
# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

## ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION


# WEBOVÉ ROZHRANÍ SYSTÉMU ŘÍZENÍ BUDOVY S PROTOKOLEM BACNET

WEB INTERCASE FOR BUILDING CONTROL SYSTEM WITH BACNET PROTOCOL

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS


AUTOR PRÁCE

AUTHOR

Tomáš Kretek

VEDOUCÍ PRÁCE

SUPERVISOR

prof. Ing. Pavel Václavek, Ph.D.



BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

# Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

***Student:*** Tomáš Kretek ***ID:*** 221304

***Ročník:*** 3 ***Akademický rok:*** 2021/22

**NÁZEV TÉMATU:**

## Webové rozhraní systému řízení budovy s protokolem BACnet

**POKYNY PRO VYPRACOVÁNÍ:**

1. Seznamte se se základy protokolu BACnet
2. Zpracujte rešerši dostupných nástrojů pro simulaci zařízení komunikujících protokolem BACnet
3. Zpracujte rešerši nástrojů pro vytváření webových rozhraní pro vizualizaci a ovládání systémů řízení budov s protokolem BACnet
4. Demonstrujte funkčnost vybraných vizualizačních nástrojů ve spojení se simulátorem zařízení BACnet a simulací jednoduchého modelu místnosti s tepelným modelem. Zaměřte se na ověření základních funkcí, včetně práce s prioritami BACnet.
5. Zhodnoťte funkčnost vybraných nástrojů s ohledem nasazení v reálném systému řízení budovy a možnost použití na HW typu RPi.

**DOPORUČENÁ LITERATURA:**

BACnet International: Introduction to BACnet, 2014

***Termín zadání:*** 7.2.2022 ***Termín odevzdání:*** 23.5.2022

***Vedoucí práce:*** prof. Ing. Pavel Václavek, Ph.D.

**doc. Ing. Václav Jirsík, CSc.**
předseda rady studijního programu

**UPOZORNĚNÍ:**

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

# Abstrakt

Tato bakalářská práce rozebírá základní vlastnosti protokolu BACnet, na který navazují vizualizační platformy a nástroje ThingsBoard, OpenHAB a Node-RED. Pro simulaci BACnet zařízení je vybrána knihovna BAC0 a program BOSS Simulator. Platforma ThingsBoard je použita pro demonstraci funkčnosti s knihovnou BAC0, která umožnila návrh modelu dvou místností s vytápěním a přechody tepla skrze stěny. V závěru práce je zhodnocen provoz ThingsBoard platformy na počítači Raspberry Pi.

# Klíčová slova

Vizualizační platformy, simulace BACnet zařízení, BACnet, ThingsBoard, OpenHAB, Node-RED, BAC0, BOSS Simulator

# Abstract

This bachelor thesis analyzes the basic features of the BACnet protocol, which is followed by visualization platforms and tools ThingsBoard, OpenHAB and Node-RED. The BAC0 library and the BOSS Simulator program are selected for the BACnet device simulation. The ThingsBoard platform is used to demonstrate the functionality with the BAC0 library, which allowed the design of a two-room model with heating and heat transfer through the walls. At the end of the work, the operation of the ThingsBoard platform on a Raspberry Pi computer is evaluated.

# Keywords

Visualization platforms, BACnet device simulation, BACnet, ThingsBoard, OpenHAB, Node-RED, BAC0, BOSS Simulator

# Bibliografická citace

KRETEK, Tomáš. *Webové rozhraní systému řízení budovy s protokolem BACnet* [online]. Brno, 2022 [cit. 2022-05-13]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/142702. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Pavel Václavek.

# Prohlášení autora o původnosti díla

| | |
|---|---|
| **Jméno a příjmení studenta:** | *Tomáš Kretek* |
| **VUT ID studenta:** | *221304* |
| **Typ práce:** | *Bakalářská práce* |
| **Akademický rok:** | *2021/22* |
| **Téma závěrečné práce:** | *Webové rozhraní systému řízení budovy s protokolem Bacnet* |

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dne: 13. května 2022 ----------

podpis autora

# Poděkování

Děkuji vedoucímu bakalářské práce prof. Ing. Pavlu Václavkovi, Ph.D. za trpělivost, podnětné rady k práci, časté konzultace a odborné vedení.


V Brně dne: 13. května 2022 ----------

podpis autora

# Obsah

# SEZNAM OBRÁZKŮ

# Úvod

Svět automatizace se neustále rozrůstá a zároveň s ním se vyvíjí nástroje, které usnadňují mnoho práce. Spousta těchto nástrojů je ovšem komerčně zaměřená a stojí nemalé peníze, proto je cílem této práce poukázat a přiblížit se k *open-source* nástrojům, které dokáží splnit vesměs vše potřebné bez nutnosti koupě licence a s podporou komunity. *Open-source* nástroje ovšem ne vždy fungují bezchybně a nenabízí všechny dostupné komponenty jako ty komerční. V takovýchto případech je uvedeno jedno alternativní řešení rozšiřující *open-source* platformu o další funkce zajišťující plnohodnotné nasazení v produkčním světě.

Práce je zaměřená na použití protokolu BACnet s vybranými nástroji. Každý z nástrojů je zcela unikátní a funguje na vlastním principu vytváření a programování funkcionalit. Kromě klientských nástrojů se práce zaměřuje také na návrh vlastního simulovaného BACnet zařízení pomocí jednoho ze dvou vybraných nástrojů, přičemž jeden patří do skupiny *open-source* a druhý do skupiny komerční.

Jako vizualizačním nástrojem s popsaným návrhem pro demonstraci funkčnosti je použita *open-source* platforma ThingsBoard v kombinaci s *open-source* IoT bránou, vyvíjenou pod záštitou ThingsBoard a veřejnou komunitou. Simulované zařízení je naprogramováno pomocí knihovny BAC0 a kombinuje dvě pokojové místnosti s vytápěním a tepelnými přechody skrze stěny.

Práce se dělí do šesti kapitol. První kapitola představuje základní vlastnosti protokolu BACnet, přičemž se podrobněji zaměřuje na jeho důležité prvky, jako jsou objekty, služby a priority. Ve druhé kapitole se nachází vizualizační platformy ThingsBoard, OpenHAB a Node-RED se základním představením jejich funkcionality. Platforma Thingsboard je zaměřena na průmyslové použití, OpenHAB na domácí použití a Node-RED na jiné typy aplikací, ale své uplatnění najde i v protokolu BACnet. Všechny zmíněné platformy slouží pro tvorbu klientských prostředí s možností zobrazování aktuálních a historických dat, včetně ovládání nakonfigurovaných zařízení. Třetí kapitolu tvoří knihovna a simulátor pro testování klientských aplikací. *Open-source* knihovna BAC0 z hlediska grafického zobrazení disponuje pouze možností základního přehledu sítě a trendů. Tvorba zařízení se zde provádí ručně psaným kódem. Komerční simulátor má všechno obsáhlé v přehledné aplikaci.

Čtvrtá kapitola rozebírá použití a strukturu navrženého simulátoru dvou místností s radiátory a tepelnými přechody pomocí knihovny BAC0.

Pátá kapitola ukazuje použití platformy ThingsBoard, včetně instalace a důležitých oprav chyb, které platformu značně omezovaly. Použit byl jeden dashboard s ovládacími tlačítky pro topení, indikátory stavů, nastavením teplot v místnostech a zobrazováním aktuálních dat. Mimo dashboard je navržen také vlastní kalendář teplot pomocí pravidel.

V kapitole šesté je zhodnoceno použití platformy ThingsBoard s IoT bránou na počítači Raspberry Pi

# 1. BACNET

BACnet lze vyjádřit jako standardizovaný komunikační protokol pro automatizační a řídící systémy budov. Protokol byl zřízen kvůli nekompatibilitě softwaru různých výrobců. Kvůli nekompatibilitě bylo nutné každý systém ovládat samostatně nebo s použitím vhodného rozhraní pro vzájemnou výměnu informací mezi systémy rozdílných výrobců. Vznik protokolu vnesl do oblasti automatizace nové možnosti díky sjednocení komunikace mezi systémy a jednotlivými zařízeními.

BACnet je koncepčně neutrální a s nelicencovanými specifikacemi [1].

## 1.1 Základní koncepce

BACnet stanovuje, jakým způsobem se zprávy dostanou od jednoho zařízení k druhému. Zprávy v sobě mohou nést hodnoty binárních a analogových vstupů a výstupů, vypočtené digitální a analogové hodnoty vstupů a výstupů, časové spínací programy, funkce signalizující alarmy a události, soubory, informace pro řízení, regulaci a další[1].

Přenos a předání zpráv je možné uskutečnit pomocí různých typů sítí, jako je *Ethernet*, *MS/TP, LONTALK, ARCNET* nebo také *Point-to-Point*, který je pro velké vzdálenosti.

Komunikace pomocí BACnetu je možná přes standardizované objekty představující například zařízení v síti, jeho jméno, vstupy a výstupy. S touto vlastností protokolu je možno získat informace o jakémkoliv zařízení bez nutnosti znalosti jeho vnitřní konstrukce nebo konfigurace.

## 1.2 Standardizované objekty

V této kapitole jsou uvedeny jedny z nejpoužívanějších standardizovaných objektů včetně vysvětlení jejich vlastností. Jednotlivých entit těchto objektů v zařízení je možné mít dle potřeby, přičemž každá entita je unikátní díky takzvanému objektovému identifikátoru, představujícího 32bitové číslo.

**Analogový vstup a binární vstup**

Analogový vstup zastupuje hardwarový vstup, například připojený snímač měřící teplotu okolí. Binární vstup reprezentuje logické stavy zapnuto/vypnuto a je možné jej invertovat pomocí *Property_Polarity*.

**Analogový a binární výstupní objekt**

Prostřednictvím analogových objektů vystupuje napětí a proud na hardwarový konektor zařízení. Pomocí binárního výstupu opět logický stav, například zapnutí a vypnutí klimatizace. Analogový výstup obsahuje *Property_Priority_Array*, kde se ukládají

[1] Kompletní přehled je dostupný na http://www.bacnet.org/Bibliography/ES-7-96/ES-7-96.htm

příchozí akční veličiny pro použití v *Present_Value* v kombinaci s prioritou (viz kapitola 1.4). Binární výstup má k dispozici opět inverzi a také nejkratší dobu zapnutí a vypnutí, která se využívá zejména u motorů, protože se nemohou rychle vypínat a zapínat.

**Objekt analogové a binární hodnoty**
Objekty analogové a binární hodnoty nejsou vázány na vstupy a výstupy zařízení, jako u předešlých objektů. Představují vnitřní proměnné, dle kterých se následně řídí další systémy či běh programu.

**Objekt střední hodnoty**
V časovém intervalu objekt střední hodnoty počítá minimum, maximum a střední hodnotu. Volitelně lze nastavit, v jak velkém intervalu a z kolika vzorků se tyto hodnoty počítají. Disponuje přístupem k objektům daného zařízení nebo dalšího zařízení v síti.

**Další důležité objekty**
**Objekt hlásičů ohrožení** je používán pro bezpečnostní kritické aplikace, jako je hlášení požáru nebo kouře. Navazující **objekt oblasti bezpečnosti** v sobě má několik těchto hlásičů ohrožení a bezpečnosti. Umožňuje použití poplachu jenom v určitých definovaných částech budovy. V neposlední řadě existují i objekty samotného zařízení popisující všechny jeho vlastnosti a mnoho dalších[2].

## 1.3 Služby

Služby zprostředkovávají komunikaci mezi zařízeními (přistupují k objektům). Jako příklad lze uvést dotazování *Present_Value* (představující aktuální hodnotu) skrze *ReadProperty* s určitým identifikátorem objektu představujícího například analogovou hodnotu, viz kapitola 1.2. Služby se dělí do pěti skupin.

První kategorie představuje služby přístupu k objektům, tedy čtení a zápis. Patří zde *ReadProperty, WriteProperty, CreateObject, DeleteObject* a další[3].

Druhou kategorii představují alarmy a události zprostředkovávající sdělování stavů alarmů, provozních stavů, poruch a také změnu naměřených veličin. Další členění této kategorie je speciálně vyvinuto z hlediska potřeb automatizace. Dělí se na hlášení při změně hodnoty, interní objektová hlášení používající se například při dosažení maximální nebo minimální nastavené teploty. Další podkategorii představuje hlášení změny algoritmu. K těmto hlášením lze také přiřadit požadovanou prioritu.

---

[2] Kompletní seznam je k nalezení na http://www.bacnet.org/Bibliography/ES-7-96/ES-7-96.htm v tabulce č. 1.

[3] Kompletní seznam přístupových služeb lze najít na http://www.bacnet.org/Bibliography/ES-7-96/ES-7-96.htm, v tabulce č. 6.

Třetí kategorií jsou služby správy zařízení a řízení sítě. Služby v této kategorii umožňují například vypnout, zapnout a restartovat zařízení nebo nastavit hodiny. Spadá zde také služba *Who-Has* dotazující se na přítomnost zařízení v síti a hlášení *I-Have* nesoucí identifikátor zařízení s adresou v síti.

Předposlední kategorie přistupuje k souborům zařízení a umožňuje zápis a čtení souboru skrze *AtomicReadFile* a *AtomicWriteFile*. *Atomic* vyjadřuje, zákaz dvou současných přístupů k souborům.

Poslední kategorie je služba virtuálního terminálu, která je používána administrátory pro vzdálenou konfiguraci zařízení.

## 1.4 Zajištění dat a priority

**Zajištění dat** představuje pojistku při výpadku nebo jakékoliv technické poruše. Většina BACnet zařízení se konfiguruje pomocí programů dodávaných výrobcem nahraných přímo v daném zařízení. Tyto programy se mohou při poruše ztratit, a proto BACnet umožňuje zálohování a následné obnovení skrze *AtomicReadFile* a *AtomicWriteFile*.

**Priority příkazů** existují pro možnosti ovládat zařízení z více míst. Definují jakousi váhu daného příkazu. BACnet podporuje šestnáct stupňů priorit, přičemž nejnižší číslo představuje nejvyšší prioritu a naopak. Většina z těchto priorit je volně definovatelná, některé jsou však předem dané a standardizované.

Priority jsou podporovány jenom některými typy proměnných, a to takzvanými příkazovými proměnnými. Například u objektu s analogovým výstupem je to aktuální hodnota, tedy *Present_Value*. Hodnota je zapisována do tabulky i s prioritou. Zařízení následně podle této tabulky používá aktuální hodnotu dle určené nejvyšší priority. Aby zařízení smělo použít hodnotu s nižší prioritou, než aktuálně používá, musí se vyšší priorita smazat zapsáním hodnoty *null* do políčka aktuální hodnoty u dané priority.

# 2. VIZUALIZAČNÍ NÁSTROJE

## 2.1 ThingsBoard Community Edition

ThingsBoard je široce rozšířená IoT platforma pro rychlý vývoj, správu a škálování IoT projektů. Mezi její hlavní funkce se řadí poskytování zařízení definovaným uživatelům, sběr a vizualizace dat ze zařízení, analýza telemetrie a následné spouštění definovaných alarmů, ovládání pomocí vzdálených procedur, definice specifických funkcí pomocí řetězců pravidel nazývaných *rule chains*, vytváření dynamických a responzivních řídících panelů a přeposílání dat do jiných systémů.

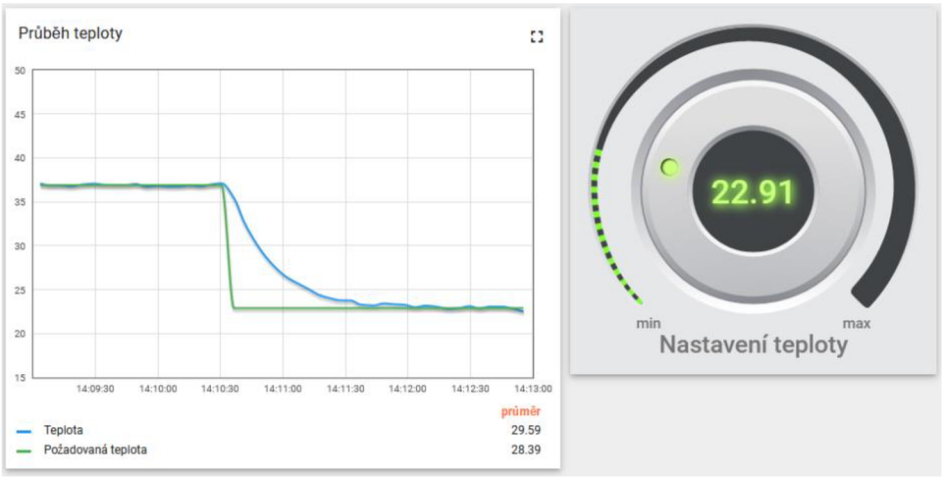
Platforma je dostupná ve více verzích. *Community Edition* je *open-source* a plně zdarma. *Professional Edition* je placená verze s více funkcemi, kterou je možné hostovat v cloudu [2].

Provoz této platformy je možný na jakémkoliv systému. Pro Linux je třeba mít nainstalovanou Javu, příslušnou databázi a frontovou službu zvolenou podle očekávaného použití. Pro Windows a MAC OS je doporučeno použít Docker.

K dispozici je také zdrojový kód pro mobilní aplikaci, který umožňuje publikaci aplikace v obchodě Google Play nebo App Store. Při jeho úpravě je nutné změnit adresu serveru, ke kterému se aplikace připojuje. Následné nastavení zobrazovaných částí dashboardu v aplikaci lze provést v samotné platformě ThingsBoard v příslušných zobrazovaných komponentech (widgetech).

### 2.1.1 Dashboard

Dashboard je navržen pro vizualizaci dat a ovládání zařízení. Jejich počet není nijak omezen a záleží na administrátorovi, jaký počet prostředí definuje a jakým uživatelům povolí přístup, případně konfiguraci. Prostředí se skládá z několika prvků. Mezi ty hlavní se řadí widgety a časová okna.




Obrázek 2.1 Ukázka dashboardu z platformy ThingsBoard

Před vytvořením dashboardu je nutné vytvořit požadovaný alias, který se následně považuje jako zdroj dat z jednoho nebo více zařízení.

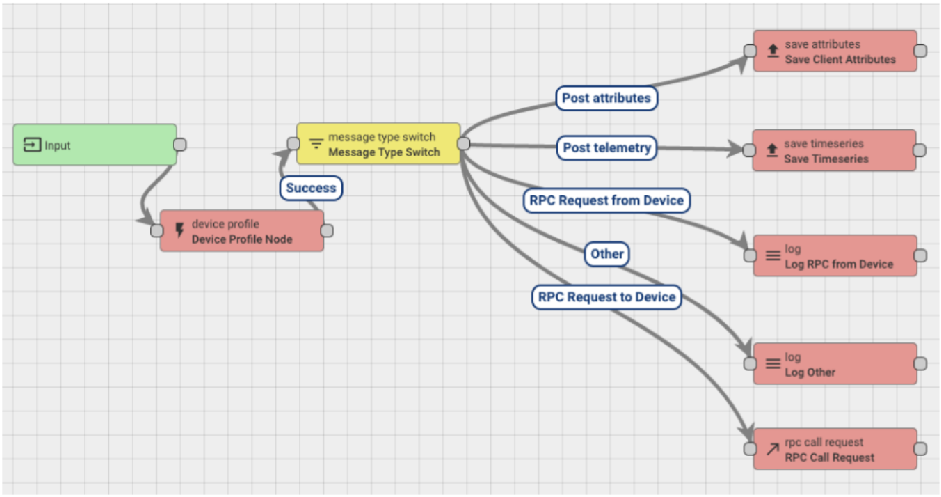
Každý widget poskytuje funkce pro koncového uživatele. Do těchto funkcí spadá již zmíněná vizualizace dat a také ovládací prvky pro definované zařízení. Dělí se na pět kategorií. **Časové grafy** zobrazují naměřená data za určené období. **Poslední hodnoty** zobrazují poslední naměřené hodnoty ze zařízení. **Ovládací widgety** umožňují zasílání příkazů danému zařízení přes *RPC*. **Alarmy** jsou definovány pomocí podmínek a mohou upozornit například na vysokou teplotu. Poslední kategorií zastávají **statické widgety**.

**Časová okna** definují, v jakém čase se data na widgetech zobrazují a jaká je použita agregátní funkce. Existují zde dva módy. Prvním je mód v reálném čase zajišťující neustálé získávání nových dat a následné zobrazování v definovaném časovém okně. Druhý mód je historický. V tomto případě dashboard požádá o data pouze při jeho inicializačním načtení a tyto data zobrazí.

**Agregátní funkci** je možno použít pouze pokud se nejedná o widget s alarmem, přičemž dokáže generovat minimální a maximální hodnotu v daném časovém období, průměr, součet a počet. Používá se zejména pokud je třeba snížit celkové využití sítě a zlepšit výkon koncového klientského prohlížeče, protože generuje tyto data na úrovni databáze.

### 2.1.2 Rule engine

Rule engine je *Framework* vytvořený pro jednoduchou tvorbu postupů založených na událostech. Skládá se ze tří hlavních komponent. První komponentou je **zpráva**. Ta definuje příchozí události, jako jsou data ze zařízení, události z *REST API* nebo *RPC* požadavek. Druhou komponentou je **pravidlo** aplikující se na příchozí zprávu a třetí komponentou je **řetězec pravidel**, což není nic víc než propojení více pravidel mezi sebou.




Obrázek 2.2 Ukázka pravidel z platformy ThingsBoard

Typickým použitím rule enginu je validace a modifikace dat před uložením do databáze, vytváření a úprava alarmů, načtení dalších požadovaných dat a vytvoření dalších *RPC* požadavků.

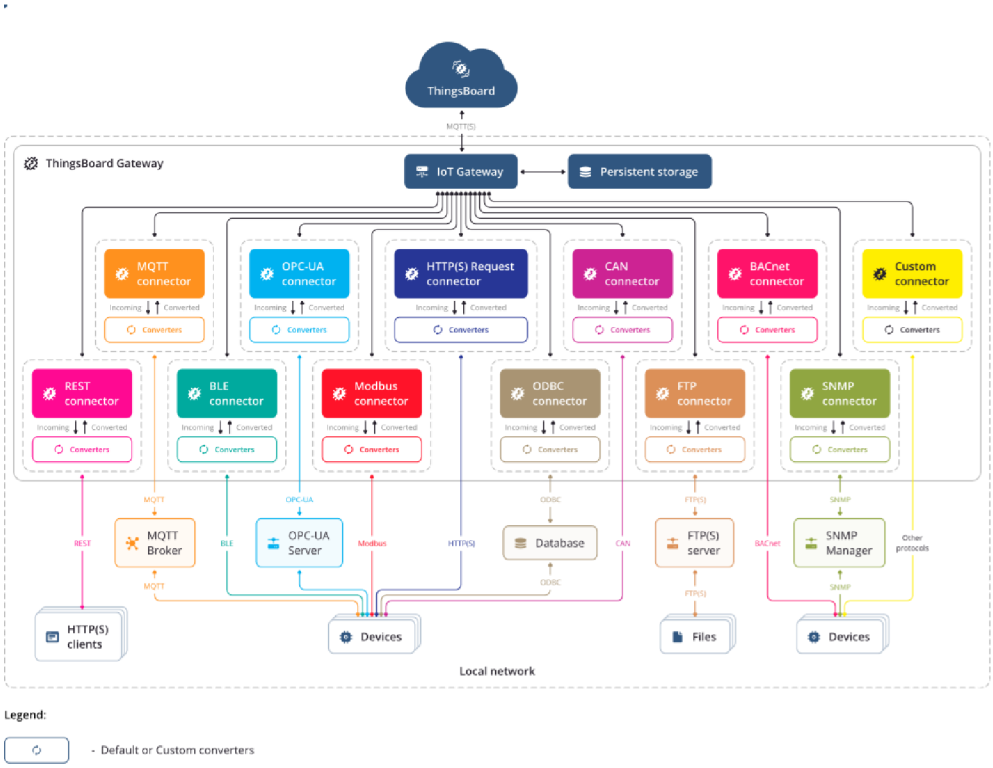
### 2.1.3 Možnosti připojení zařízení

Existuje široké předdefinované pole možností, jak zařízení připojit. ThingsBoard disponuje vestavenými protokoly *MQTT, CoAP a HTTP* a vlastní IoT bránou.

Pokud zařízení umí komunikovat jedním z těchto protokolů, je možné jej k ThingsBoardu připojit napřímo přes nachystané API.

Pokud výše zmíněné protokoly nepodporuje, použije se IoT brána. Tato brána napomáhá k připojení zařízení na místní síti, které nemají přístup k internetu nebo k protokolům využívající IP adresu. Její funkcí je převod dat ze zařízení do interního ThingsBoard formátu a následné odeslání dat přes *MQTT* protokol do platformy.

V konfiguračním souboru brány lze vybrat mezi řadou předdefinovaných konektorů. Například *MQTT, Modbus, CAN, BACnet a REST* konektor. Konfigurace každého konektoru probíhá v jeho vlastním konfiguračním souboru, ve kterém se definují parametry zařízení. Pokud žádný z konektorů nevyhovuje daným požadavkům, lze si navrhnout vlastní.




Obrázek 2.3 Schéma brány z platformy ThingsBoard [3]

V případě BACnet konektoru jsou k dispozici následující parametry. Pro pojmenování zařízení je k dispozici parametr *deviceName*, který může obsahovat přímo napsaný název zařízení nebo jej ze zařízení převzít. Dále je zde *deviceType* pro určení typu zařízení, pole *address* pro zadání IP adresy zařízení a *pollPeriod* pro periodické vyčítání dat z *timeseries*, ze kterých se vykreslují grafy v dashboardu.

Pole *attributes* v sobě uchovává atributy, které lze použít při návrhu dashboardu. Je zde parametr *key* jako identifikátor atributu, *objectId* nesoucí název objektu v BACnet zařízení a *propertyId* pro určení typu vyčítané hodnoty, například *presentValue*. Stejné parametry se používají v poli *timeseries*.

O něco zajímavější jsou pole *attributeUpdates* a *serverSideRpc* sloužící primárně pro zápis hodnot do zařízení. K dispozici jsou parametry *key, requestType, objectId, propertyId* a v *serverSideRpc* také *timeout*. *RequestType* definuje, jestli se daná hodnota čte nebo zapisuje, *timeout* udává čas v milisekundách pro maximální dobu čekání na odezvu zařízení. Ostatní parametry jsou stejné jako u *timeseries* a atributů.

## 2.2 openHAB

Platforma s názvem openHAB (open Home Automation Bus) je *open-source* řešení pro všechny chytré domácnosti. Její silné stránky jsou integrace velkého množství dalších systémů a zařízení díky nespočtu dostupných *addonů*[4] a poskytování přehledného jednotného uživatelského rozhraní. Díky flexibilitě platformy lze navrhnout téměř jakékoliv řešení na jakýkoliv požadavek z hlediska automatizace.

Architektura platformy je rozdělena na několik částí, kde každá část představuje nějakou funkci. Mezi ty hlavní se řadí **Stránky**, **Věci**, **Kanály**, **Položky** a **Pravidla** (pojmy odvozeny z anglických názvů) [4].

Provoz platformy je možný opět na jakémkoliv systému s nainstalovanou Javou v příslušné verzi. K použití je také Docker, jako v případě platformy ThingsBoard.

Mobilní aplikace je více přívětivěji řešena než v případě ThingsBoard aplikace, jelikož je dostupná v obchodech Google Play, App Store a Windows Store, kde není vyžadováno vlastní programování a následné zpřístupnění aplikace skrze zmíněné obchody (vyžaduje založení vývojářských profilů a placení poplatků). Pro použití aplikace je třeba v menu aplikace nastavit IP adresu serveru, na kterém je spuštěný openHAB.

### 2.2.1 Stránky

S příchodem openHAB třetí verze je nově předělané uživatelské prostředí. Prostředí náleží administraci platformy a také pro prezentaci definovaných systémů uživatelům. Nachází se zde všechny widgety a ovládací prvky, stejně jako v případě ThingsBoard dashboardu (viz kapitola 2.1.1).

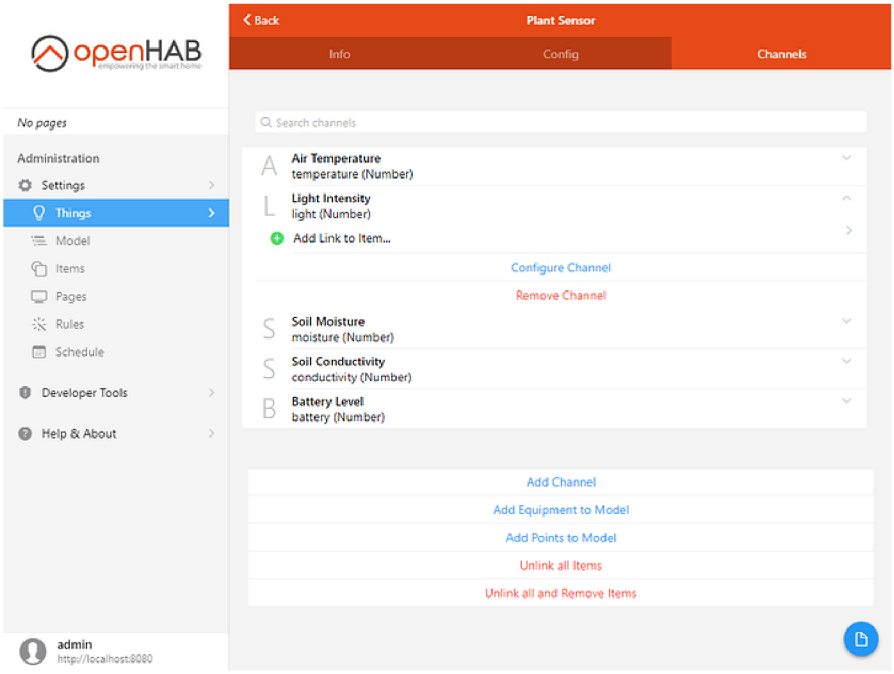
[4] Kompletní seznam addonů je dostupný na https://www.openhab.org/addons/

Prostředí disponuje zobrazením pro administrátory a pro obyčejné uživatele. Obyčejný uživatel má přístup k interaktivním prvkům stránky, k zobrazování a otevírání dalších aplikací, ale nemá možnost vidět administrativní prvky a nastavení.

### 2.2.2 Věci

Věci reprezentují přidané entity, které mohou poskytovat velké množství funkcí. Nemusí se jednat o konkrétní fyzické zařízení, je možné si pod touto definicí představit například webovou službu.

Věci jsou obohaceny také mosty, které jsou používány pro zpřístupnění dat jinačí věci.




Obrázek 2.4 Ukázka nastavení věcí v platformě OpenHAB [5]

Úzce spojené s věcmi jsou kanály, které představují konkrétní funkci poskytovanou věcí. Nachází se mezi fyzickou a virtuální vrstvou platformy a zajišťují tím propojení mezi věcmi a položkami. Skrze kanály následně proudí celá komunikace mezi definovanou věcí a jednotlivými funkcemi v uživatelském prostředí.

### 2.2.3 Položky

Položky představují virtuální vrstvu a nesou funkcionalitu používanou v aplikaci. Jejich použití tkví v událostech. Definovaných je několik typů položek, jako příklad lze uvést barva, kontakt, lokace nebo stmívač.

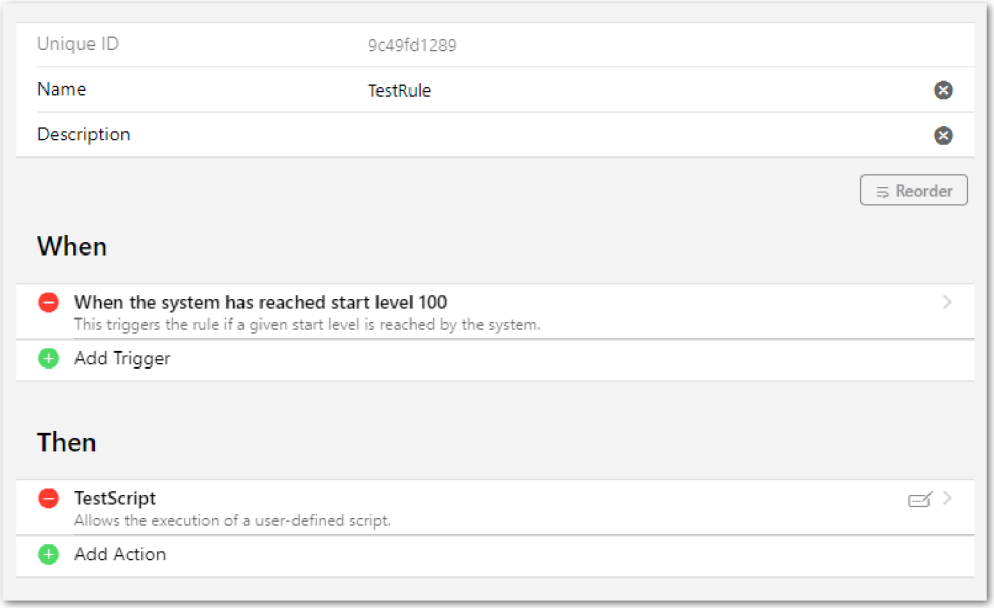
Skupinové položky jsou volně definovatelné, přičemž uživatelské rozhraní disponují možností zobrazovat tyto položky samostatně. Položky mají možnost nést svůj definovaný stav založený na ostatních položkách v dané skupině.

K položkám je v případě nutnosti možné připojit také metadata. Tyto data reprezentují další informace pro zařízení, aby mohlo příkaz lépe zpracovat a pochopit původní význam. Typickým použitím takového postupu je při použití chytrého zařízení Alexa.

### 2.2.4 Pravidla

OpenHAB stejně jako ThingsBoard obsahuje výkonný modul pravidel. Každé pravidlo spouští příslušný skript, který provádí požadované úkony. Tyto pravidla se nachází v konfigurační složce *rules*. Díky této jednoduché přístupnosti k souborům je možné všechny pravidla umístit do jednoho velkého souboru nebo je rozdělit do samostatných souborů, protože mezi sebou navzájem sdílí přístup k proměnným.

Nabízí se zde také možnost tvořit pravidla skrze *UI* platformy, což je poněkud pohodlnější. V nastavení se přidá nové pravidlo, definuje se jeho jméno a spouštěč. Následně se vybere funkce a pokud se jedná o spouštění skriptu, lze tento skript napsat přímo v nachystaném editoru, který usnadní práci díky kontrole syntaxu kódu, rozlišování barev a přichystaným šablonám.




Obrázek 2.5 Ukázka nastavení pravidla v platformě OpenHAB [6]

První kategorií jsou **pravidla založená na událostech**. Umožňují naslouchat změně stavu dané věci. Administrátor má na výběr z použití konkrétního nebo více komplexního příkazu a stavu. K dispozici je také naslouchání změn stavu dané skupiny věci.

Další kategorií jsou **pravidla založená na čase**. Tyto pravidla jsou užitečné, pokud je třeba měnit nastavení v definovaném čase, například při večerce. V podpoře je také doplněk *cron*[5].

**Pravidla založená na naslouchání věcí** jsou k zachytávání změn přijatých z dané věci. Opět je zde možnost zachytávání jednoduchého nebo komplexního příkazu.

Předposlední kategorii jsou **pravidla naslouchající spouštěcím kanálům**. Tyto kanály podporují jenom některé *addony*, které ovšem nenesou informace o stavu věci.

Do poslední kategorie spadají **pravidla založená na událostech ze systému**. Příklad takového použití je spuštění operačního systému, na což pravidlo reaguje.

Skripty psané pro openHAB používají jazyk *Xtend*[6], který je podobný jazyku Java a aplikuje se přímo za běhu aplikace, takže se nemusí předem kompilovat.

### 2.2.5 Možnosti připojení zařízení

Připojení dalších systémů a zařízení je prováděno skrze vazby. Platforma disponuje obrovským množstvím těchto vazeb vytvořených speciálně pro konkrétní systémy. Každá vazba má vlastní *xml* soubor, s jejími vlastnostmi.

Přestože openHAB disponuje takovým množstvím vazeb, pro BACnet aktuálně není k nalezení žádná správně funkční. Tento problém lze řešit skrze další spuštěnou službu na serveru, která se chová jako brána mezi protokolem *MQTT*, který platforma plně podporuje, a protokolem BACnet.

U brány je nutno se spolehnout na řešení od třetí strany nebo na své vlastní. Jako použitelná verze od třetí strany se jeví *bacnet-mqtt-gateway* od autora infinimesh[7]. Brána je celá napsaná formou aplikace v jazyce *JavaScript*. Je tedy jednoduše použitelná a upravitelná. Pro použití je nutné mít nainstalovaný *Node.js*[8], na kterém aplikace (brána) funguje. Využívá dvou knihoven napsaných také autory třetí strany, a to knihovny *bacstack*[9] a *mqtt*[10]. K dispozici je také jednoduché webové rozhraní pro hledání BACnet zařízení na síti.

Nevýhoda této brány je ovšem v tom, že je stavěná na jednosměrnou komunikaci, a to směrem z BACnet zařízení do *MQTT brokeru*. Zprávy chodící opačným směrem v této bráně mizí, a proto je třeba si tento směr komunikace dopsat samostatně.

Aby brána mohla komunikovat skrze *MQTT* ve spojení s openHABem, musí na systému běžet služba *MQTT broker*. V tomto případě lze využít *Eclipse Mosquitto*[11].

---

[5] Více o cronu je k nalezení na https://www.quartz-scheduler.org/documentation/quartz-2.2.2/tutorials/tutorial-lesson-06.html

[6] Více o jazyce Xtend je k nalezení na https://www.eclipse.org/xtend/documentation/203_xtend_expressions.html

[7] Zdrojový kód je dostupný na https://github.com/infinimesh/bacnet-mqtt-gateway

[8] Dostupné na https://nodejs.org/en/

[9] Knihovna bacstack je dostupná na https://github.com/fh1ch/node-bacstack

[10] Knihovna mqtt je dostupná na https://github.com/mqttjs/MQTT.js

[11] Více o Eclipse Mosquitto na https://mosquitto.org/

## 2.3 Node-RED

Node-RED zastává funkci takzvaného *flow-based* programovacího nástroje. Jedná se o poměrně odlišný nástroj než výše popisované platformy ThingsBoard (viz kapitola 2.1) a openHAB (viz kapitola 2.2). Již ze zmíněného *flow-based* lze odvodit, že vývoj v tomto prostředí je založen na toku událostí mezi uzly. Zatímco v případě ThingsBoard se takovým stylem definují pravidla, v tomto případě se takhle vytváří celá aplikace.

Nástroj je v základu postavený na *Node.js runtime*, což mu umožňuje plné využití jeho událostmi řízeného modelu. Provoz tohoto nástroje je tedy možný opět na jakémkoliv zařízení, na kterém je *Node.js* nainstalován.

Po spuštění aplikace lze skrze internetový prohlížeč otevřít připravený editor. V editoru se celá aplikace programuje pomocí propojování uzlů. Následným kliknutím na nahrávací tlačítko se celá aplikace nahraje do *runtime* a je přístupná skrze další adresu v prohlížeči [7].

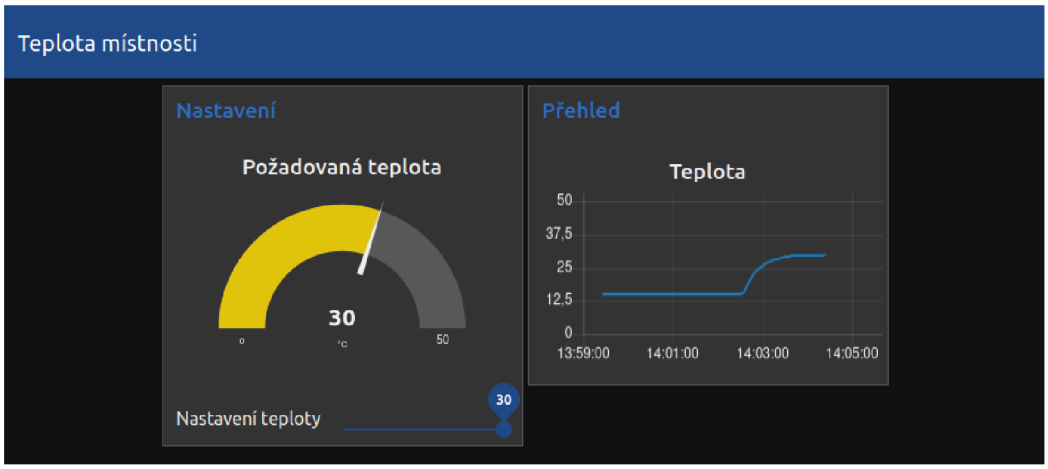
Node-RED v základu přichází pouze se základními typy uzlů (balíčků). K dispozici je ovšem celá řada dalších[12] díky aktivní komunitě. Pro instalaci se nabízí dvě možnosti. První možností je instalace skrze webový editor, který disponuje vyhledáváním aktuálních dostupných balíčků. Druhá možnost je instalace balíčků pomocí nástroje *npm*[13], který se instaluje automaticky s *Node.js*. Instalace se provádí použitím příkazu `npm install <jméno balíčku>`. Stejně tak se provádí aktualizace a mazání pomocí `npm update <jméno balíčku>` a `npm remove <jméno balíčku>`. Pokud je třeba zpracovat více balíčku najednou, jejich názvy se oddělí mezerou.

### 2.3.1 Dashboard

Přestože je Node-RED rozsáhlý programovací nástroj, je možné jej díky dostupnosti různých balíčků proměnit v přívětivé, responzivní a interaktivní uživatelské prostředí. V tomto případě k tomu lze použít *node-red-dashboard*. Balíček obsahuje set předdefinovaných uzlů k použití, například číselné a textové vstupní prvky, přepínače, formuláře, uzly pro výběr barvy a potom také přehrávače audia a grafy.

[12] Kompletní seznam dostupných uzlů je k dispozici na https://flows.nodered.org/

[13] Více o nástroji npm na https://www.npmjs.com/package/npm

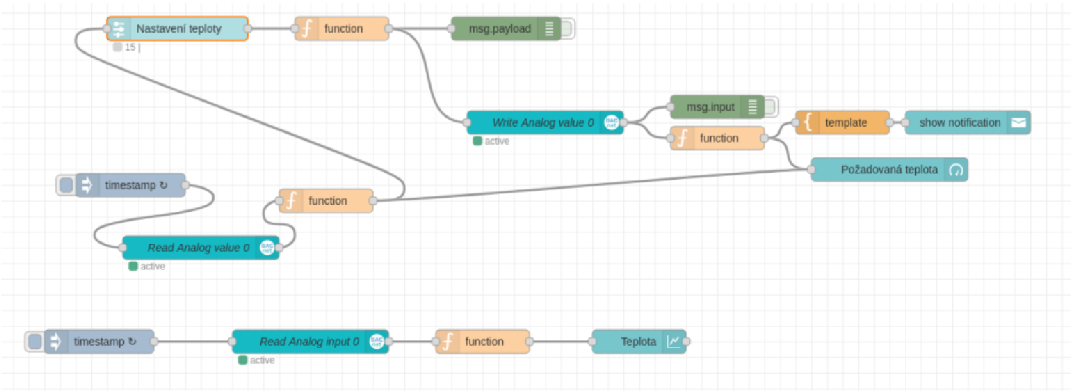



Obrázek 2.6 Ukázka dashboardu z nástroje Node-RED

Výsledné rozložení dashboardu je ve stylu tabulky. Algoritmus balíčku se pokouší rozložit prvky co nejvýše a vlevo. K tomuto rozložení napomáhají definované skupiny, do kterých se následně požadované prvky přiřazují. Pokud je skupin více, algoritmus při dostatku místa poskládá skupiny vedle sebe, naopak při zmenšení okna nebo otevření stránky na mobilním telefonu skupiny poskládá pod sebe. Stejně tak se chovají widgety v každé skupině. Pro přizpůsobení vzhledu je možnost u každé skupiny a widgetu nastavit jeho šířku.

### 2.3.2 Uzly

Uzly zde zastávají celkovou funkci programování od jednoduchých až po komplexní aplikace. Do základních uzlů se řadí uzly *Inject, Debug, Function, Change, Switch a Template*.




Obrázek 2.7 Ukázka uzlů z nástroje Node-RED

**Inject uzel** zastává jednoduše řečeno tlačítko vstupu, které má přesně definované, jakou zprávu pošle a může se ovládat manuálně nebo automaticky. Manuální ovládání probíhá přímo z editoru, automatické pomocí nastavených časových intervalů.

**Debug uzel** je velice užitečným pomocníkem při návrhu celé aplikace. Při jeho použití se do debugovacího okna v editoru vypisují všechny zprávy zachycené tímto uzlem. Zprávy obsahují důležité informace, jako čas přijetí zprávy, zdroj této zprávy a její obsah. Nabízí také možnost zápisu do logu.

**Function uzel** náleží k spouštění *JavaScript* kódu pro zpracování a úpravu přijaté zprávy. Lze s ním vytvořit různé požadované operace. V podpoře je rozesílání zpráv do více výstupů, takže pokud určitá zpráva nemá pokračovat na první výstup, lze ji poslat například na druhý. Další užitečná funkce je možnost generování více různých zpráv, které se posílání přesně za sebou. První dvě zprávy například na první výstup, třetí zpráva na druhý. Mimo jiné uzel podporuje také tvorbu asynchronních zpráv, logování, zpracování chyb a další[14] užitečné funkce.

**Change uzel** je jednoduchá verze *Function* uzlu umožňující vytváření, změnu, přesunutí a mazání zpráv. **Switch uzel** také zpracovává zprávy, ale v jiném kontextu, a to v rozdělování zpráv na různé výstupy podle definovaných pravidel.

Posledním uzlem je **Template**. Tento uzel dokáže formovat prostý text, doplněný o tělo přijaté zprávy. Uplatnění najde například při zasílání zprávy na uzel vytvářející oznámení na dashboardu při změně požadované teploty v místnosti. Jazyk použitý pro tvorbu textu se nazývá *Mustache*[15].

### 2.3.3 Zprávy

*Flow* v Node-RED funguje díky posíláním zpráv mezi uzly ve tvaru JavaScriptových objektů. Tyto zprávy obsahují *payload* a *_msgid*. *Payload* je používán jako obsah zasílané zprávy, *_msgid* identifikuje zprávu pomocí unikátního ID.

V některých případech použitých uzlů se může stát, že obsah zprávy není platný JavaScriptový objekt. To lze řešit pomocí *JSON* uzlu, který takto poslanou zprávu převede na platný JavaScriptový objekt, aby ostatní uzly mohly se zprávou dále pracovat.

Sekvence ve zprávách představují v podstatě více zpráv nebo dat schovaných v jedné zprávě. Tento případ může nastat například při čtení souboru. Pro zjednodušení, každý řádek souboru obsahuje data o produktu, které je nutné modifikovat. Takovýto soubor se načte do aplikace a následně rozdělí pomocí uzlu *Split*, který sám dokáže rozpoznat tvar zprávy a na jeho základě rozkouskovat zprávu na více částí. Tyto části se spočítají a přidají se jim ID a pozice v sekvenci, odešlou se dál, modifikují a následně opět spojí pomocí uzlu *Join*.

### 2.3.4 Kontext

Kontext je jedna z dalších užitečných funkcí, které Node-RED obsahuje. Otevírá nové možnosti práce se zprávami. Jedná se o *store* pro informace sdílené mezi uzly bez nutnosti posílat zprávy přes *flow*, tedy bez propojování uzlů mezi sebou.

---

[14] Více o dalších funkcích na https://nodered.org/docs/user-guide/writing-functions

[15] Více o Mustache na https://mustache.github.io/mustache.5.html

Rozsah sdílení dané kontextové hodnoty mezi uzly závisí na jeho definici. Node-RED podporuje tři typy těchto kontextů. Prvním je *Node*, který omezuje viditelnost kontextové hodnoty na uzel, na kterém byl vytvořen. Druhý typ kontextu je *Flow*. Ten je o něco volnější a umožňuje viditelnost kontextové hodnoty na celé větvi propojených uzlů. Třetí typ je *Global* a jak už ze slova plyne, viditelnost kontextu není nijak omezena a je dostupná pro všechny uzly.

Kontext je ve výchozím stavu nastaven tak, aby se ukládal do paměti serveru. To zapříčiní, že neobsahuje po restartu žádné data. Toto nastavení lze změnit na ukládání do lokálního souborového systému a zaručí, že data, které byly v kontextu uloženy, zůstanou uloženy i při restartu systému či jakémkoliv výpadku nebo chybě. Další možnost nastavení je také definice více kontextových *store* a zároveň pro každý zvlášť nastavení typu ukládání dat.

### 2.3.5 Projekty

Tvorba projektu je zajímavá a doporučená funkce při vytváření nové aplikace. Zaručuje jednoduché sdílení projektu mezi jinými vývojáři, včetně verzování pomocí nástroje *Git*[16]. Všechny použité balíčky pro vývoj aplikace jsou uloženy v souboru *package.json*, který je mezi vývojáři v JavaScriptu dobře známý. Každý použitý balíček zde obsahuje své jméno, pomocí kterého se instaluje s nástrojem *npm* nebo pomocí editoru (viz kapitola 2.3) a jeho požadovanou verzi. Takto při sdílení projektu stačí naklonovat z Gitu repositář s aplikací a nainstalovat balíčky. Vše ostatní je již připraveno, a pokud nějaký balíček chybí, editor upozorní nebo doporučí jeho instalaci.

Verzování projektu probíhá automaticky nebo manuálně. Při zvolení manuální varianty se příslušné změny v souborech ukazují v záložce historie, která zároveň umožňuje tvorbu *commitů*. V podzáložce historie *commitů* si lze následně prohlédnout všechny změny v daných verzích aplikace.

### 2.3.6 BACnet propojení

Pro práci s BACnet zařízeními v tomto editoru existují dvě možnosti. První možnost je stejná jako pro openHAB (viz kapitola 2.2.5), a to posílaní a čtení zpráv přes *MQTT* protokol skrze bránu zprostředkovávající komunikaci mezi protokolem *MQTT* a BACnet. Tato možnost je ale v tomto případě zbytečně nepraktická, protože pro Node-RED existuje balíček s uzly podporující BACnet zařízení, a to balíček *node-red-contrib-bacnet*.

Balíček využívá ke komunikaci JavaScriptovou knihovnu napsanou opět třetí stranou, jako v případě *BACnet MQTT* gateway použité u platformy openHAB. Podporuje definování BACnet klienta pro komunikaci na síti, definování jednotlivých zařízení pomocí *BACnet Device* a příkazy pro ovládání a vyčítání dat ze zařízení.

---

[16] Více o nástroji Git na https://git-scm.com/

# 3. SIMULÁTORY BACNET ZAŘÍZENÍ

Simulátory zařízení komunikujících přes BACnet umí vytvořit virtuální zařízení chovající se jako by se jednalo o zařízení reálné. Takové řešení přináší z hlediska vývoje určité aplikace a sítě řadu výhod. Aplikace vyvíjené za použití simulátoru lze testovat a nastavovat přímo ve fázi vývoje bez nutnosti dostupnosti reálné sítě zařízení. Díky takovému postupu lze aplikaci nasadit do produkčního použití bez nutnosti velkých změn, protože je předem ozkoušená a naprogramovaná.

Na internetu existuje řada simulátorů, ať už *open-source* nebo *komerčních*. Každý má většinou své výhody a nevýhody, nabízí jiné možnosti použití a nastavení. U komerčních simulátorů se lze spolehnout na jejich správnou funkci, ale už se nelze spolehnout na to, že budou tak volně nastavitelné s podporou všech potřebných funkcí k testování. *Open-source* simulátory naopak ne vždy fungují tak jak mají, ale pro změnu jsou svým volně dostupným a upravitelným zdrojovým kódem velice flexibilní. Na úkor více stráveného času zkoumáním a úpravou funkcionality *open-source* simulátoru v něm lze navrhnout prakticky cokoliv, co je třeba k testování, pokud je už v základu správně navržen.

## 3.1 BAC0 – BACnet test tool

BAC0 zastává *open-source* knihovnu napsanou v programovacím jazyce Python pro zpracování BACnet zpráv. Použití jazyka Python umožňuje snadnější pochopení funkcionality knihovny a tím i rychlé navrhování virtuálních zařízení.

V základu knihovna podporuje již zmíněné vytváření virtuálních zařízení (kontrolérů), jejich objektů (viz kapitola 1.2) a také čtení a zápis, což umožňuje návrh jakéhokoliv systému a regulátoru. Zajímavá funkce je také trendování všech definovaných proměnných v intervalu deseti sekund, což poskytuje již ze startu přístup k historickým datům kontroléru zařízení. Dále knihovna disponuje webovým rozhraním pro zobrazení základního přehledu sítě, trendů a různými testy [8].

### 3.1.1 Zprovoznění aplikace

Pro používání knihovny je nutné mít nainstalován Python verze 3.5 a vyšší, čehož je možné docítil buď instalací samostatného Pythonu[17] nebo použitím nástroje *Anaconda*[18]. Následně se pomocí nástroje *pip*[19] nainstaluje samotná knihovna pomocí `pip install BAC0` a pokud systém již obsahuje knihovnu *bacpypes*[20], je možno knihovnu začít používat. Pro používání webového rozšíření a ostatních funkcí je nutné mít nainstalovaný také *Bokeh, Pandas* a *Flask* (včetně *flask_bootstrap*).

[17] Více o Pythonu na https://www.python.org/

[18] Více o nástroji Anaconda na https://www.anaconda.com/products/individual

[19] Více o nástroji pip na https://pypi.org/project/pip/

[20] Více o BACpypes na https://github.com/JoelBender/bacpypes

### 3.1.2 Definování sítě

Ze startu je nejdříve nutné definovat síť, na které bude aplikace komunikovat. Toho lze docílit dvěma způsoby, a to použitím příkazu `BAC0.connect()`, který automaticky startuje knihovnu včetně webového serveru (za předpokladu nainstalovaných komponentů) anebo použitím příkazu `BAC0.lite()`, který startuje pouze základ bez webového serveru a trendovacích funkcí. Oba příkazy přijímají také parametr IP adresy a masky podsítě pro daný internetový interface, pokud se nepovede tyto parametry detekovat automaticky.

Pro interakci se zařízeními mimo lokální síť knihovna podporuje *BBMD*[21], které organizuje broadcast zprávy zasílané do jiných podsítí.

Po úspěšném startu je možné komunikovat s již existujícími zařízeními na síti, číst jejich objekty, zkoumat obsah sítě, zapisovat do zařízení nebo vytvořit své vlastní.

### 3.1.3 Definování a práce se zařízením

Prvním způsobem je definování instance pomocí `BAC0.connect()` a až následné definování zařízení pomocí `BAC0.device()` s použitím parametru instance, adresy zařízení a požadovaných objektů. Druhý způsob je přímé definování zařízení pomocí `BAC0.lite()` s použitím opět IP adresy zařízení, ID zařízení a požadovaných objektů. S kontrolérem zařízení lze následně provádět různé operace. Pokud je instance uložená například v proměnné *controller*, lze pomocí této proměnné ovlivňovat přiřazené objekty. Kontrolér má přímý přístup k těmto objektům, takže se používají jejich jména.

Další způsob interakce s hodnotami objektů je možný použitím příkazů `read_property()` a `write_property()`. První zmíněný příkaz vypíše celé prioritní pole daného objektu. Prioritním polem se zde rozumí tabulka hodnot s přiřazenými prioritami, viz kapitola 1.4. Druhý zmíněný příkaz zapisuje hodnotu do daného objektu s možností použití požadované priority.

### 3.1.4 Práce s historií objektů

Historie hodnot objektů a *TrendLog* pocházející přímo z BACnet protokolu jsou dvě rozdílné funkce. BAC0 využívá k záznamu historie knihovnu *Pandas*. Interval vzorkování hodnot je ve výchozím stavu nastaven na 10 sekund. Tento interval ale není pevně daný a lze jej modifikovat podle potřeby příkazem `resample()`. Data takto uložená jsou následně k dispozici pomocí příkazu *history*.

BACnet *TrendLog* je objekt, který může být naimplementován v kontroléru. Je ovšem limitován pamětí kontroléru, a tak je nutné zvolit dostatečně velký interval vzorkování pro požadovanou velikost záznamu. BAC0 převádí TrendLog do záznamu *panda Series* pro umožnění použití příkazu *history*.

---

[21] Více o BBMD na http://www.bacnet.org/Tutorial/BACnetIP/sld015.html

### 3.1.5 Další funkce

BAC0 také podporuje zápis a čtení do objektu týdenního plánu nebo COV (odebírání změny hodnoty) a na to navázání zpětné vazby, ukládání do výkonnější databáze, než je SQLite[22] a další[23].

## 3.2 BOSS Simulator

BOSS simulátor spadá do odvětví komerčních simulátorů a je vlastněný firmou Softdel[24]. Je to přehledný a jednoduchý program se spoustou funkcí. Nabízí se zde konfigurace BACnet sítě s generováním provozu, posílání příkazů do sítě, správa fyzických i virtuálních zařízení, monitorování sítě v reálném čase a záloha objektů zařízení pro pozdější obnovení. Zálohování je především užitečná funkce, pokud je nutné upravovat síť bez její fyzické dostupnosti [9].

Simulátor také podporuje *BDDM* funkci pro registraci zařízení z jiných podsítí a funkci automatického vyhledávání zařízení na síti včetně jejich objektů.

### 3.2.1 Práce s virtuálními zařízeními

Přidávání virtuálních BACnet zařízení je v tomto programu velice jednoduchou záležitostí. Nabízí se zde dvě možnosti, přičemž první možností je přidání jednoho nového zařízení, kde se zároveň vybere jeho ID instance. Druhou možností je přidání více zařízení najednou pomocí zvoleného rozsahu a začátku instancí. Každé zařízení lze následně kompletně replikovat nebo smazat.

Práce s objekty je zde také velice jednoduchá a intuitivní. Vše, co je nutné udělat, je vybrat typ objektu a přiřadit mu ID instance. Stejně jako v případě zařízení, i zde je možné objekty mazat a replikovat.

V pokročilém nastavení zařízení je k nalezení celková konfigurace. Nachází se zde všechny možnosti služeb a objektů, které je možné zapínat a vypínat podle potřeby, možnost změny MAC adresy, frekvence simulace a nastavení COV.

### 3.2.2 Simulace

Simulace je možná na analogových a binárních objektech, akumulačních a více stavových objektech. Mezi podporované typy simulací se řadí **náhodná hodnota** v daném rozmezí, **rampa**, což není nic jiného než periodické přičítání hodnoty a pro binární objekty je dostupná **reverzace hodnoty**.

Na výběr je zde také ze simulace daného zařízení nebo ze simulace globální týkající se všech zařízení. Frekvence simulace je velmi důležitý parametr, který je možné měnit v závislosti na daných požadavcích. Výchozí hodnota činí 5 sekund.

---

[22] Více o SQLite na https://www.sqlite.org/index.html

[23] Více o dalších funkcích BAC0 na https://bac0.readthedocs.io/en/latest/index.html

[24] Více o firmě Softdel na https://www.softdel.com/

V podpoře je také nahrávání předem definované globální simulace pomocí CSV souboru.

Nevýhoda těchto typů simulací je nemožnost definování své vlastní, což je ne vždy ideální z hlediska testování a návrhu svého virtuálního zařízení nebo celkové simulace, která má reprezentovat například simulací místnosti s vytápěním.

# 4. NAVRŽENÝ SIMULÁTOR PRO TESTOVÁNÍ

Simulátor BACnet zařízení je naprogramován jako spustitelný skript pomocí BAC0 knihovny pro jazyk Python. Python spadá mezi jednodušší jazyky, což umožňuje rychlejší tvorbu programů. BAC0 čerpá z knihovny *bacpypes*, která je vytvořená za účelem ovládání a možnosti tvorby BACnet zařízení. BAC0 tyto možnosti ještě vylepšuje a zaměřuje se na co nejjednodušší práci s BACnet zařízeními. Dokumentace k BAC0 ukazuje všechny možnosti použití a je dosti rozsáhlá. Patrnou nevýhodou je, že se knihovna stále vyvíjí, a neodemyká tak všechny možnosti *bacpypes*, a některé funkce nefungují dle očekávání.

Skript simuluje dvě oddělené místnosti o rozměrech 4x4x2,5 metrů. Obě místnosti disponují svým zdrojem tepla, v tomto případě elektrickým topením. Simulovány jsou zde také tepelné přechody zvenčí a mezi místnostmi, kde venkovní teplota činí 5°C. Topení je v první místnosti nastaveno na 21°C, v druhé na 20°C. Výchozí teplota pro první místnost činí 18°C, pro druhou místnost 19°C. Každá z těchto hodnot místností reprezentuje jeden objekt. Další objekty reprezentují povolení topení a stavy radiátorů. Posledním objektem je objekt kalendáře ovládající teplotu podle daného dne a hodiny. Objekt kalendáře ovšem nefunguje správně a není k němu dostupná dokumentace.

Pro výpočet teplot v místnostech je používán tepelný tok v kombinaci s tepelnou kapacitou místností. Použité hodnoty nekorespondují s realitou, jsou pouze převzaty a upraveny pro optimální běh a trvání simulace.

Regulaci teploty zastávají reléové regulátory s hysterezí 0,5°C.

## 4.1 Struktura skriptu

- Import potřebných funkcí pro běh skriptu
- Definice vstupních argumentů skriptu
- Vytvoření instance BACnet kontroléru
- Definice objektů zařízení a jejich přiřazení do kontroléru
- Definice konstant
- Hlavní smyčka skriptu simulující dvě místnosti

## 4.2 Spouštění skriptu

Skript lze spustit více možnostmi, přičemž ta nejběžnější je použitím terminálu (konzole). Povinným parametrem pro běh skriptu je předání IP adresy s maskou sítě zařízení v lokální síti parametrem *-a*. Nápověda je pro vstupní argumenty zobrazitelná s použitím argumentu *--help*, tedy *python rooms_simulator.py --help*.

```
tomaskretek@Tomas-MacBook-Air BACnet-simulator % python rooms_simulator.py --help
usage: rooms_simulator.py [-h] [-a ADDRESS] [-p PORT] [-di DEVICEID] [-st SLEEPTIME] [-log LOGGING]

options:
  -h, --help            show this help message and exit
  -a ADDRESS, --address ADDRESS
                        IPV4 address of device interface with network mask, e.g. 10.0.0.34/24 (default: None)
  -p PORT, --port PORT  Port of BACnet device that will be created (default: 47809)
  -di DEVICEID, --deviceId DEVICEID
                        ID of BACnet device that will be created (default: 101)
  -st SLEEPTIME, --sleepTime SLEEPTIME
                        Main loop sleep time (simulation speed) (default: 0.4)
  -log LOGGING, --logging LOGGING
                        Enable logging to console (default: False)
tomaskretek@Tomas-MacBook-Air BACnet-simulator %
```

Obrázek 4.1 Výpis konzole po zadání help argumentu

Spouštěcí příkaz se všemi parametry je

*python rooms_simulator.py -a 10.0.0.34/24 -p 47811 -di 200 -st 0.3 -log True*

kde *a* značí použitou IP adresu s maskou sítě, *p* použitý port, *di* zvolené ID zařízení, *st* zvolenou rychlost simulace a *log* výpis aktuálních teplot, požadovaných teplot a stavů radiátorů do konzole.

Je striktně doporučeno testovat simulátor na lokální síti pro zajištění dostupnosti používaných portů a komunikaci s platformou ThingsBoard. Předpokladem pro použití skriptu je dostupnost všech potřebných nástrojů a knihoven, viz kapitola 3.1.1.

Skript je dostupný skrze přílohu práce (viz příloha A.1) nebo GitHub[25].

[25] Odkaz na GitHub: https://github.com/Kretiss/BACnet-simulator/tree/rooms

# 5. ZPROVOZNĚNÍ VIZUALIZAČNÍHO NÁSTROJE

Vybraným nástrojem pro vizualizaci je platforma ThingsBoard. Disponuje rozsáhlou sadou dostupných widgetů, včetně varianty tvorby svých vlastních, jednoduchým a intuitivně řešeným prostředím, možností definovat vlastní pravidla pro zpracování a tvorbu událostí a v neposlední řadě také přehledem využití API, který je užitečný pro kontrolu stavu a vytížení platformy.

Přidružená IoT brána doplňuje chybějící možnosti konektivity s protokolem BACnet.

## 5.1 Instalace ThingsBoard s nástrojem Docker

Dokumentace[26] uvádí více možností instalace. Provoz platformy je doporučený na systému Linux bez použití nástroje Docker nebo v cloudu. Pro účely této práce je ovšem Docker použit z důvodu kompatibility systému. Ačkoliv je instalace jednoduchá a přímočará, dokumentace neuvádí řešení na problém, který nastává velice často po každé instalaci, a to nemožnost znovuspuštění platformy, například po restartu. Tento problém je rozebrán a vyřešen níže.

Pro použití tohoto typu instalace je nutné mít nainstalovaný Docker Engine a Docker Compose. V případě použití Docker Desktop varianty jsou tyto součásti nainstalovány automaticky, včetně prostředí pro správu kontejnerů, obrazů a disků.

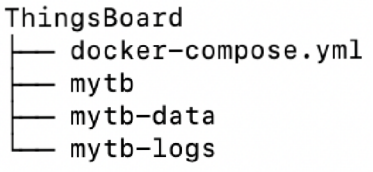
Dalším krokem je zvolení správné kombinace *ThingsBoard/databáze* pro příslušný provoz. Vybranou variantou je postgres databáze, která vystačí na testování a menší produkční provoz (vyzkoušeno v kombinaci s reálným PLC řídícím spoustu světel a zobrazováním aktuálních dat) a zároveň není náročná na požadavky hardwaru.

Frontová služba je opět řešena více možnostmi, v tomto případě postačí zvolit výchozí službu v paměti. Pro nasazení do produkce s větším provozem je doporučeno použít platformu Kafka nebo RabbitMQ.

Následuje konfigurace souboru *docker-compose.yml*. Soubor se v rámci této práce nachází ve vytvořené složce ThingsBoard v kořenovém adresáři uživatele. Definuje verzi souboru pro kompatibilitu s *Docker Engine* a příslušné služby. V této části nastává výše zmiňovaný problém s nemožností opětovného spuštění platformy. V dokumentaci je použit pouze jeden obraz platformy s databází, přičemž po opětovném spuštění přestává ThingsBoard s databází komunikovat (databáze se nespouští). Řešením je přidání druhého obrazu obsahujícího pouze *postgres* databázi s vlastním spouštěcím souborem v jiném umístění. Data databáze jsou ukládána do stejného umístění jako uvádí dokumentace. V *environment* části každého obrazu jsou definovány proměnné pro správnou komunikaci mezi kontejnery, přičemž u *mytb* služby je zvlášť pozornost věnována proměnným typu server zvyšující výchozí nastavení frontové služby

[26] Všechny možnosti dostupné na https://thingsboard.io/docs/user-guide/install/installation-options/

a aktualizací na jedno uživatelské sezení. Bez tohoto zvýšení thingsboard nezvládá aktualizovat dashboard při používání se simulátorem. V části *volumes* jsou definovány příslušné složky v počítači, kde budou data uložena. Výslednou strukturu složky demonstruje Obrázek 5.1.




Obrázek 5.1 Struktura složky ThingsBoard

Strukturu docker-compose souboru demonstruje Obrázek 5.2.

```
version: '2.2'
services:

  mytb:
    restart: unless-stopped
    image: "thingsboard/tb-postgres"
    expose:
      - "8080"
    ports:
      - "8080:9090"
      - "1883:1883"
      - "7070:7070"
      - "5683-5688:5683-5688/udp"
    environment:
      TB_QUEUE_TYPE: in-memory
      server.ws.limits.max_queue_per_ws_session: 1000
      server.ws.limits.max_updates_per_session: 1000:1,10000:60
      SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/thingsboard
      SPRING_DATASOURCE_PASSWORD: postgres
      SPRING_DATASOURCE_USERNAME: thingsboard
    volumes:
      - ./mytb-data:/data
      - ./mytb-logs:/var/log/thingsboard
      - ./mytb:/var/run/postgresql
    depends_on:
      - postgres

  postgres:
    restart: unless-stopped
    image: "postgres"
    ports:
      - "5432"
    environment:
      POSTGRES_USER: thingsboard
      POSTGRES_DB: dummmy
      POSTGRES_PASSWORD: postgres
    volumes:
      - ./mytb-data/db:/var/lib/postgresql/data
      - ./mytb:/var/run/postgresql

```

Obrázek 5.2 Konfigurace souboru docker-compose.yml

U složek je třeba změnit práva pomocí *sudo chown -R 799:799 <složka>* pro správnou funkci, přičemž *-R* značí rekurzivně a *799:799* označuje *vlastníka:skupinu*.

Zbývá otevřít terminál v ThingsBoard složce a napsat příkaz *docker-compose up*, který stáhne potřebná data a spustí kontejnery. Další zastavení a spuštění platformy je možné pomocí příkazu *docker-compose stop* a *docker-compose start*, popřípadě přes aplikaci *Docker Desktop*, pokud je nainstalována.

## 5.2 Instalace brány

IoT brána je napsaná v jazyce Python, a proto je její instalace jednodušší než v případě samotné platformy. Opět je dostupných více[27] možností instalace, kdy většinu práce odvede instalační balíček. V rámci této práce ovšem nastaly problémy, a to takové, že pokud bylo potřeba uvolnit danou prioritu objektu v zařízení, musela se poslat hodnota typu *Null*, ale tato hodnota byla přiřazována do datového typu knihovny, který ji nedokázal zpracovat. S níže popisovanou úpravou kódu se tento požadavek zpracovává správně a hodnota se dostane až k danému zařízení, kde uvolní zvolenou prioritu. Upravený kód je již dostupný v master branchi na GitHubu[28], ale nevyšel ještě v nové verzi. Proto pokud se jedná o verzi brány 3.0.1 a nižší, je možnost provést instalaci doporučeným způsobem a upravit části kódu, které se změnily, anebo stáhnout aktuální změny z master branche a nainstalovat vše manuálně.

Kvůli testování v reálném nasazení, úpravě zdrojového kódu a ověření funkčnosti v rámci zadání bakalářské práce byl zvolen postup instalace na zařízení Raspberry Pi (pro použití na systému Windows nebo MacOS je dostupný Docker). Utilitou *wget* se stáhne poslední vydání brány v *deb* balíčku. Balíček se následně nainstaluje příkazem *sudo apt install ./python3-thingsboard-gateway.deb -y*, kdy tečka s lomítkem znamená spuštění balíčku a *-y* automatické odsouhlasení všech dotazů v průběhu instalace.

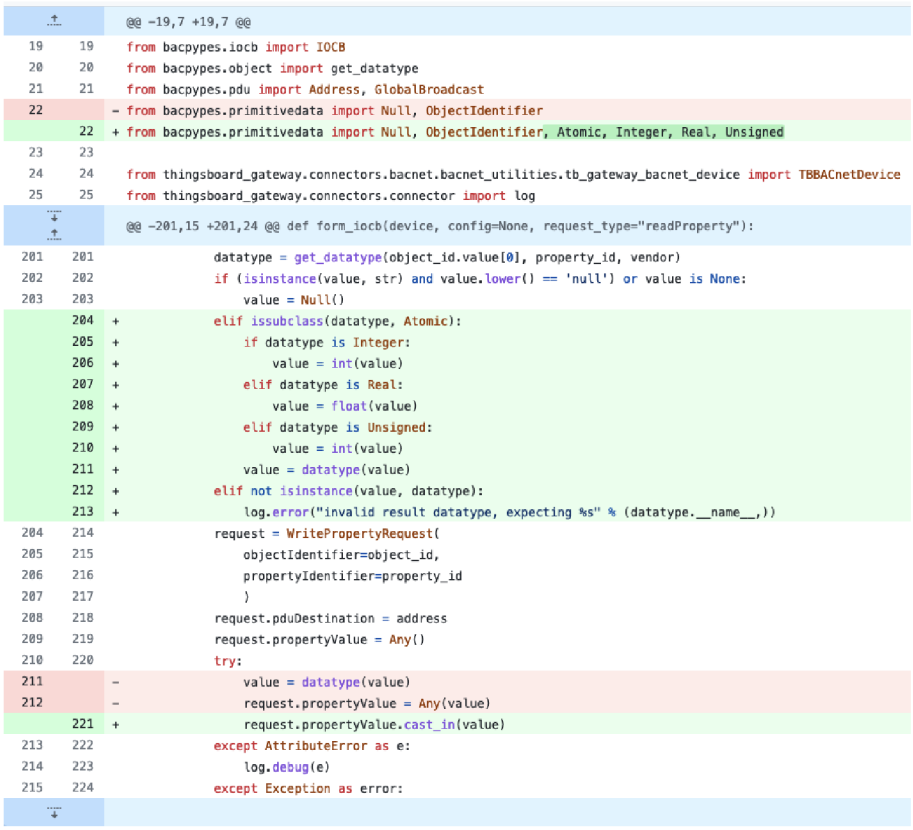
Úspěšnou instalací byly vytvořeny nové složky pro logy, konfiguraci a vlastní navržené konektory protokolů. Konfigurace se nachází v umístění */etc/thingsboard-gateway/config*.

Následuje nutná úprava kódu pro funkční komunikaci s prioritami ve složce, kde se nacházejí distribuované balíčky Pythonu. Cesta ke složce může být odlišná podle nastavení systému. Pro zobrazení definovaných cest lze použít příkaz *python3 -m site*, následně stačí zjistit, jestli se ve složce *dist-packages* nachází složka *thingsboard-gateway*. Pokud ano, pak je ve složce *connectors/bacnet/bacnet-utilities/* soubor s názvem *tb_gateway_bacnet_application.py*, který je třeba upravit. Na řádku 22 byly naimportovány další datové typy *Atomic, Integer, Real a Unsigned* používající se pro porovnávání přijaté hodnoty. Porovnávání prošlo změnou také. Na řádku 204 přibyly další podmínky pro zpracovávání přijaté hodnoty a následné přiřazování

[27] Více na https://thingsboard.io/docs/iot-gateway/installation/

[28] GitHub repositář na https://github.com/thingsboard/thingsboard-gateway

do datového typu knihovny *bacpypes*. Pokud zadaný datový typ neodpovídá, vypíše se chyba. Na řádku 221 je hodnota zařazena do *requestu*. Pokud se ve složce nachází také složka *__pycache__*, je lepší ji smazat pro zajištění, že se při příštím spuštění brány soubory znovu zkompilují. Celkový přehled úprav demonstruje Obrázek 5.3.



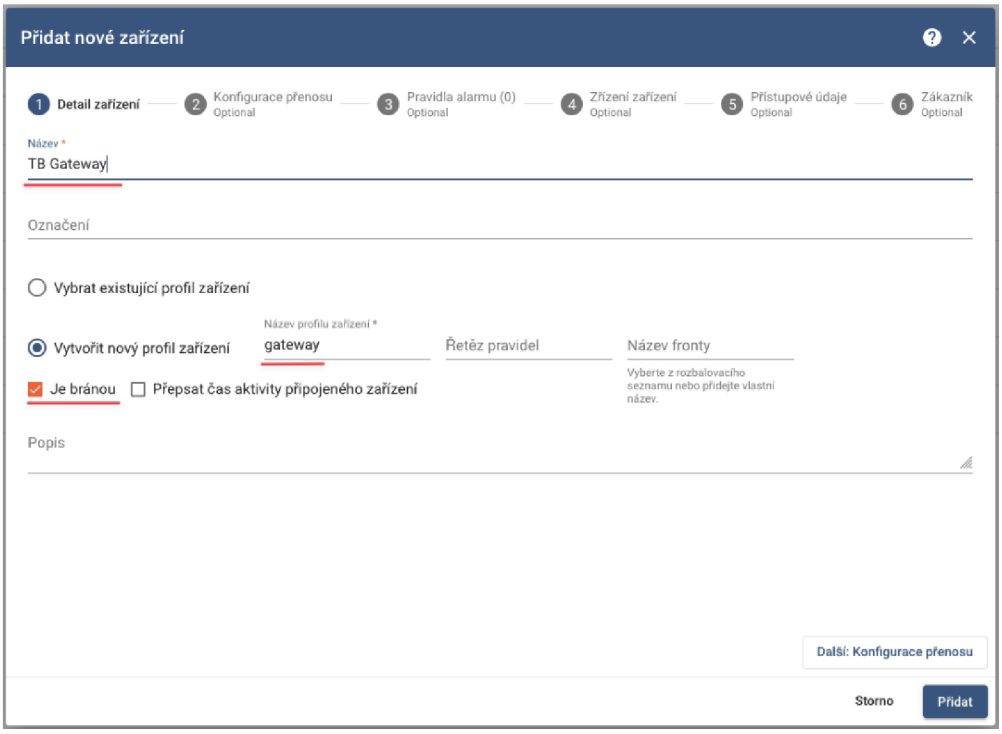
```
@@ -19,7 +19,7 @@
19  19  from bacpypes.iocb import IOCB
20  20  from bacpypes.object import get_datatype
21  21  from bacpypes.pdu import Address, GlobalBroadcast
22      - from bacpypes.primitivedata import Null, ObjectIdentifier
    22  + from bacpypes.primitivedata import Null, ObjectIdentifier, Atomic, Integer, Real, Unsigned
23  23
24  24  from thingsboard_gateway.connectors.bacnet.bacnet_utilities.tb_gateway_bacnet_device import TBBACnetDevice
25  25  from thingsboard_gateway.connectors.connector import log
@@ -201,15 +201,24 @@ def form_iocb(device, config=None, request_type="readProperty"):
201 201              datatype = get_datatype(object_id.value[0], property_id, vendor)
202 202              if (isinstance(value, str) and value.lower() == 'null') or value is None:
203 203                  value = Null()
    204 +            elif issubclass(datatype, Atomic):
    205 +                if datatype is Integer:
    206 +                    value = int(value)
    207 +                elif datatype is Real:
    208 +                    value = float(value)
    209 +                elif datatype is Unsigned:
    210 +                    value = int(value)
    211 +                value = datatype(value)
    212 +            elif not isinstance(value, datatype):
    213 +                log.error("invalid result datatype, expecting %s" % (datatype.__name__,))
204 214              request = WritePropertyRequest(
205 215                  objectIdentifier=object_id,
206 216                  propertyIdentifier=property_id
207 217                  )
208 218              request.pduDestination = address
209 219              request.propertyValue = Any()
210 220              try:
211     -                value = datatype(value)
212     -                request.propertyValue = Any(value)
    221 +                request.propertyValue.cast_in(value)
213 222              except AttributeError as e:
214 223                  log.debug(e)
215 224              except Exception as error:
```

Obrázek 5.3 Změny ve zdrojovém kódu brány

Popisovaná oprava byla zásadní pro následné navrhování a testování platformy ThingsBoard, jejíchž funkce byla doposud značně omezená.

## 5.3 Konfigurace brány

Prvním krokem spojení brány s platformou je vytvoření nového zařízení po přihlášení emailem **tenant@thingsboard.org** a heslem **tenant** přes adresu **http://localhost:8080** v prohlížeči. Vše se nachází v záložce zařízení, kde se zobrazují i další entity v sítí (včetně již předpřipravených ukázkových zařízení), pokud navážou spojení s bránou. Bránu je možné pojmenovat jakkoliv, například TB Gateway, důležité je ovšem zaškrtnutí políčka **Je bránou** a vybrání, případně vytvoření profilu typu **gateway** (tento profil zobrazí bránu v dashboardu *Gateways*, kde je dostupný celkový přehled o provozu brány včetně logů a vzdálené konfigurace). Po vytvoření a kliknutí na detail brány je potřeba zkopírovat přístupový token, který se vloží do atributu *accessToken* v souboru gateway.yaml (soubor v konfigurační složce).




Obrázek 5.4 Přidání brány do platformy ThingsBoard

Dále konfigurační soubor umožňuje **nastavení potřebných konektorů** pro různé protokoly, kde pro účel této práce zůstane aktivní pouze BACnet konektor, **nastavení typu úložiště**, přičemž úložiště v paměti je dostačující a je doporučena alespoň hodnota 1000 v atributu *read_records_count*, **nastavení IP adresy** atributu *host*, kterému náleží adresa brány v síti (adresa *interface* od zařízení jenž provozuje tuto bránu) a v neposlední řadě **povolení vzdálené konfigurace** atributem *remoteConfiguration* hodnotou *true*. Vzdálená konfigurace přispívá k pohodlnějšímu testování při tvorbě *dashboardu* a nastavování brány.

```yaml
thingsboard:
  host: 10.0.0.34
  port: 1883
  remoteShell: false
  remoteConfiguration: true
  statsSendPeriodInSeconds: 3600
  minPackSendDelayMS: 0
  checkConnectorsConfigurationInSeconds: 60
  handleDeviceRenaming: true
  checkingDeviceActivity:
    checkDeviceInactivity: false
    inactivityTimeoutSeconds: 120
    inactivityCheckPeriodSeconds: 10
  security:
    accessToken: C6YEv1sF74xfJv2gUnsj
  qos: 1
storage:
  type: memory
  read_records_count: 1000
  max_records_count: 100000
```

Obrázek 5.5 Ukázka konfigurace parametrů brány

```yaml
#    name: CAN Connector
#    type: can
#    configuration: can.json
#
  -
    name: BACnet Connector
    type: bacnet
    configuration: bacnet.json
#
# -
#   name: ODBC Connector
#   type: odbc
#   configuration: odbc.json
```

Obrázek 5.6 Ukázka povolování konektoru brány

V tomto stavu je vše připraveno k použití, zbývá nastavit atributy pro používaná zařízení. K tomu slouží soubor *bacnet.json*. Atribut *address* obsahuje IP adresu zařízení i s portem, *pollPeriod* definuje periodické vyčítání z *timeseries*. Pole *attributes* definuje pouze atributy zařízení, které se neukládají do databáze pro pozdější využití. Do databáze jsou ukládány pouze atributy pole *timeseries*. Pole *attributeUpdates* předává sdílené atributy platformy zařízení a *serverSideRpc* náleží ovládacím prvkům z dashboardu nebo pravidlům.

Pro zavedení úprav je nutné bránu restartovat přes terminál příkazem *systemctl restart thingsborad-gateway* (příkaz platný pro systém Linux). Pokud je vše správně nastaveno a simulátor místností běží, automaticky se objeví v zařízeních platformy ThingsBoard.

```
{
  "method": "r1_get_radiator_state",
  "requestType": "readProperty",
  "requestTimeout": 10000,
  "objectId": "binaryInput:10",
  "propertyId": "presentValue"
},
{
  "method": "r1_set_temperature_prior_8",
  "requestType": "writeProperty",
  "requestTimeout": 10000,
  "objectId": "analogOutput:10",
  "propertyId": "presentValue",
  "priority": 8
},
```

Obrázek 5.8 Ukázka konfigurace serverSideRpc metod

```
{
  "key": "r1_radiator_state",
  "type": "bool",
  "objectId": "binaryInput:10",
  "propertyId": "presentValue"
},
{
  "key": "r1_manual_control",
  "type": "bool",
  "objectId": "analogOutput:10",
  "propertyId": "priorityArray",
  "propertyIndex": 8
},
```

Obrázek 5.7 Ukázka konfigurace timeseries atributů

Dokumentace postrádá zmínku důležitých a v této práci používaných možností nastavení objektů v jednotlivých kategoriích. V atributu *propertyId* je občas výhodné použít místo hodnoty *presentValue* hodnotu *priorityArray* s přidáním dalšího atributu *propertyIndex* udávajícího číslo priority v prioritním poli. S tímto nastavením brána dokáže vyčíst hodnotu z objektu na dané prioritě a vrátit tak hodnotu nebo prázdné pole (využitelné například pro zjištění, jestli je zařízení v manuálním nebo automatickém režimu, což vyjadřuje priorita číslo 8). Dále nikde není zmínka o atributu *priority* pro *serverSideRpc* požadavky umožňující zápis hodnoty na vybranou prioritu.

Kompletní soubor s nastavím brány a BACnet konektorem pro použití se simulátorem je dostupný v příloze práce, viz příloha A.2 (stačí upravit accessToken a IP adresu brány včetně simulovaného zařízení s portem definovanou při spouštění simulátoru).

## 5.4 Tvorba Dashboardu

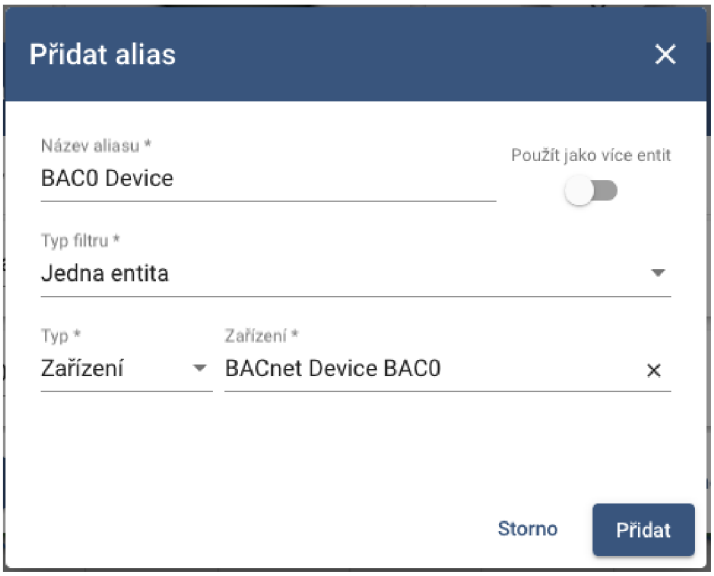
Dashboard poskytuje prostředí pro ovládání a zobrazování dat z různých zařízení. Není pravidlem, že by na jednom dashboardu muselo být pouze jedno zařízení, vše se odvíjí od počtu vytvořených aliasů.

### 5.4.1 Základní nastavení

Nový dashboard se definuje v záložce *Dashboardy*. Po vytvoření a zobrazení je dashboard úplně prázdný. Nabízí se zde dostatečné množství widgetů pro různé použití, od ovládání až po úpravu atributů, zobrazování zařízení na mapě nebo zobrazování aktuálních dat. Existuje také možnost widgety upravovat (v nastavení widgetu) a navrhovat své vlastní (v záložce Knihovna widgetů).

Pro začátek je vhodné dashboard přizpůsobit potřebám a nastavit aliasy zařízení, které se budou následně používat ve všech widgetech. Po rozkliknutí tužky v pravém spodním rohu vyjedou možnosti pro přidání nového widgetu a zároveň se zobrazí další možnosti v horní liště dashboardu, kde vlevo lze nastavit rozmístění a rozdělit dashboard na dvě poloviny (využitelné pro větší obrazovky) a změnit počet sloupců (po rozkliknutí hlavního nebo pravého panelu) pro lepší rozložení widgetů viz Obrázek 5.9. Na pravé straně lišty je důležitější nastavení, a to hlavně potřebných aliasů. Alias může mít jakékoliv jméno a může se skládat z jedné nebo více entit, přičemž pro účely této práce stačí jeden alias pro simulátor a jeden pro bránu, viz Obrázek 5.10. Další důležitý prvek je nastavení časového okna pro optimální zobrazení hodnot na grafech.


Obrázek 5.9 Lišta s nastavením v platformě ThingsBoard




Obrázek 5.10 Vytváření aliasu v platformě ThingsBoard

### 5.4.2 Ovládání simulátoru

Po základním nastavení je dashboard připraven k osazení widgety. Požadovaná teplota místností simulátoru je nastavována pomocí dvou *Knob Control* pod prioritou číslo 8 nacházejících se v ovládacích widgetech. K vyplnění je zde cílové zařízení, což je alias zařízení ze simulátoru, popisek, minimální a maximální hodnota a pole pro *RPC requesty*. První místnost má *get* hodnotu nastavenou na *r1_get_temperature*, *set* hodnotu na *r1_set_temperature_prior_8*. Druhá místnost má zaměněné *r1* za *r2*. K dispozici je také kolonka *RPC request perstistent*, při jejíž aktivaci jsou *get requesty* posílány periodicky pro zobrazování aktuální teploty na widgetu v případě, že by teplota byla změněna z jiného místa.

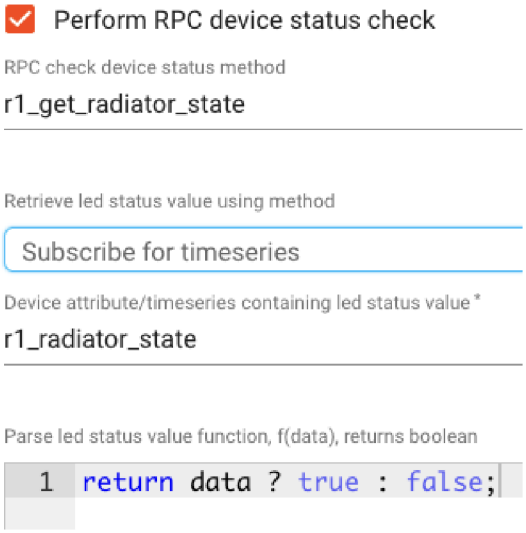

Obrázek 5.11 Widgety pro nastavování teploty

Další částí jsou čtyři indikátory a dva přepínače (opět kategorie ovládací widgety). Dva indikátory ukazují, zda místnosti topí, další dva, jestli je požadovaná teplota v manuálním režimu (nastavena pomocí *Knob Control* widgetů) a přepínače povolují topení.

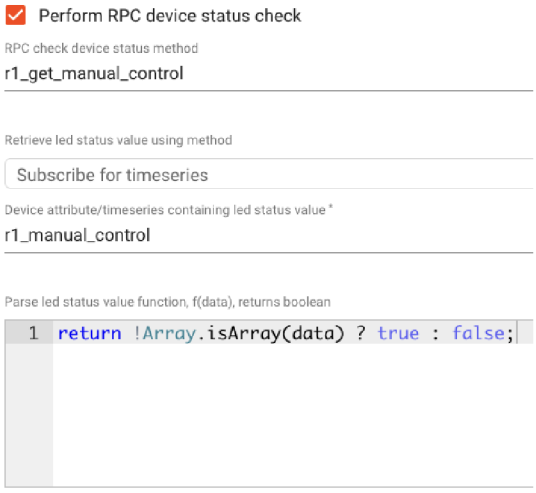
Indikace topení používá při prvotním načtení dashboardu *RPC request* pro zjištění aktuální hodnoty, později je vázaná na *timeseries* atribut. Do *RPC requestu* patří *r1_get_radiator_state*, do *timeseries* atribut *r1_radiator_state*. *Retrieve led status value using method* musí být nastaveno na *Subscribe for timeseries*. Indikace manuálního režimu používá *r1_get_manual_control RPC request* a *r1_manual_control* timeseries atribut a nastává zde důležitá změna pro zpracování již zmiňovaného prioritního pole. Podmínka vracející *true/false* musí být upravena na tvar

```
return !Array.isArray(data) ? true : false;
```

kde pokud je hodnota typu pole, vrátí se *false*, což znamená, že hodnota v prioritním poli pod prioritou osm je *Null*, tedy není zapsána.

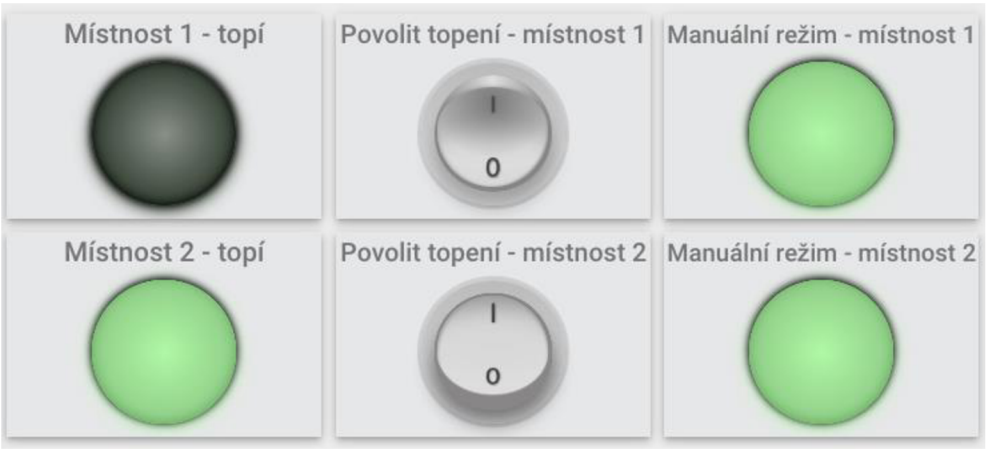



Obrázek 5.13 Nastavení indikace topení




Obrázek 5.12 Nastavení indikace manuálního režimu

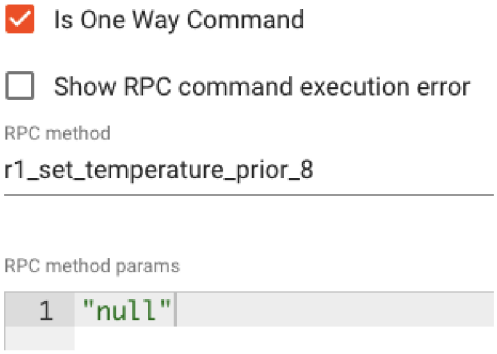
Přepínače topení používají jenom *RPC requesty*, a to *r1_get_heating_state* pro zjištění stavu topení a *r1_set_heating_state* pro nastavení stavu. Je možnost použít timeseries atribut pro periodické zjišťování stavu, ale v tomto případě se může stát, že se přepínač po stisknutí vrátí do předchozího stavu a ukáže špatný stav do doby, než se timeseries opět obnoví, proto je vhodnější použít opět periodické zasílání RPC requestu.




Obrázek 5.14 Ukázka vzhledu indikace a ovládání topení

Poslední ovládací prvky jsou tlačítka zvané *RPC Button* pro uvolňování manuálního režimu teploty pod prioritou osm. Metoda RPC requestu je stejná jako pro nastavování teploty, tedy *r1_set_temperature_prior_8*. Rozdíl mezi nastavováním teploty a uvolňování priority je v posílané hodnotě, kdy pro teplotu je parametrem číselná hodnota, pro uvolňování priority je hodnota *null* v uvozovkách jako textový řetězec, viz Obrázek 5.16.


Obrázek 5.16 Vzhled tlačítek pro uvolňování priorit

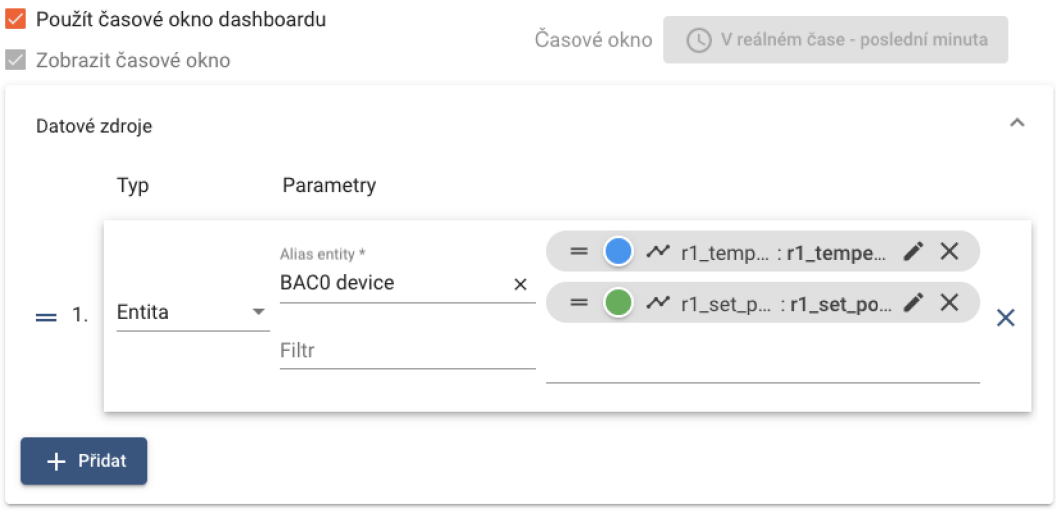



Obrázek 5.15 Konfigurace tlačítek pro uvolňování priorit

Atributy a metody v první a druhé místnosti se opět odlišují pouze v označení *r1* a *r2*, jak u indikátorů, tak u přepínačů.

### 5.4.3 Průběhy teplot

Grafy v této práci zobrazují pouze průběh požadované a aktuální teploty v obou místnostech. Jedná se o *Timeseries Line Chart* grafy, kde vstupními daty jsou takzvané datové zdroje. Typ parametru je Entita, alias entity je alias simulátoru a časové řady jsou atributy *r1_temperature* a *r1_setpoint* (obdobně pro druhou místnost s *r2*).




Obrázek 5.17 Konfigurace grafů pro teploty

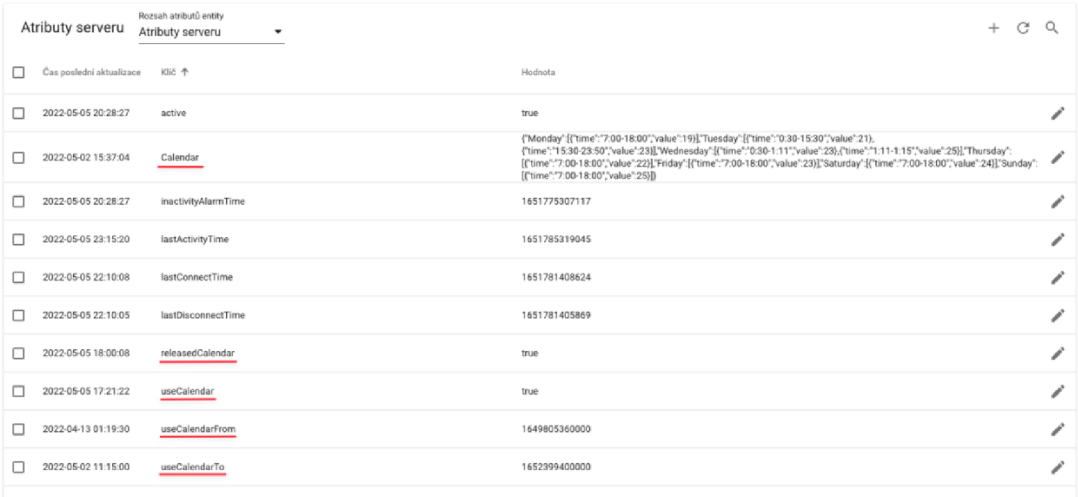
## 5.5 Vlastní teplotní kalendář

V BACnetu existují objekty nesoucí kalendáře navázané na další objekty podle výběru. Tyto kalendáře podle data a času upravují aktuální hodnoty na přiřazené prioritě, většinou se jedná o prioritu 15, tedy druhou nejnižší.

Kalendář vytvořený v platformě ThingsBoard je podobného způsobu, podle aktuálního dne a času upravuje požadovanou teplotu v místnosti na čtrnácté prioritě. Zároveň takové provedení kalendáře demonstruje použití řetězů pravidel, kterými ThingsBoard disponuje a v určitých případech eliminuje nutnost zasahovat do konfigurace reálného zařízení, což přispívá k pohodlnosti ovládání, ale ne vždy je tento typ řešení ideální.

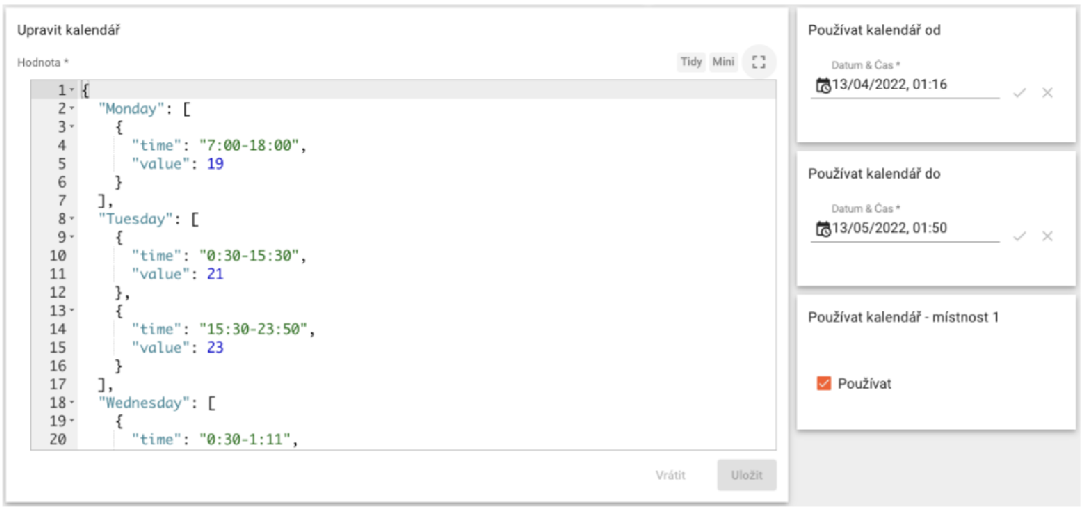
### 5.5.1 Serverové atributy zařízení

Pro použití kalendáře je třeba v detailu zařízení nadefinovat předem dané nové atributy používají se ve skriptech v řetězci pravidel. V detailu zařízení je záložka Atributy s možností přepnutí typu. Nově vytvořené atributy spadají do atributů serveru. Atribut *Calendar* typu JSON obsahuje souhrn dnů a hodnot s možností rozdělení každého dnu do více intervalů. Další dva atributy *useCalendar* a *releasedCalendar* nesou pravdivostní hodnotu, tedy *true* a *false*. Již z jejich názvu vyplývá, že první atribut zapíná a vypíná kalendář, druhý atribut je jako zpětná vazba uvolnění priority pro řetězy pravidel, pokud se kalendář nepoužívá. Poslední atributy jsou číselné *useCalendarFrom* a *useCalendarTo* definující interval platnosti kalendáře (například od začátku do konce měsíce).




Obrázek 5.18 Přehled vytvořených atributů kalendáře

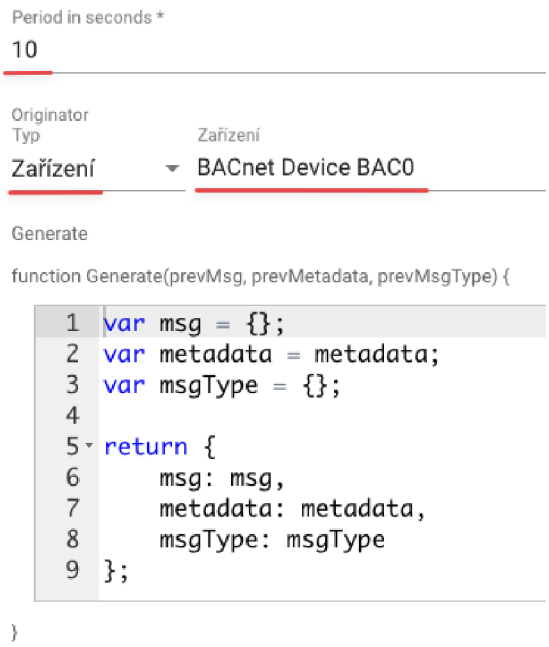
Tyto atributy je možno opět měnit z dashboardu, kdy pro *Calendar* existuje widget typu *Update JSON attribute*, pro zapínání a vypínání kalendáře widget *Update server boolean attribute* a pro určování intervalu platnosti je k dispozici widget *Update server date attribute*. Je důležité si všimnout stylu zápisu JSONu a jak jsou *key – value* páry definovány. Tento styl zápisu je nutné dodržet, opak způsobí nefunkčnost skriptu zpracovávajícího tento JSON. Každý den je pojmenován anglicky, přičemž obsahuje pole objektů. V každém objektu je zapsán časový interval (musí obsahovat i minuty) a použitá *hodnota*, viz příloha A.5.



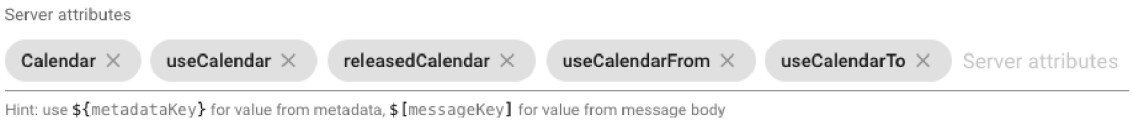

Obrázek 5.19 Ukázka nastavení kalendáře z dashboardu

### 5.5.2 Řetězec pravidel

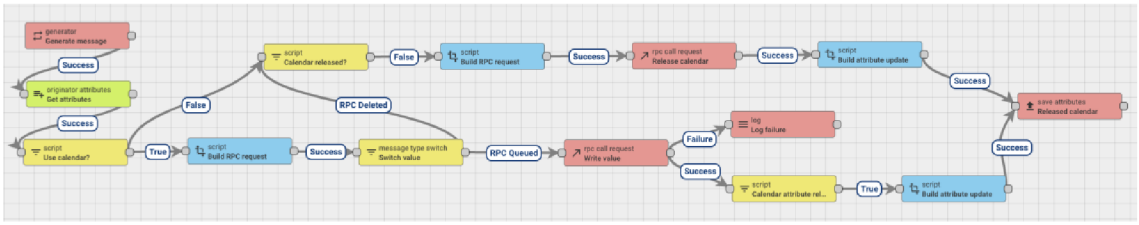
Řetězec na základě zpracování všech nastavených atributů periodicky ovládá celý proces kalendáře. Vše začíná generátorem posílajícím zprávy pod označením zařízení simulátoru. Pokud je původ zpráv nastaven na jinačí nebo žádné zařízení, ovládání nefunguje (rozpadá se vazba *serverSideRpc* metod v nastavení brány). Uzel *originator attributes* váže do metadat zprávy dříve vytvořené atributy kalendáře. Za těmito uzly se již nachází celá logika ovládání kalendáře.




Obrázek 5.20 Konfigurace generátoru zpráv




Obrázek 5.21 Konfigurace navazování atributů do metadat




Obrázek 5.22 Výsledné provedení kalendáře v pravidlech

První žlutý *script* uzel porovnává *useCalendar*, *useCalendarFrom* a *useCalendarTo* atributy. Pokud je kalendář zapnut a odpovídá interval platnosti, pokračuje se zpracováním samotného JSON kalendáře. Pokud podmínka splněna není, zkontroluje se, zda je priorita 14 používaná kalendářem uvolněná, jestliže není, pošle se *RPC request* s hodnotou *NULL* pro uvolnění priority. Úspěšný request nastaví *releasedCalendar* atribut do hodnoty *true*. Tímto je celý cyklus kalendáře vykonán, nedochází k posílání dalších požadavků.

Pokud dojde ke splnění podmínky prvního žlutého *script* uzlu, modrý *script* uzel zpracuje přijatý JSON a podle nastavených denních intervalů pošle buďto požadovanou hodnotu s označením *RPC Queued*, která se následně odešle jako *RPC request* (pokud je požadavek úspěšný, nastaví se *releasedCalendar* atribut do hodnoty *false*) nebo pošle hodnotu *NULL* s označením *RPC Deleted*, přičemž následné chování je stejné jako u nesplnění podmínky použití kalendáře.

Podle typu instalace je nutno zkontrolovat čas používaný těmito pravidly, na zařízení Raspberry Pi odpovídá realitě, Docker může mít čas posunutý o dvě hodiny zpět (podle zimního a letního času). Úprava je použita pouze v prvním žlutém a modrém *script* uzlu, viz úryvek zdrojového kódu níže.

```
date.setHours(date.getHours() + 2)
```

Pravidla jsou vyexportována stejně jako dashboard v příloze práce, stačí je naimportovat a patřičně upravit (viz příloha A.3 a A.4).

# 6. ZHODNOCENÍ REÁLNÉHO NASAZENÍ

Vybraná platforma ThingsBoard je široce používaným nástrojem pro monitorování dat a ovládání zařízení. Samotná dokumentace poukazuje na segmenty využití, například v odvětví potravinářství monitoruje mrazáky pro správnou kvalitu potravin, v kancelářích kontroluje správnou kvalitu ovzduší, v segmentu zemědělství hlídá kvalitu půdy. Všechny tyto příklady lze realizovat stejným způsobem přes protokol BACnet, za předpokladu, že jej všechny použité komponenty podporují. Pokud jej nepodporují, existují další možnosti připojení, jako například protokol MQTT.

V případě této práce byla celá platforma použita v reálném nasazení, kde řídila světelnou techniku, vzduchotechniku a zobrazovala vnitřní a venkovní stav ovzduší bez sebemenšího problému a s rychlou odezvou.

Platforma v kombinaci s PostgreSQL databází a bránou byla provozována na počítači Raspberry Pi 4 Model B, který disponuje dostatečným hardwarem pro použití v takové míře (dokumentace ukazuje dokonce instalaci na starší Raspberry Pi 3 Model B). Instalace je na Raspberry z části omezena, nelze si zvolit z výkonnějších kombinací databází (PostgreSQL + Cassandra nebo PostgreSQL + TimescaleDB) a frontových služeb (Kafka a RabbitMQ). Pro slabší verze Raspberry je zde možnost omezit využití operační paměti až na 256 MB (což samo o sobě hovoří o náročnosti platformy).

Konfigurace brány nabízí stejné možnosti jako v případě instalace na jakýkoliv jiný systém, přičemž pokud je na disku málo místa, což u Raspberry s použitím SD karty může nastat (nebo je žádoucí vyhnout se dalším zápisům pro prodloužení životnosti karty), lze nastavit ukládání přijatých dat do operační paměti (používá se jako výchozí nastavení). Pro větší rychlost je také možné použít SQLite databázi.

# ZÁVĚR

Cílem této práce bylo seznámit se se základy protokolu BACnet a nástroji na tvorbu uživatelských webových rozhraní pro vizualizaci a ovládání zařízení, včetně návrhu simulovaného zařízení a následné otestování jejich vzájemné funkcionality. Posledním bodem bylo zhodnocení nasazení vybraného nástroje v reálném řízení budovy. Tyto cíle se podařilo splnit, včetně otestování funkcionalit a návrhu řešení na případná úskalí.

Z hlediska kompatibility a jednoduchosti vychází nejlépe platforma ThingsBoard. Díky průmyslovému zaměření v kombinaci s vlastní IoT bránou je konfigurace zařízení velice intuitivní a přímočará. Všechny správně nakonfigurované zařízení se automaticky objevují v platformě a je možno s nimi ihned začít pracovat, zobrazovat naměřená data a ovládat je. Thingsboard byl použit pro demonstraci použitelnosti v kombinaci s BACnet protokolem a simulátorem místností, kde je nutno vyzdvihnout vlastně řešený kalendář pro automatické ovládání zařízení. Placená verze platformy disponuje svou možností tvorby kalendářů, která je poněkud pohodlnější. V průběhu návrhu se objevily komplikace s dokumentací a použitelností priorit. Chyba s prioritami byla opravena a publikována na GitHub repositář od IoT brány. Druhá chyba byla při použití nástroje Docker, kdy se opětovně nespouštěla databáze a platforma vůbec nefungovala. Správný postup byl uveden při popisu instalace. Chybějící informace z dokumentace pro konfiguraci brány byly vyčteny z kódu a následně aplikovány.

OpenHAB platforma je primárně určená na ovládání chytrých domů, s čímž se pojí její nepřeberné množství dostupných připojení všemožných existujících systémů a zařízení. V případě BACnet protokolu tato možnost připojení chybí, a proto je třeba použití brány mezi MQTT a BACnetem od třetí strany.

Node-RED zastává funkci zcela jiné aplikace, než ThingsBoard a openHAB, ale díky dostupným balíčkům pro tvorbu dashboardu a komunikaci po BACnet protokolu zde najde podobné uplatnění, konfigurace ale není tak dokonalá jako v případě ThingsBoard.

Z kategorie vybraných simulátorů je nejvíce flexibilní knihovna BAC0. Díky dostupnosti zdrojového kódu a rozsáhlé dokumentaci byla použita pro simulaci dvou oddělených místností s vlastním topením a přechody tepla zvenčí a mezi místnostmi. Rychlost simulace lze měnit pomocí vstupního parametru. Předpokladem pro běh simulátoru je dostupnost souvisejících knihoven.

Komerční simulátor je také velice dobře použitelný, a to jak z hlediska jednoduchosti konfigurace zařízení, tak z použití v průmyslu. Nevýhodou je omezení simulace na konkrétní hodnoty objektů, z čehož plyne, že v něm nelze navrhovat zařízení jako v případě knihovny BAC0.

V reálném systému řízení budovy je kombinace platformy ThingsBoard a počítače Raspberry Pi dostačující. Platforma byla na počítači řádně otestována fungovala bez sebemenších problémů. Pokud hardware počítače není dostatečný, lze částečně upravit výchozí požadavky platformy.

# LITERATURA


[1] MERZ, Hermann, Thomas HANSEMANN a Christof HÜBNER. *Automatizované systémy budov: sdělovací systémy KNX/EIB, LON a BACnet*. Praha: Grada, 2008. Stavitel. ISBN 978-80-247-2367-9.

[2] THINGSBOARD. What is ThingsBoard? *ThingsBoard* [online]. 2017 [cit. 2021-12-21]. Dostupné z: https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/

[3] THINGSBOARD, 2022. GitHub - thingsboard/thingsboard-gateway: Open-source IoT Gateway - integrates devices connected to legacy and third-party systems with ThingsBoard IoT Platform using Modbus, CAN bus, BACnet, BLE, OPC-UA, MQTT, ODBC and REST protocols. *GitHub* [online] [cit. 2022-05-13]. Dostupné z: https://github.com/thingsboard/thingsboard-gateway

[4] openHAB Foundation e.V. Introduction. *Openhab.org* [online]. 2021 [cit. 2021-12-22]. Dostupné z: https://www.openhab.org/docs/

[5] Anon., 2022. Adding Things - Advanced. *Openhab.org* [online] [cit. 2022-05-13]. Dostupné z: https://www.openhab.org/docs/tutorial/things_advanced.html

[6] Anon., 2022. Rules. *Openhab.org* [online] [cit. 2022-05-13]. Dostupné z: https://www.openhab.org/docs/configuration/rules-dsl.html

[7] OpenJS Foundration & Contributors. About : Node-RED. *Nodered.org* [online]. 2013 [cit. 2021-12-23]. Dostupné z: https://nodered.org/about/

[8] Christian Tremblay, P.Eng. Welcome to BACØ – BACnet Test Tool — BAC0 documentation. *Readthedocs.io* [online]. 2021 [cit. 2021-12-27]. Dostupné z: https://bac0.readthedocs.io/en/latest/index.html

[9] Softdel Systems Pvt. Ltd. BOSS Simulator | BACnet Device Simulator | BACnet/IP Simulation. *Softdel* [online]. 2019 [cit. 2021-12-28]. Dostupné z: https://www.softdel.com/bacnet-device-simulator/

# SEZNAM PŘÍLOH

# Příloha A - Použité konfigurace

## A.1 Skript simulátoru

- Soubor uložen na přiloženém CD

## A.2 Konfigurační soubory brány

- Soubory uloženy na přiloženém CD

## A.3 Dashboard

- Soubor uložen na přiloženém CD

## A.4 Řetězec pravidel kalendáře

- Soubor uložen na přiloženém CD

## A.5 JSON pro kalendář

```
{
  "Monday": [
    {
      "time": "7:00-18:00",
      "value": 19
    }
  ],
  "Tuesday": [
    {
      "time": "0:30-15:30",
      "value": 21
    },
    {
      "time": "15:30-23:50",
      "value": 23
    }
  ],
  "Wednesday": [
    {
      "time": "0:30-1:11",
      "value": 23
    },
    {
      "time": "1:11-1:15",
      "value": 25
```

```
    }
  ],
  "Thursday": [
    {
      "time": "7:00-18:00",
      "value": 22
    }
  ],
  "Friday": [
    {
      "time": "7:00-18:00",
      "value": 23
    }
  ],
  "Saturday": [
    {
      "time": "7:00-18:00",
      "value": 24
    }
  ],
  "Sunday": [
    {
      "time": "7:00-18:00",
      "value": 25
    }
  ]
}
```