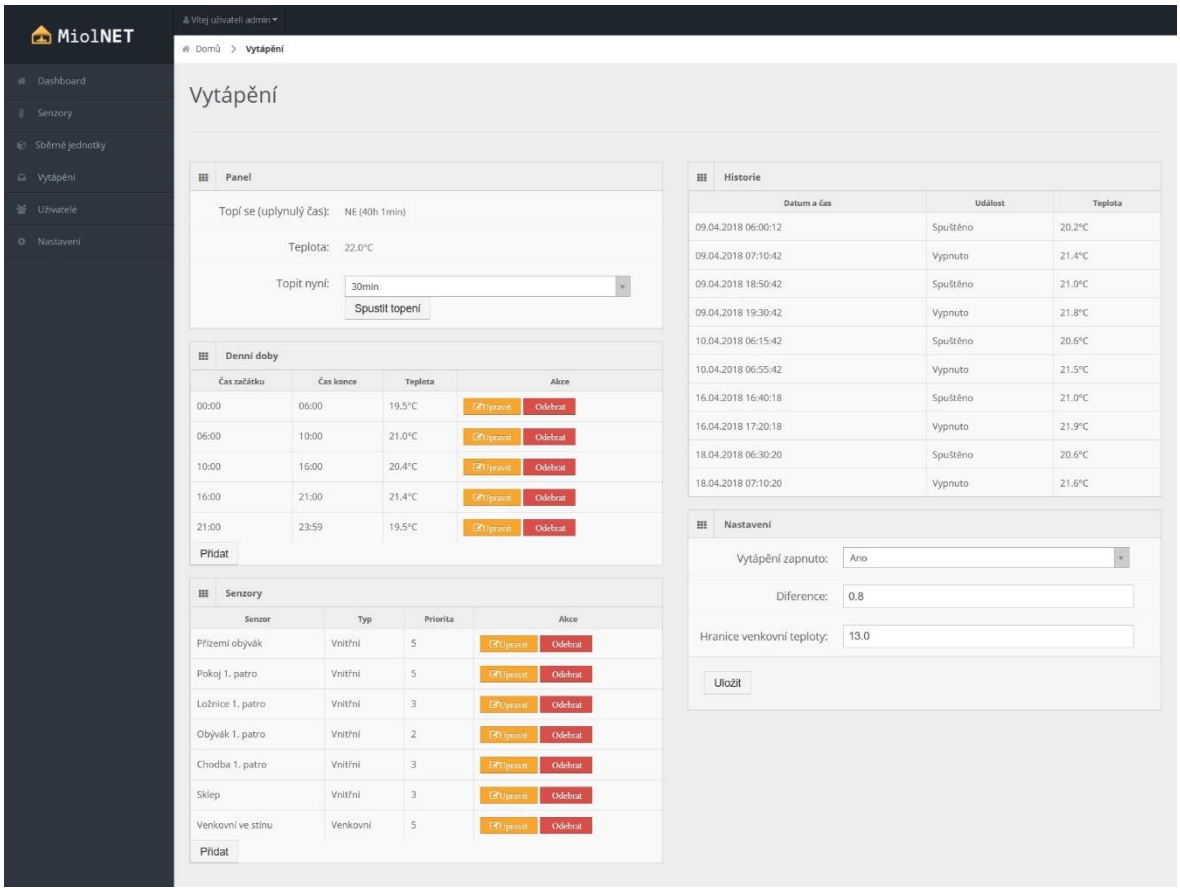
1) Ukázky částí webového rozhraní
2) Entitně relační diagram
3) Relační datový model
4) Zdrojové soubory aplikace na kompaktním disku



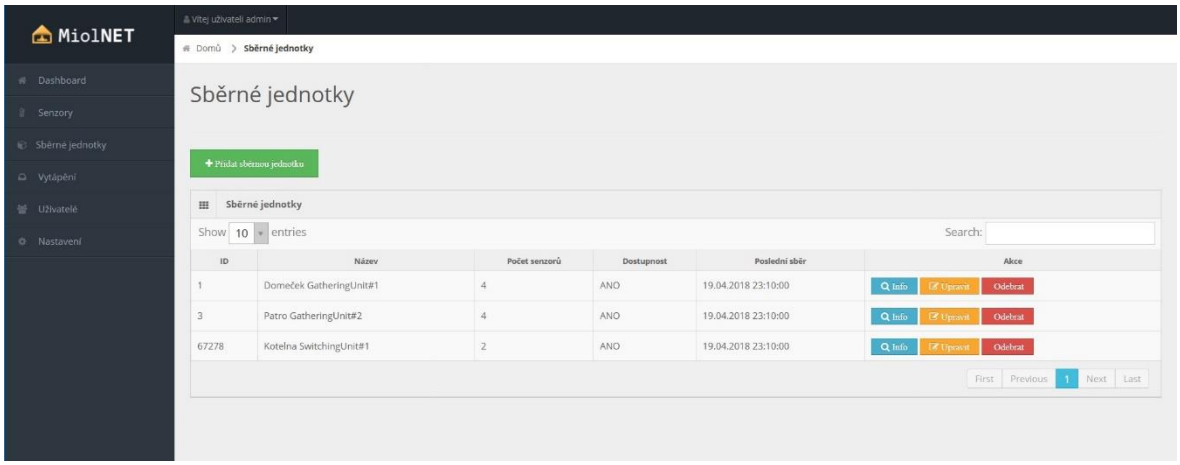
Obr. 1: Úvodní obrazovka aplikace



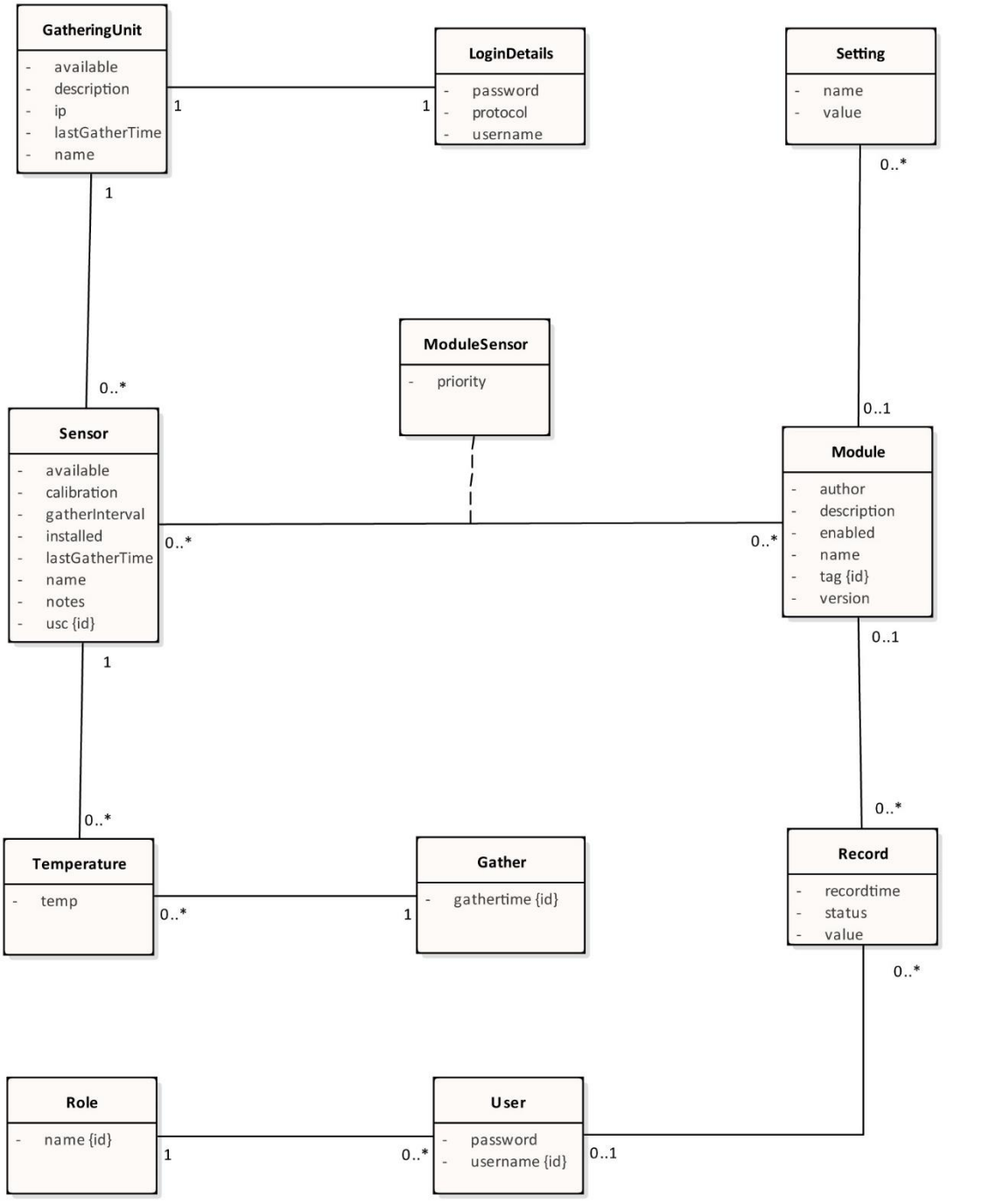
Obr. 2: Detailní zobrazení senzoru



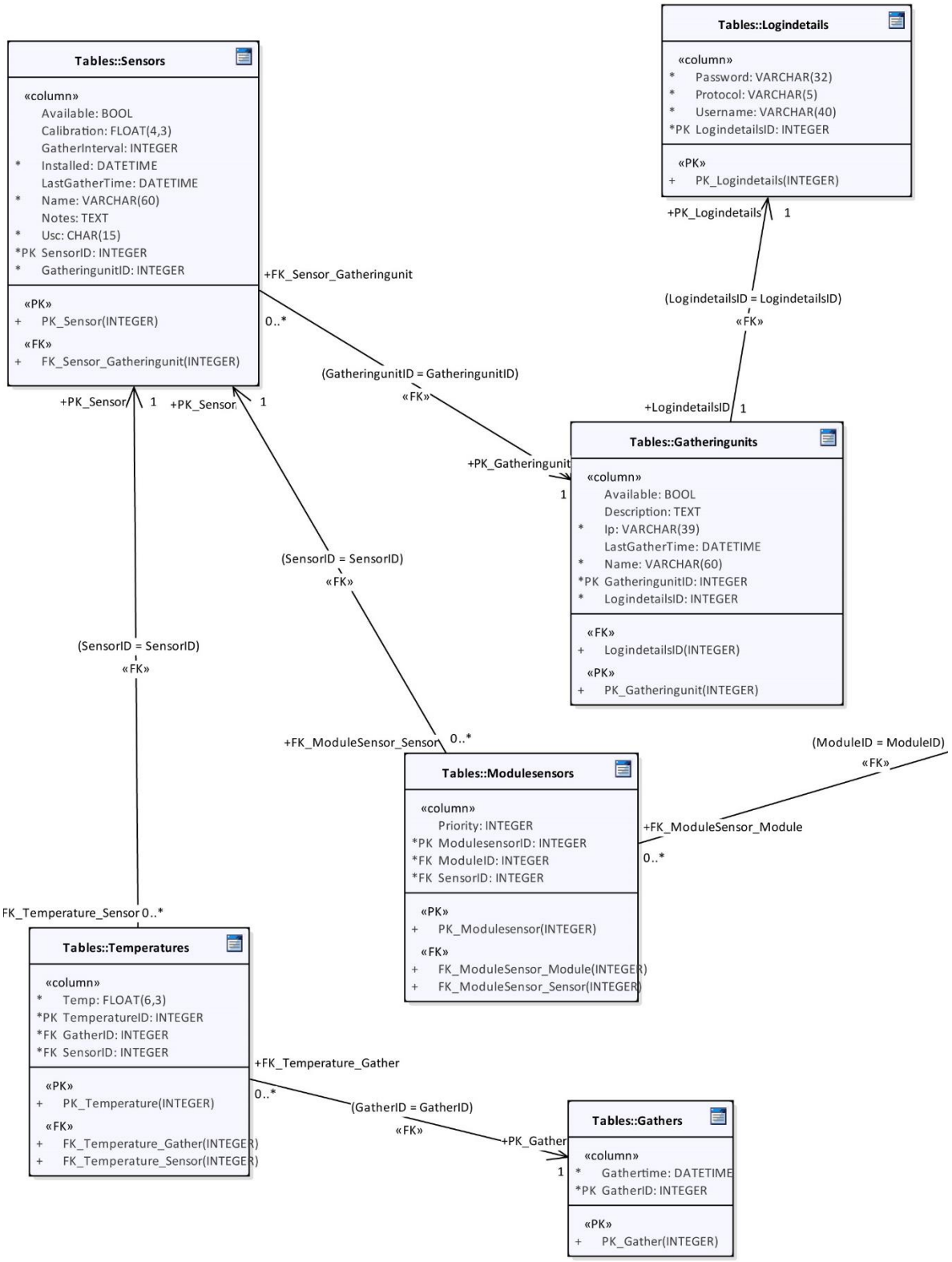
Obr. 3: Sekce Vytápění



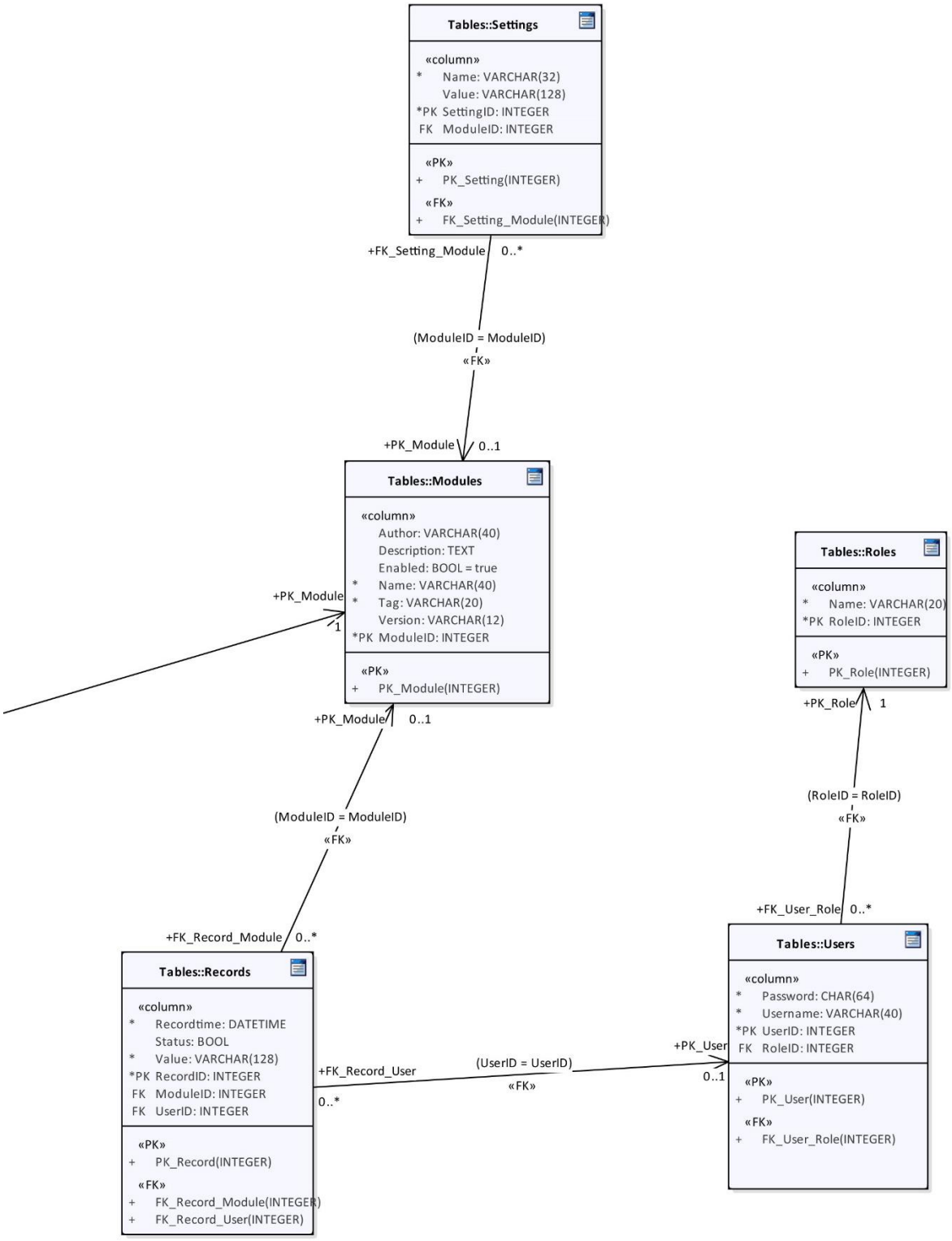
Obr. 4: Výpis sběrných jednotek v systému



Name: Entitně-relační diagram
Author: Olmr Miloš
Version: 1.3
Created: 1.8.2017 0:00:00
Updated: 19.4.2018 21:17:05



Name: MySQL Data Model
Author: Olmr Miloš
Version: 1.3
Created: 18.11.2017 0:00:00
Updated: 19.4.2018 20:55:16



Name: MySQL Data Model
Author: Olmr Miloš
Version: 1.3
Created: 18.11.2017 0:00:00
Updated: 19.4.2018 20:55:16

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Olmr Miloš	Husova 458, Poděbrady - Poděbrady II	I1500404

TÉMA ČESKY:

Řízení teploty v domě na základě dat ze senzorů

TÉMA ANGLICKY:

In-house temperature control based on sensor data

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce:
Navrhnout a realizovat Java EE aplikaci pro regulaci teploty v domě s využitím senzorů připojených k Raspberry Pi (nebo obdobnému zařízení) včetně webového rozhraní pro správu.

Osnova:
1. Úvod
2. Teoretické podklady
3. Analýza
4. Návrh
5. Implementace
6. Závěr
7. Literatura

SEZNAM DOPORUČENÉ LITERATURY:

GONCALVES, Antonio. Beginning Java EE 7. Berkeley, CA: Apress, 2013. ISBN 9781430246275.
DOUBRAVA, Jiří. Regulace ve vytápění. 2., upr. vyd. Praha: Společnost pro techniku prostředí, 2007. Sešit projektanta - pracovní podklady. ISBN 978-80-02-01951-0.
The Pi4J Project [online]. c2012-2016 [cit. 2017-10-09]. Dostupné z: http://pi4j.com/

Podpis studenta: ..... **Datum:** 10.10.2017

Podpis vedoucího práce: ..... **Datum:** 10.10.2017