



**MATEMATICKO-FYZIKÁLNÍ
FAKULTA**
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Jan Kytka

System pro automatickou regulaci v domácnosti

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. David Obdržálek, Ph.D.

Studijní program: Informatika

Studijní obor: Informatika se specializací
Systémové programování

Praha 2022

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V dne

Podpis autora

Děkuji RNDr. Davidovi Obdržálkovi, Ph.D. za poskytnutí věcných připomínek a cenného vhledu do problematiky. Za jeho důvěru, trpělivost a čas, který mi věnoval při vedení této práce. Také bych chtěl poděkovat Evgenii a Tomášovi za emocionální podporu. Děkuji i všem, kteří přispívali svými nápady, nebo se podíleli na testování.

Název práce: Systém pro automatickou regulaci v domácnosti

Autor: Jan Kytka

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. David Obdržálek, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: V problematice automatizace domácnosti a obecně řízení na základě dat ze vstupních jednotek existuje mnoho komerčních i hobby řešení. Nabízejí různá paradigmatu zápisu pravidel regulace, jako je popis reakcí na události, propojení pomocí sítě hradel, aj. Analyzovali jsme nedostatky událostmi řízených systémů a na základě nich jsme navrhli a implementovali aplikaci, kde se pravidla popisují explicitním zápisem přechodové funkce. Ukázali jsme, jak v tomto odlišném paradigmatu formulovat složitější regulační úlohy a předběžně ho nechali otestovat malým vzorkem uživatelů. Ti se shodli na tom, že je velmi použitelné, i když obtížnější na pochopení.

Klíčová slova: automatizace výrazy řízení aplikace MQTT IoT ASP.NET

Title: Home automation

Author: Jan Kytka

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. David Obdržálek, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: In the area of home automation and control based on data from input units, there are many commercial and hobby solutions. They offer different paradigms for representing control rules, such as description of trigger conditions and events, gate networks, etc. We analyzed the shortcomings of event-driven systems and, based on them, we designed and implemented an application which represents the rules as explicit transition function expressions. We showed how to formulate more complex control tasks in this different paradigm and had it pre-tested with a small sample of users. They agreed that it is very usable, even if perhaps more difficult to understand.

Keywords: automation expressions control application MQTT IoT ASP.NET

Obsah

| | |
|--|-----------|
| Úvod | 4 |
| Struktura práce | 4 |
| 1 Analýza | 5 |
| 1.1 Problémy událostmi řízených řešení | 5 |
| 1.2 Existující řešení | 6 |
| 1.2.1 Node-RED | 6 |
| 1.2.2 eWeLink | 8 |
| 1.2.3 Google Cloud IoT | 10 |
| 1.3 Diskuse o požadavcích | 10 |
| 1.3.1 Požadavky na komunikaci | 10 |
| 1.3.2 Přejížděcí funkce jako výraz | 10 |
| 1.4 Požadavky | 11 |
| 1.4.1 Funkční požadavky | 11 |
| 1.4.2 Nefunkční požadavky | 11 |
| 2 Návrh aplikace | 12 |
| 2.1 Konceptuální model | 12 |
| 2.2 Volba technologií ovlivňujících návrh | 12 |
| 2.2.1 Message Queuing Telemetry Transport (MQTT) | 12 |
| 2.3 Aplikační vrstva | 13 |
| 2.3.1 Architektura | 13 |
| 2.3.2 Vyhodnocování výrazů | 13 |
| 2.3.3 Ukládání historie | 14 |
| 2.3.4 HTTP rozhraní | 14 |
| 2.4 Databáze | 14 |
| 2.5 Prezentáční vrstva | 14 |
| 2.5.1 Uživatelské rozhraní | 15 |
| 3 Implementace | 17 |
| 3.1 Aplikační vrstva | 17 |
| 3.1.1 Vyhodnocování výrazů | 17 |
| 3.1.2 HTTP rozhraní | 18 |
| 3.2 Databáze | 20 |
| 3.2.1 Schéma | 20 |
| 3.3 Frontend | 22 |
| 3.3.1 Uživatelské rozhraní | 22 |
| 4 Dokumentace | 23 |
| 4.1 Uživatelská dokumentace | 23 |
| 4.2 Instalační příručka | 23 |
| 4.3 Vývojová dokumentace | 23 |
| 4.4 Kód aplikace | 23 |

| | | |
|----------|--|-----------|
| 5 | Evaluace | 24 |
| 5.1 | Splnění požadavků | 24 |
| 5.2 | Uživatelské testování | 24 |
| 5.2.1 | Vzorové řešení úloh | 25 |
| 5.2.2 | Výsledky | 25 |
| 5.3 | Provozní nasazení | 26 |
| 5.3.1 | Implementace ekvitermní regulace | 26 |
| 5.4 | Další rozšíření | 27 |
| | Závěr | 28 |
| | Seznam použitých zdrojů | 29 |
| | Seznam obrázků | 31 |
| | Seznam použitých zkratk | 32 |
| | Přílohy | 33 |
| A | Uživatelská dokumentace | 34 |
| A.1 | O aplikaci | 34 |
| A.2 | Jak funguje MQTT | 34 |
| A.3 | Popis uživatelského rozhraní | 34 |
| A.3.1 | Přístrojová deska | 34 |
| A.3.2 | Nastavení karty | 35 |
| A.3.3 | Seznam zařízení | 35 |
| A.4 | Syntaxe pravidel | 36 |
| A.4.1 | Příklady | 36 |
| A.4.2 | Přehled všech operátorů a funkcí | 37 |
| B | Instalační příručka | 39 |
| B.1 | Požadavky | 39 |
| B.1.1 | MySQL | 39 |
| B.1.2 | MQTT broker | 39 |
| B.1.3 | .NET runtime | 39 |
| B.2 | Instalace IoT-Dash | 39 |
| B.3 | Konfigurace | 39 |
| B.3.1 | Webové rozhraní | 40 |
| B.3.2 | Připojení k databázi | 40 |
| B.3.3 | Nastavení JSON Web Token (JWT) | 40 |
| B.3.4 | Nastavení MQTT | 41 |
| B.3.5 | Nastavení logování | 41 |
| C | Vývojová dokumentace | 43 |
| C.1 | Aplikační vrstva | 43 |
| C.1.1 | Sestavení | 43 |
| C.1.2 | Dokumentace HTTP API | 43 |
| C.2 | Prezentační vrstva | 43 |
| C.2.1 | Sestavení | 43 |
| C.2.2 | Architektura | 44 |

| | | |
|----------|-------------------------------|-----------|
| D | Dotazník pro testování | 46 |
| D.1 | Popis systému | 46 |
| D.2 | Scénáře | 46 |
| D.3 | Dotazník | 47 |
| E | Zdrojový kód aplikace | 48 |

Úvod

V problematice automatizace domácnosti existuje mnoho komerčních i hobby řešení. Nabízejí různá paradigmatu zápisu pravidel regulace, jako je popis reakcí na události, propojení pomocí sítě hradel, aj. (viz 1.2). V takových systémech se stává, že jeden výstup může záviset na několika vstupech. Pro některé uživatele je velké množství závislostí překážkou při snaze odladit nastavení takového systému (viz 1.1).

Při ruční implementaci nějaké regulační úlohy jako programu by programátor zpravidla popisoval přechodové funkce stavů jednotlivých regulovaných prvků. Takový vývoj je ale samozřejmě technicky náročný, pro běžné uživatele nedosažitelný. V naší aplikaci se toto paradigma pokusíme přiblížit i neprogramátorům, a to vývojem doménově specifického jazyka, ve kterém se přechodové funkce dají přímo formulovat.

Motivační úlohou této práce je modernizace stávajícího topného systému rodinného domu. V mé domácnosti se za poslední 2 roky postupně vystřídal několik technologií pro ovládání topení, v posledním roce i zavlažování, a s každým měli uživatelé potíže, které se v této práci snažím řešit.

Nejprve to byla jednoduchá analogová regulace pomocí elektromechanických termostatů a stykačů. Později jsme experimentovali s zařízeními pro internet věcí (IoT) a systémem Node-RED. Ten se ale ukázal jako příliš složitý pro ostatní členy domácnosti. Pak jsme přešli na eWeLink, který se ze začátku zdál ideální díky jednoduchému UI. Časem se ale objevilo, že dalším problémem jsou kolize, které vznikají když více událostí ovlivňuje stav jednoho zařízení (viz 1.1).

Struktura práce

Práce se skládá z pěti kapitol. V kapitole 1 si představíme do detailu problém, který tato práce řeší. Analyzujeme existující řešení a po sléze formulujeme požadavky na aplikaci.

V kapitole 2 na základě těchto požadavků zvolíme některé technologie a vytvoříme návrh nové aplikace.

V třetí kapitole uvedeme technologie a metody, které byly zvoleny během implementace aplikace. Také uvedeme některé důvody k jejich volbě.

V kapitole 4 představíme celkovou dokumentaci tohoto projektu.

V kapitole 5 pak zhodnotíme navrženou aplikaci z hlediska požadavků, které jsme vznesli v výše (viz 1.3). Popíšeme, jak probíhalo uživatelské testování, představíme provozní nasazení, ve kterém byla aplikace vyvíjena, ukážeme příklad složitější regulační úlohy a v poslední řadě uvedeme další možná rozšíření a výhled do budoucnosti.

1. Analýza

V této kapitole přesně formulujeme problém, který budeme řešit, prozkoumáme existující řešení, a nakonec diskutujeme a formulujeme požadavky pro návrh řešení vlastního.

Mějme systém několika zařízení sestávající ze soustavy *čidel a ovládacích prvků*. *Čidlo* je zařízení, které produkuje nějaký číselný nebo kvalitativní údaj, představme si např. půdní nebo atmosferický teploměr či vlhkoměr, tlakoměr, měřič slunečního záření, měřič elektrické spotřeby nebo výkonu solární elektrárny, voltmetr na baterii, pohybové PIR čidlo, ale třeba i spínač nebo tlačítko, měřič hladiny CO₂ aj. *Ovládací prvek* je zařízení, jehož ovládním můžeme mít vliv na skutečný svět. Je to např. topné těleso, ventilátor, čerpadlo, servomotor ovládající okna nebo dveře, elektronicky řízený ventil atp. Takový systém pro účely této práce (a také proto, že slovo „systém“ je už velice přetíženým pojmem) nazývávejme *domácnost*, i když se může jednat o jiné, například industriální nasazení těchto zařízení.

V takové domácnosti zpravidla chceme, aby platily nějaké invarianty. Například chceme, aby teplota v obývacím bytě byla 20 °C, v noci svítilo noční osvětlení, voda na mytí aby měla aspoň 60 °C, a aby — pokud má zrovna solární elektrárna přebytek — se vyrobená el. energie spotřebovala na ohřev vody, protože výkupní cena je dnes velmi nízká, činí zhruba 500 Kč za 1 MWh. Ve srovnání s prodejní cenou je to zhruba desetina [1]. Od aplikace která „automatizuje domácnost“ chceme, aby nám umožnila v nějaké podobě formulovat pravidla, kterými se bude domácnost řídit tak, aby platily kýžené invarianty.

Často budeme popisovat pravidla a invarianty na systému pro vytápění, ale neomezujeme se pouze na něj. Chceme aplikaci, jejíž popis pravidel je dostatečně expresivní, aby se v ní dala popsat pravidla udržující libovolné invarianty.

Jedním takovým složitějším případem regulace je tzv. *ekvitermní vytápění* [2]. Spočívá v ohřátí teploty topného média v závislosti na venkovní teplotě a odhadu tepelných ztrát místnosti. Pro zvolenou teplotu místnosti se určí tzv. *ekvitermní křivka*, která popisuje, jakou teplotu topného média máme přivést do radiátoru, abychom dosáhli zvolené teploty v závislosti na venkovní teplotě.

1.1 Problémy událostmi řízených řešení

V událostmi řízených systémech, kde se kříží mnoho pravidel ovlivňujících stejné výstupy, vznikají konflikty. Komerční řešení navíc často optimalizují pouze první dojem na uživatele a snadnost používání. Uvažme pro příklad nějaký takový systém. Uživatel naivně nastaví tato pravidla:

- (a) „Pokud teplota klesne pod 40 °C, zapni ohřev vody.“
- (b) „Pokud teplota stoupne nad 60 °C, vypni ohřev vody.“

Nyní se systém chová jako běžný termostat s hysterezí a uživatel je s jeho chováním spokojený. Později si ale všimne, že takto systém zbytečně topí během drahé sazby el. energie, a rozhodne se přidat další pravidla týkající se hromadného dálkového ovládní (HDO):

- (c) „Pokud HDO signalizuje levnou sazbu, zapni ohřev vody.“
- (d) „Pokud HDO nesignalizuje levnou sazbu, vypni ohřev vody.“

Nyní se mohou nastat dva scénáře, které si ale díky svobodě, s jakou je možné pravidla zadávat, uživatel vůbec neměl šanci rozmyslet:

1. V 5:55 klesne teplota v bojleru pod 40 °C a začne se topit. V 6:00 ale končí levná sazba, takže se topení vzápětí vypne. V 8:00 se chce uživatel osprchovat a zjistí, že voda v bojleru má 37,5 °C. Vidí, že stav bojleru je v rozporu s pravidlem (a) a není si jistý, co udělal špatně.
2. V 19:55 se bojler ohřeje na 60 °C a ohřev se vypne. Ve 20:00 ale začíná levná sazba, takže se ohřev opět zapne. V 1:00 uživatele probudí uvolnění přetlakového ventilu bojleru a vidí, že teploměr ukazuje teplotu 120 °C. Podobně jako v předchozím scénáři se může zdát, že stav bojleru je v rozporu s pravidlem (b).

Tedy je vidět, že zdánlivá snadnost používání nějakého paradigmatu na popis pravidel nemusí nutně indikovat, že je opravdu snadné toto paradigma používat správně.

1.2 Existující řešení

Z mnoha dostupných řešení pro regulaci (a obecně řízení na základě dat ze vstupních jednotek) vybereme pro srovnání ty, se kterými mám zkušenosti z provozu v reálné domácnosti. Navíc k nim ještě přidáme Google Cloud IoT, protože jde o komplexní řešení připravené na industriální nasazení, nabízející velice širokou paletu služeb, a srovnání s ním by nemělo chybět.

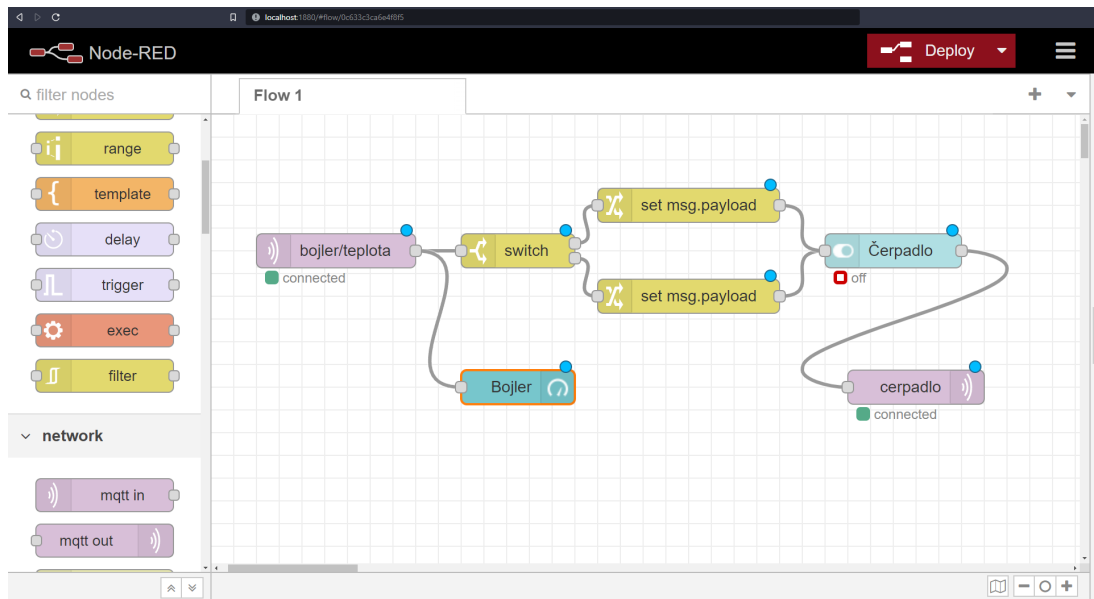
1.2.1 Node-RED

Oficiální dokumentace [3] uvádí následující:

Node-RED je grafický programovací nástroj určený pro propojení zařízení, API a online služeb komunikujících po různých protokolech. Může být spuštěn lokálně na počítači uživatele, nebo na lokálním serveru, na který přistupuje několik uživatelů přes webový prohlížeč. Umožňuje taky spuštění v cloudu pomocí IBM Cloud, AWS, Microsoft Azure a SenseTecníc FRED.

Uživatel v něm prostřednictvím webové aplikace vytváří program, tzv. *flow*, který přijímá a odesílá události. Z palety si uživatel vybere programovatelné bloky, které vzájemně propojí tak aby uvnitř programu vhodně zpracovávaly události (viz obrázek 1.1). Některé bloky pak slouží ke komunikaci za hranice programu, např. MQTT, které můžete vidět vyznačené na obrázku fialovou barvou. Další dostupné jsou MySQL dotaz, práce se soubory, HTTP dotaz, HTTP server atd.

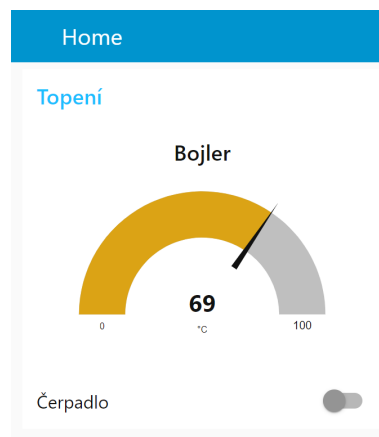
Jak je vidět, tento nástroj je poměrně silný. Pro některé uživatele ale může být příliš složitý. Například pokud chce uživatel vytvořit podmínku složenou ze dvou



Obrázek 1.1: Jednoduchý program pro řízení čerpadla poslouchajícího na MQTT kanálu `cerpadlo` na základě naměřené teploty bojleru vybaveného teploměrem publikujícím data na kanálu `bojler/teplota`.

zdrojů událostí, musí ručně pracovat se stavem bloků, aby si program zapamatoval poslední přijatou hodnotu z obou zdrojů. Jinak se podmínka nedá vyjádřit, protože události z více zdrojů chodí grafem postupně, ne v párech.

Node-RED umožňuje jednoduchou instalaci rozšíření pomocí balíčkovacího systému `npm`. Jedním z nich je i balíček `node-red-dashboard`, který umožňuje vytvořit jednoduchý přehled různých ukazatelů naměřených hodnot [4]. Balíček přidá do serveru HTTP trasu na které je k dispozici přehled jako interaktivní webová stránka (viz obrázek 1.2).



Obrázek 1.2: Přehled vygenerovaný balíčkem `node-red-dashboard` k *flow* z obrázku 1.1.

Tento balíček mimo jiné umožňuje zobrazit graf historie naměřených hodnot. Ten umožňuje pohled na naměřené hodnoty v pevném časovém okně. Při větším množství grafů a bodů v grafu ale dramaticky stoupá odezva celé aplikace, protože každý připojený prohlížeč si z přehledu pokaždé stáhne všechny body v celém časovém okně. Prvek navíc neumožňuje prohlížení za hranice časového okna ani export dat.

Kompatibilní zařízení

Výhodou systému je, že je s ním kompatibilní široká řada zařízení. Interagovat může v podstatě s jakýmkoli zařízením, se kterým může komunikovat počítač. Složitější komunikační protokoly mohou ale být překážkou pro některé uživatele.

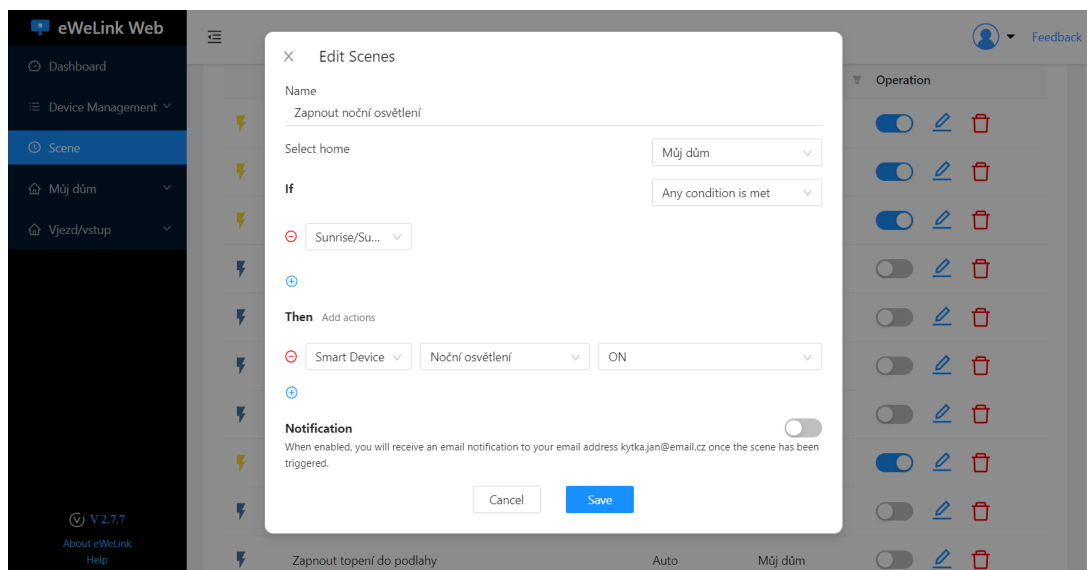
Souhrn

Když vezmeme v potaz například již zmíněnou ekvitermní regulaci, tak tu v systému Node-RED vyjádřit lze.

Není těžké pozorovat, že systém Node-RED je turignovskly úplný, už jen proto, že v paletě nabízí blok, který vykoná libovolný kód zapsaný v jazyce JavaScript. Tato síla s sebou ale přináší ale určité nároky na uživatelskou expertízu. Tedy pokročilé výpočty reprezentovat lze, ale náročností se blížíme programování samotnému.

1.2.2 eWeLink

Aplikace eWeLink je cloudové řešení pro kompatibilní IoT zařízení. Umožňuje jednoduché ovládání, ze zde zmíněných alternativ asi nejjednodušší pro běžného spotřebitele. Uživatel si koupí kompatibilní zařízení, nainstaluje aplikaci na mobil, a může jej uvést do provozu. Aplikace ho všim provede krok za krokem, celý proces trvá cca 5 minut.



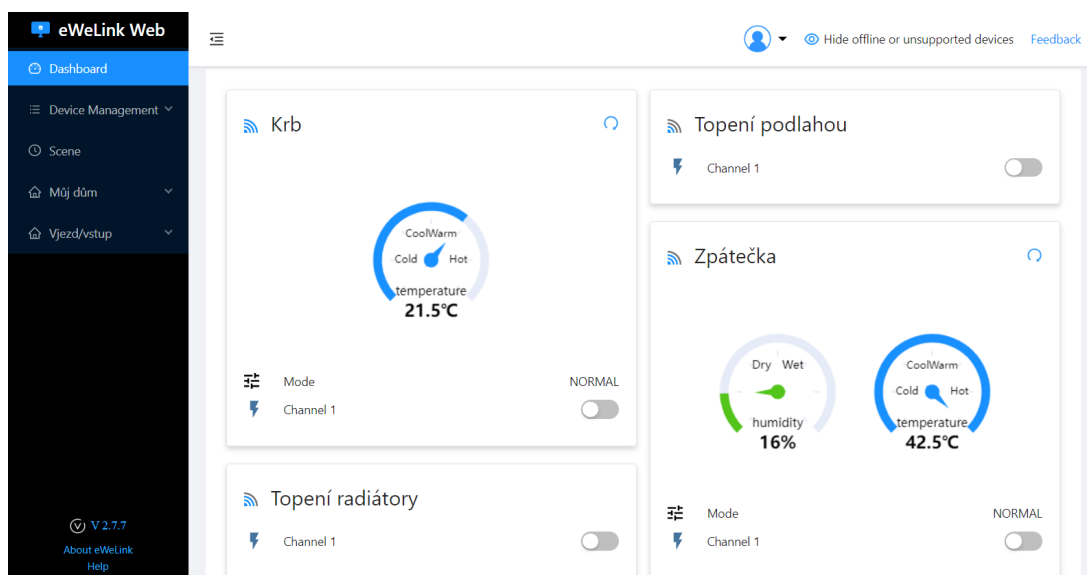
Obrázek 1.3: Aplikace eWeLink, obrazovka s nastavením pravidel regulace.

V systému eWeLink lze vyjádřit jen jednoduchá pravidla formou podmíněných reakcí na události (obrázek 1.3). Podmínky reprezentujeme jako konjunkce nebo disjunkce několika nerovností na naměřených hodnotách v systému. Obecnější podmínky systém neumožňuje.

Uživatel nepotřebuje prakticky žádnou přípravu k používání takového systému. To se nejprve může jevit jako výhoda, ale taky jako promarněná příležitost. Pokud si někdo tento systém v domácnosti nainstaluje, není od věci předpokládat, že bude ochotný investovat nějaký čas do přípravy nebo vzdělání aby mohl takový systém používat naplno.

Kompatibilní zařízení

Mezi kompatibilní zařízení patří například výrobky od firmy SONOFF [5]. Jde o různé teploměry, vlhkoměry a relé, které kromě komunikace s aplikací eWeLink přes internet podporují ovládání přes technologii ZigBee nebo bezplatné pásmo 433 Mhz.



Obrázek 1.4: Aplikace eWeLink, obrazovka s přístrojovou deskou.

Výhody

- Umožňuje snadno vytvořit přístrojovou desku a manuálně z ní ovládat některá zařízení.

Nevýhody

- Poměrně limitované možnosti regulace.
- Vznikají konflikty ve zdánlivě neškodných pravidlech (viz 1.1).
- V bezplatném režimu není možné prohlížet historii naměřených dat. Není možnost lokálního hostování.

1.2.3 Google Cloud IoT

Google Cloud nabízí širokou řadu služeb, mezi nimi jsou i různorodá řešení pro IoT [6]. Služba IoT Core nabízí správu zařízení, včetně autorizace, vydávání certifikátů pro přístup k Google Cloud a odesílání konfigurace do jednotlivých zařízení.

V oblasti získávání dat nabízí Google Cloud zabezpečený MQTT server, který je automaticky přímo propojen se službou Pub/Sub [7]. Data ze služby Google Cloud Pub/Sub lze po sléze vizualizovat službou Google Cloud Monitoring [8]. Jednoduché ale mocné uživatelské rozhraní umožňuje vytvořit nejrůznější grafy a přístrojové desky s informacemi z různých zařízení.

Služba Google Cloud Dataflow [9] pak umožňuje provádět transformace nad nasbíranými daty. K dispozici je například výpočetní prostředí Apache Beam. V oblasti řízení takového systému v reálném čase ale Google Cloud žádnou službu nenabízí. Je možná realizace pomocí lokálně hostované služby, která má přístup k Pub/Sub, poslouchá data která IoT zařízení generují a publikuje příkazy, kterými IoT zařízení řídí. To je ale daleko za hranicí dostupnosti pro běžné uživatele, takže Google Cloud pro naše účely nevyhovuje.

1.3 Diskuse o požadavcích

Následuje diskuse o různých požadavcích, které chceme aby aplikace splňovala. V nich se zaměříme zejména na eliminaci problému popsaného v části 1.1, ale taky budeme chtít některé běžné funkce, které uživatel od tohoto typu aplikace může očekávat.

V dalším bodě pak požadavky shrneme a očíslovíme.

1.3.1 Požadavky na komunikaci

Pochopitelně chceme aby aplikace byla kompatibilní s co největším množstvím IoT zařízení. V ideálním případě chceme, aby na tom nezáleželo, nějakou technologii ale musíme zvolit. V návrhu tedy pouze požadujeme, aby to byla oddělená vrstva, a případně se jiná technologie dala snadno doprogramovat.

1.3.2 Přejížděcí funkce jako výraz

Pokračujme v příkladu z podkapitoly 1.1. Poučený uživatel odstraní pravidla (c, d) a upraví pravidla (a, b) takto:

(a*) „Pokud teplota klesne pod 40 °C a zároveň je levná sazba, zapni ohřev.“

(b*) „Pokud teplota stoupne nad 60 °C, nebo je drahá sazba, vypni ohřev.“

Toto nastavení je sice o mnoho lepší, ale zase může nastat situace, kdy si uživatel bude muset počkat na levnou sazbu aby se mu ohřála voda. V tuto chvíli by bylo vhodné mít možnost formulovat pravidla obecněji jako nějaký výraz a tím docílit kompromisu, například:

(a**) „Zapni ohřev, pokud teplota vody klesne pod 40 °C.“

(b**) „Vypne ohřev, pokud teplota vody stoupne nad:“

- 60 °C (pokud je levná sazba)
- 50 °C (jinak)

Hraniční teplota v pravidle (b**) je funkcí indikátoru HDO levné sazby s konečným množstvím hodnot, takže ji lze vyjádřit jako konečně mnoho pravidel (konkrétně dvě), každé v konjunkci s jiným stavem HDO. Kdybychom chtěli, aby hraniční teplota byla spojitou funkcí nějakého jiného jevu, už by to nešlo.

Proto chceme aby naše aplikace umožnila vyjádřit přechodové funkce explicitně jako výrazy.

1.4 Požadavky

Z předchozí diskuse zformulujeme konkrétní uchopitelné požadavky.

1.4.1 Funkční požadavky

- F1** Aplikace zaznamenává historii naměřených hodnot do databáze. Ty si pak uživatel může prohlížet (běžná funkce, kterou uživatel může očekávat).
- F2** Aplikace deleguje zprávy mezi zařízeními na základě uživatelem popsaných přechodových funkcí (viz 1.3.2).
- F3** Uživatel si může na úvodní obrazovce poskládat přehled různých ukazatelů měření, včetně grafů zobrazující historii naměřených hodnot (běžná funkce, inspirace z eWeLink).
- F4** V grafech se nově naměřené hodnoty objevují v reálném čase (tento požadavek byl vznesen jako návrh od jednoho uživatele).

Požadavek F2 je právě něco, co odlišuje tento systém od ostatních, a předmětem této práce dále je ověřit, jestli jeho zavedením zlepšíme použitelnost.

1.4.2 Nefunkční požadavky

- N1** Historii naměřených hodnot lze prohlížet plynule v libovolném měřítku, tj. latence uživatelského rozhraní je nejvýše 150 ms, nezávisle na měřítku grafu (jeden z nedostatků Node-RED).
- N2** Architektura aplikace umožňuje rozšíření o dodatečné databázové a IoT technologie (diskuse o komunikaci 1.3.1).

2. Návrh aplikace

Tato kapitola dokumentuje návrh architektury aplikace, použité technologie a důvody k jejich výběru.

Na úplný přístup metodou *domain driven design* (DDD) máme příliš malý problém na to, abychom ho rozdělovali na domény. Kvůli požadavku na extenzibilitu (N2) ale chceme, aby byla architektura dostatečně flexibilní. Architekturu aplikace proto sestavme jako tradiční třívrstvý model webové aplikace s prvky DDD.

Jednotlivé vrstvy nechť spolu komunikují přes pevně stanovená rozhraní, ale můžeme na ně také nahlížet jako na domény, které uvnitř pracují s doménově specifickými objekty, a ven publikují pouze data odpovídající specifikaci daného rozhraní.

Následující podkapitoly se věnují nejprve konceptuálnímu modelu a pak jednotlivým vrstvám.

2.1 Konceptuální model

Zde si popíšme entity, které budou v systému figurovat.

Zařízení ¹

Pro aplikaci představuje přístupový bod do reálného světa (vstupní i výstupní). Vyměňuje si informace se systémem a na základě této komunikace potenciálně upravuje svoje chování. Vážou se s ním informace, jak takové zařízení kontaktovat a taky nastavení, zda ukládat naměřené hodnoty, nebo je zahazovat.

Záznam o měření

Představuje záznam o proběhlém naměření nějaké veličiny na nějakém zařízení. Záznam obsahuje naměřenou hodnotu a časovou známku (datum a čas) měření.

2.2 Volba technologií ovlivňujících návrh

V této podkapitole si představme rozhodnutí, která by se dala řadit do implementační stránky aplikace, ale měla zásadní vliv na návrh architektury a uživatelského rozhraní.

2.2.1 MQTT

Jako komunikační protokol který poslouží jako rozhraní mezi naší aplikací a Internet of Things (IoT) zařízeními zvolme MQTT protokol. Jeho centralizovaná a kanálově orientovaná architektura přináší maximální eliminaci závislostí mezi účastníky komunikace, tj. mezi naší aplikací a IoT zařízeními.

¹K pojmenování: z pohledu programu jde o logické *rozhraní* mezi programem a světem IoT, proto se v kódu často vyskytuje pod pojmem `interface`. Ale pro uživatele se zpravidla jedná o *zařízení*. Jde o tu stejnou entitu, ale v textu práce jsem rozhodl označovat ji takto.

MQTT je navíc velmi populární v oblasti embedded systému a různých mikrokontrolerů, protože provoz jeho klienta není náročný na výpočetní zdroje, jak uvádí specifikace MQTT [10]. Citace:

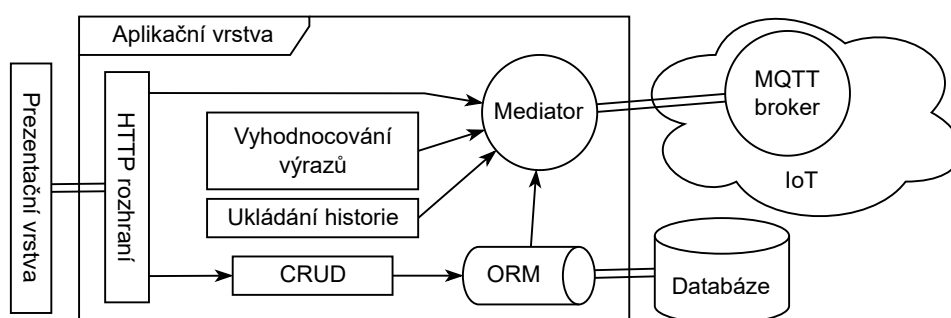
MQTT je protokol přenosu zpráv přes prostředníka pomocí publikování a odběru. Je lehký, otevřený, jednoduchý a navržený tak, aby se dal snadno implementovat. Díky těmto vlastnostem je ideální pro použití v mnoha situacích, včetně omezených prostředí, jako je například komunikace Machine to machine (M2M) a IoT, kde je vyžadována malá kódová stopa nebo je omezená šířka pásma sítě.

2.3 Aplikační vrstva

Tato vrstva obsahuje část aplikační logiky která se odehrává centralizovaně pro celou aplikaci. Aplikační vrstva má několik úkolů, které si postupně představíme v následujících sekcích.

2.3.1 Architektura

Aby byla tato vrstva modulární a umožnila tak snadné rozšiřování a tím splnila požadavek N2, přistupujme k realizaci níže uvedených úkolů pomocí služeb. Služby jsou uzavřené funkční celky, které dávají navenek k dispozici nějaké rozhraní. Navíc zavedme další centralizovaný model komunikace, tzv. *mediator* design pattern. Ten umožní eliminaci některých závislostí mezi službami, protože služby komunikující přes něj prostřednictvím zpráv nemusí znát navzájem svoje rozhraní. To zjednoduší závislosti mezi jednotlivými komponentami při implementaci. Navíc můžeme využít mediator k transparentnímu zahrnutí MQTT zpráv do aplikace.



Obrázek 2.1: Komponentový diagram aplikační vrstvy. Šipky značí závislost na rozhraní jiných komponent (směřují od závislého k závislosti). Dvojitá čára značí komunikaci za hranice aplikace.

2.3.2 Vyhodnocování výrazů

U zařízení rozlišujeme dva druhy: vstupní a výstupní. Každé výstupní zařízení je opatřeno výrazem, který určuje přechodovou funkci stavu tohoto zařízení. Přechodová funkce je funkcí stavu všech ostatních zařízení v systému a její hodnota udává stav výstupu tohoto zařízení v dalším časovém okamžiku.

Tedy když se změní stav nějakého zařízení, systém musí přepočítat všechny výrazy které se na tento stav odkazují a výsledky publikovat na příslušná výstupní zařízení. To může dále způsobit kaskádu dalších změn.

2.3.3 Ukládání historie

Pro všechna zařízení, u kterých je povoleno ukládání naměřených hodnot, chceme ukládat naměřené hodnoty do databáze. Když se změní naměřená hodnota nějakého zařízení, ať už vstupního nebo výstupního, chceme tuto hodnotu uložit.

2.3.4 HTTP rozhraní

HTTP rozhraní slouží k interakci prezentační vrstvy se systémem. Při jeho návrhu se kromě části pro autorizaci držme principů REST [11].

Chceme aby v aplikaci bylo alespoň základní zabezpečení, tj. přihlašování pomocí uživatelského jména a hesla. To, jak si uživatel tyto údaje zvolí, ponechme na implementaci.

Pro splnění požadavku N1 zvolme sofistikovanější způsob prezentace naměřených hodnot v nějakém časovém úseku. Dotaz bude kromě začátku a konce časového intervalu obsahovat také informaci o hustotě bodů, kterou klient potřebuje. Na pozadí můžeme udělat agregační dotaz do databáze a ušetřit tak množství stažených dat z databáze, poslaných dat na klienta, a konečně i počet bodů nakreslených do grafu.

Taky chceme aby tato vrstva přes HTTP rozhraní prezentovala nové naměřené hodnoty v reálném čase (požadavek F4). Architektonicky to vyřešíme tak, že umožníme HTTP rozhraní odebírat zprávy z mediátoru (obrázek 2.1) a posílat je na klienty.

2.4 Databáze

Množství dat, se kterým bude aplikace pracovat může být potenciálně vysoké. Chceme se například podívat na vývoj teploty za posledních 24 hodin, teploměr ale posílá naměřenou hodnotu každých 5 sekund, to je 17 280 záznamů. Nad nimi budeme nejspíš chtít provést nějaký agregační dotaz, kvůli požadavku N1. Zvolme tedy nějakou efektivní relační databázi.

2.5 Prezentační vrstva

Jako technologii pro prezentační vrstvu zvolme webovou aplikaci. Je dostupná na všech platformách, na kterých je dostupný běžný webový prohlížeč.

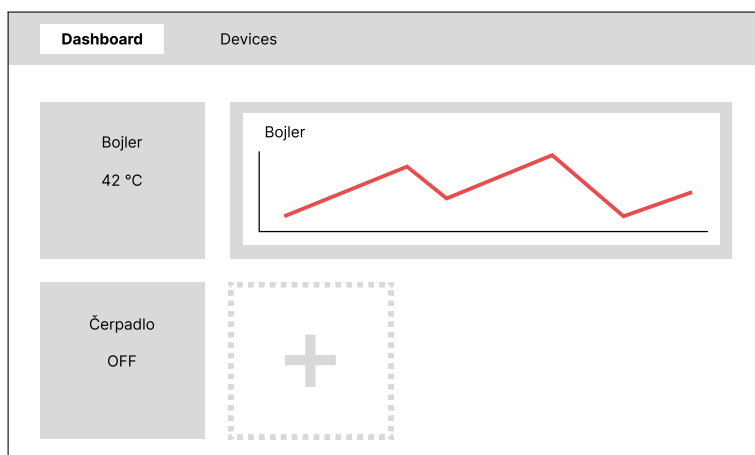
Fakt, že lidé používají různé webové prohlížeče sebou nese určitou problematiku spojenou se závislosti na konkrétním prohlížeči. Některé tyto problémy lze ale při implementaci vyřešit vhodnou volbou knihoven, které zobecněním funkcionalit smažou rozdíly mezi prohlížeči.

2.5.1 Uživatelské rozhraní

Cílem je, aby aplikace uživateli poskytla jednoduchou správu výrazů popisujících vztahy mezi MQTT zařízeními a přehledné zobrazení a procházení naměřených hodnot.

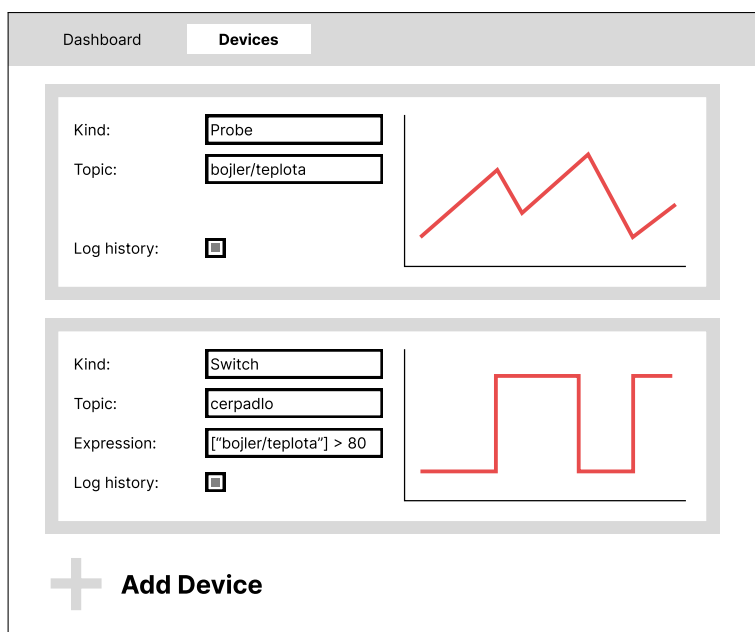
Stránka s přístrojovou deskou

Na této stránce si uživatel může vytvořit rychlý přehled naměřených hodnot. Může přidat dlaždice, které ukazují buď graf nebo poslední naměřenou hodnotu přijatou z nějakého zařízení. Způsob, jakým se tyto dlaždice vytváří a spravují ponechme na implementaci.



Obrázek 2.2: Návrh obrazovky s přístrojovou deskou

Stránka s rozhraními



Obrázek 2.3: Návrh stránky s rozhraními

Na této stránce může uživatel zaregistrovat nová MQTT zařízení. Může si zvolit, jestli se jedná o vstupní nebo výstupní zařízení, jestli má aplikace ukládat naměřené hodnoty do databáze, a u výstupního zařízení může výrazem zapsat přechodovou funkci jeho stavu.

Přihlašovací stránka

Pokud uživatel není přihlášen, zobrazí se mu stránka s jednoduchým přihlašovacím formulářem, který má pole pro přihlašovací jméno a heslo a tlačítko pro přihlášení.

3. Implementace

V této kapitole si popíšeme rozhodnutí provedené během implementace aplikace na základě návrhu.

Tato kapitola si neklade za nárok do hloubky zdokumentovat detaily, pouze vybraná rozhodnutí a technologie.

3.1 Aplikační vrstva

Jako základní technologii zvolme ASP.NET Core 5. Tato volba je subjektivní. Jde o zmodernizované provedení populární technologie ASP.NET pro platformu .NET 5. V ASP.NET Core je zvykem držet se architektury Model-View-Controller (MVC) [12], takže se jí držíme i my. Tato volba později umožní uplatnit tzv. *code-first* přístup, a to při popisu databáze i HTTP API.

Code-first znamená, že formální jazyk popisu věci je přímo programovací jazyk dané platformy, v našem případě C#. Výhodou je, že se pak do projektu nezanáší další doménově specifické jazyky a formáty, jako je SQL k popisu databázového schéma, nebo OpenAPI strukturovaný popis HTTP rozhraní.

Pro automatizaci mapování mezi objekty jazyka C# a dotazy konkrétní databáze použijeme object relational mapping (ORM) framework, konkrétně Entity Framework Core. Ten podporuje širokou řadu databázových systémů a nabízí dobře zdokumentovaný způsob, jak si takovou podporu případně naimplementovat [13, Database Providers]. To výrazně usnadní potenciální rozšíření aplikace o podporu dalších databázových systémů (požadavek N2).

3.1.1 Vyhodnocování výrazů

Podle požadavků chceme realizovat regulaci prostřednictvím přechodových funkcí zapsaných výrazy. Na to bude třeba navrhnout doménově specifický jazyk těchto výrazů, tento jazyk posléze parsovat a napsané výrazy pak vyhodnocovat.

Gramatika

Pro zápis přechodových funkcí zkonstruujeme jazyk výrazů popsany následující gramatikou. Zápis gramatiky volně odpovídá formátu GNU Bison:

```
expr: expr binary_op expr
    | unary_op expr
    | FUNC_NAME '(' ')'
    | FUNC_NAME '(' param_list ')'
    | '(' expr ')'
    | topic_literal
    | NUMBER_LITERAL
    ;

param_list: expr
           | param_list ',' expr
```

```

        ;
unary_op: '-' ;

binary_op: '*' | '/' | 'mod'
          | '+' | '-'
          | '=' | '>' | '<' | '>=' | '<='
          | 'and'
          | 'or' ;

topic_literal: '[' TOPIC_NAME ']' ;

```

Termy zapsané hůlkovým písmem představují následující terminály:

- FUNC_NAME je jméno zabudované funkce.
- NUMBER_LITERAL je číselná konstanta s desetinnou tečkou.
- TOPIC_NAME je název MQTT kanálu.

Funkce zabudované v jazyce

V jazyce potřebujeme nějaké funkce. Zvolme nějaké analytické, krokové a nějaké pro práci s časem. Jejich kompletní seznam je v příloze A.4.2 Přehled všech operátorů a funkcí.

Parsování

K parsování použijme knihovnu pro kombinování parserů Pidgin [14]. V ní je možné vytvářet parsery PEG, které jsou slabší než CFG, ale na výrazy budou stačit.

3.1.2 HTTP rozhraní

Při vývoji HTTP rozhraní volíme přístup *code-first*. V ASP.NET Core popíšeme jednotlivé trasy jako metody kontrolerů MVC. Pomocí nástroje Swagger [15] poté můžeme vygenerovat OpenAPI specifikaci

Autorizace a autentifikace

Pro přístup k některým metodám rozhraní budeme vyžadovat přihlášení. Protože je aplikace míněna primárně pro lokální hostování, vystačíme si s jedním uživatelským účtem jehož údaje budou napevno nastaveny v konfiguračním souboru.

Pro autentifikace použijme JWT [16] s možností prodloužení platnosti tokenu. Výhodou JWT je, že nemusíme na straně serveru udržovat informace o otevřených sezeních s klienty. Aby si klientská aplikace nemusela ukládat uživatelské údaje, přidejme navíc možnost prodloužení sezení pomocí tzv. *refresh tokenu*.

Životní cyklus klientských sezení bude probíhat takto:

1. Klient se přihlásí tím, že pošle přihlašovací údaje na API a dostane zpět dvojici tokenů, JWT a *refresh token*.
2. Klient nyní může posílat autorizované hyper text transfer protocol (HTTP) dotazy tak, že je označí hlavičkou `Authorization: Bearer <JWT token>`.
3. Po vypršení platnosti JWT může klient udělat jedno z následujících:
 - (a) Požádat o nový JWT pomocí *refresh tokenu*, který takto může použít jen jednou.
 - (b) *Refresh token* zahodit, čímz se prakticky odhlásí.

Microsoft SignalR

Z oficiální dokumentace [17]:

SignalR je knihovna pro ASP.NET která výrazně usnadňuje proces implementace real-time funkcionality do webových aplikací. Real-time funkcionality znamená umožnit serveru posílat informace připojeným klientům okamžitě ve chvíli kdy vzniknou a nemuset spoléhat na to, že si klienti budou o nová data muset žádat.

Na pozadí zvolí SignalR vhodnou technologii pro přenos. Jmenovitě jednu z *WebSockets*, *Server Sent Events*, *Forever frame* nebo *Long polling*.

Tato funkcionality se nám bude hodit na splnění požadavku F4. Popis takového druhu interakce nenáleží OpenAPI, ale přesto ho zahrňme do následujícího seznamu přístupových bodů.

Routování

Zde si představíme přístupové trasy HTTP. Metody označené * vyžadují autorizovaný přístup. Přesný seznam včetně schématu je uveden v příloze C Vývojová dokumentace.

POST /api/v1/identity/login

slouží k získání přihlašovacího tokenu. V těle dotazu přijímá přihlašovací jméno a heslo. Při úspěšném přihlášení vrací dvojici tokenů v těle odpovědi.

POST /api/v1/identity/refresh

slouží k prodloužení platnosti přihlašovací tokenu. V těle dotazu přijímá starý token a vrací nový v těle odpovědi.

GET* /api/v1/identity

vrací informace o probíhajícím sezení.

GET* /api/v1/interface

vrací seznam všech MQTT zařízení.

POST* /api/v1/interface

v těle dotazu přijímá částečný popis MQTT zařízení. Pokud je dotaz v pořádku, zaregistruje nové rozhraní a vrací HTTP status 201 s prázdným tělem odpovědi.

GET* /api/v1/interface/{ifaceId}

přijímá jeden `ifaceId` a vrátí informace o MQTT rozhraní s tímto id.

PATCH* /api/v1/interface/{ifaceId}

přijímá jeden parametr `ifaceId` a v těle dotazu přijímá částečný popis MQTT rozhraní. V databázi upraví záznam o rozhraní s příslušným id.

DELETE* /api/v1/interface/{ifaceId}

přijímá jeden parametr `ifaceId`. V databázi smaže záznam o rozhraní s tímto id.

GET* /api/v1/interface/{ifaceId}/history

přijímá parametr `ifaceId`, v těle dotazu časový interval od - do a počet bodů. Vrací seznam naměřených hodnot z daného intervalu agregovaný tak, aby v něm byl kýžený počet bodů. Každý bod pak představuje malý časový úsek. Obsahuje tedy záznamy o průměru, minimální a maximální hodnotě v tomto úseku.

SignalR Hub /api/v1/eventstream

posílá připojeným klientům události odehrávající se v reálném čase. Tedy mění se hodnoty na MQTT kanálech.

3.2 Databáze

Jako relační databázi zvolme MySQL. Volba je subjektivní, ale zároveň jde o populární technologii [18], tím pádem je technologie aplikace o něco přístupnější programátorské veřejnosti.

3.2.1 Schéma

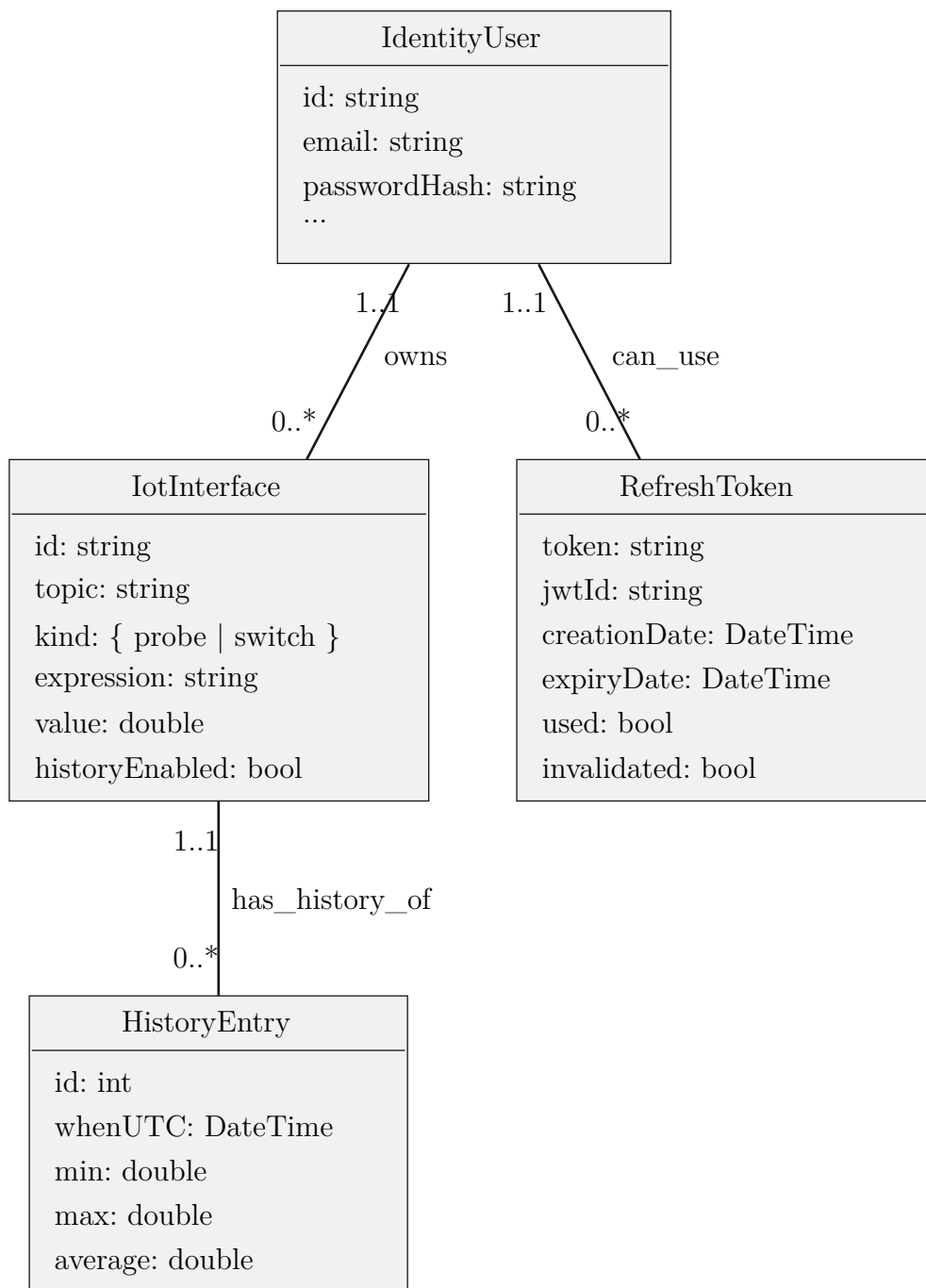
Na obrázku 3.1 můžeme vidět UML diagram databázového schéma. Význam jednotlivých tabulek je vysvětlen níže.

IotInterface obsahuje záznamy o MQTT rozhráních která systém sleduje. Rozhraní jsou dva druhy. *Probe* slouží pouze jako vstup, u kterého můžeme ukládat historii jeho hodnot. *Switch* slouží jako výstup u kterého také může ukládat historii jeho hodnot, ale navíc jeho hodnota je funkcí hodnot ostatních rozhraní. Tuto funkci popíšeme výrazem v poli *expression*.

HistoryEntry obsahuje záznamy naměřených hodnot pro každé zařízení z *IotInterface*.

RefreshToken obsahuje seznam aktuálně platných refresh tokenů pro uživatelská sezení.

IdentityUser obsahuje uživatelské účty. Momentálně je v systému pouze jediný uživatelský účet, ale tabulka je zde abychom mohli použít běžný autentifikátor v ASP.NET Core a usnadnili si případné rozvíjení pro podporu více uživatelských účtů. Tabulka obsahuje mnoho údajů, které nejsou uvedeny, protože je v aplikaci nevyužíváme.



Obrázek 3.1: UML diagram databázového schéma.

3.3 Frontend

Pro usnadnění vývoje prezentační vrstvy zvolme vývojovou platformu Angular. Volba je opět subjektivní, nepřináší výrazné zpomalení oproti ostatním dostupným platformám [19].

Podle oficiální dokumentace [20] je Angular vývojová platforma založená na programovacím jazyce TypeScript. Umožňuje vytvářet komponenty, které synchronizují svůj stav s HTML DOM. Tyto komponenty lze zanořovat do sebe a tím vytvářet složité stromy komponent. Spolu s tím je k dispozici sada nástrojů, které usnadňují vývoj, sestavení, testování a nasazení webových aplikací.

Konkrétní architektura vzniklé aplikace je popsána v příloze C Vývojová dokumentace.

3.3.1 Uživatelské rozhraní

Konkrétní provedení uživatelského rozhraní si můžete prohlédnout na obrázcích v příloze A Uživatelská dokumentace.

4. Dokumentace

V této kapitole uvedeme, jaké další dokumenty jsou s touto aplikací spojeny, co obsahují, a kde je získat.

4.1 Uživatelská dokumentace

Obsahuje popis aplikace, minimální vysvětlení toho, jak funguje MQTT, popis ovládacích prvků uživatelského rozhraní a vysvětlení jazyka výrazu pro laiky pomocí příkladů.

Je k dispozici příloze A Uživatelská dokumentace.

4.2 Instalační příručka

Obsahuje seznam požadavků, které je potřeba splnit pro spuštění aplikace, návod na instalaci a spuštění, a podrobnou dokumentaci konfigurace.

Je k dispozici příloze B Instalační příručka.

4.3 Vývojová dokumentace

Sestává z několika dokumentů.

- Dobrým začátkem je přečíst si `README.md` obou repositářů (příloha E). Jsou tam informace, jak projekty sestavit a spojit dohromady v jednu aplikaci.
- Příloha C Vývojová dokumentace obsahuje stručné instrukce k sestavení aplikační i prezentační vrstvy a přehled architektury prezentační vrstvy.
- *Programmer's manual* je detailní vývojová dokumentace aplikační vrstvy dostupná na <https://muph0.github.io/iot-dash-backend/>. Také k dispozici v příloženém archivu ZIP v adresáři `doc-backend`.
- Dokumentace HTTP rozhraní aplikační vrstvy je dostupná na <https://muph0.github.io/iot-dash-backend/rest.html> nebo v příloženém archivu ZIP v souboru `doc-backend/rest.html`.

Také je k dispozici jako OpenAPI specifikace ve formátu YAML. V příloženém archivu ZIP (`iot-dash-app/swagger.yaml`)

4.4 Kód aplikace

Kód aplikace je k dispozici v příloze E Zdrojový kód aplikace.

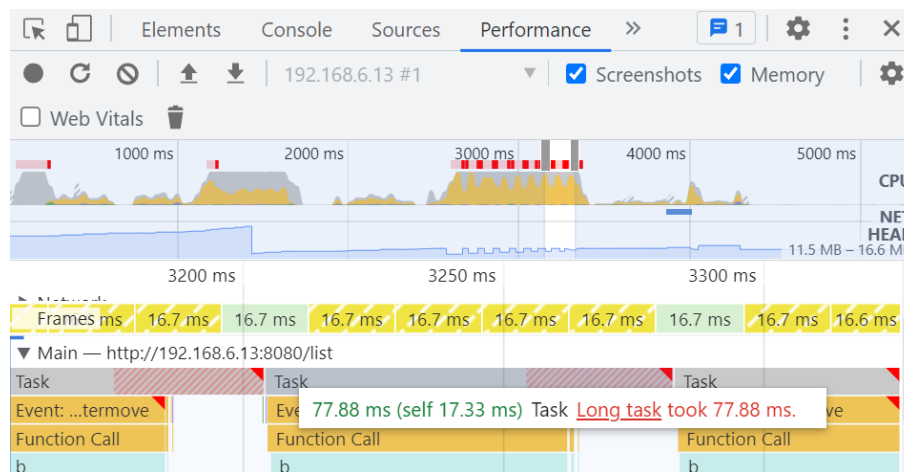
5. Evaluace

V této kapitole zhodnotíme navrženou aplikaci z hlediska požadavků, které jsme vznesli výše (viz 1.3). Popíšeme, jak probíhalo uživatelské testování. Představíme provozní nasazení, ve kterém nyní aplikace běží, a ukážeme příklad složitější regulační úlohy. V poslední řadě uvedeme možná rozšíření do budoucnosti.

5.1 Splnění požadavků

Navržená aplikace splňuje všechny funkční požadavky.

Pomocí DevTools v prohlížeči Google Chrome jsem měřil odezvu uživatelského rozhraní při prohlížení grafu naměřených hodnot za poslední týden (obrázek 5.1). Frekvence měření byla 1 sekunda. Při plynulém přibližování i posouvání v grafu byla doba vykreslení jednoho snímku animace konzistentně menší než 80 ms, takže splňuje i požadavek N1.



Obrázek 5.1: Měření odezvy UI při prohlížení grafu.

Předávání MQTT zpráv uvnitř backendu je realizováno přes mediator pattern. Pokud bychom přidali podporu pro další komunikační technologii implementovanou přes mediator pattern, můžeme pak snadno vytvořit službu, která tyto mediatory propojí.

Použili jsme ORM knihovnu Entity Framework Core, která skryje implementační detaily komunikace s MySQL. Můžeme snadno vyměnit MySQL za jinou technologii, protože na ní není závislá žádná část projektu¹.

Požadavek N2 je tedy splněn také.

5.2 Uživatelské testování

Pro testování byla připravena instalace tohoto systému v jednoduchém prostředí. Figurovala v něm zařízení komunikující po následujících MQTT kanálech:

¹Kromě agregačního dotazu z 3.1.2. U něj jsem musel použít nějaké funkce specifické pro MySQL. Je ale stále zapsaný v jazyce C# a automaticky přeložený do SQL pomocí Entity Framework Core.

bojler/teplota Teplota v bojleru (°C).

bojler/topeni Stav topení (0: vypnuto, 1: zapnuto).

fv/vykon Okamžitý výkon fotovoltaické elektrárny (W).

Testování uživatelé dostali několik úloh spočívajících v nastavení pravidel regulace systému. K ruce dostali uživatelskou dokumentaci (příloha A) a taky seznam MQTT kanálů uvedený výše. Tyto úlohy postupně nabývají v obtížnosti.

Po provedení úloh dostali dotazník měřící tzv. System Usability Scale (SUS) [21]. Je to série deseti otázek, které měří relativní použitelnost systému. Kompletní znění dokumentu, který uživatelé dostali je k dispozici v příloze D.

5.2.1 Vzorové řešení úloh

Tyto výrazy jsou možným řešením úloh z dotazníku. Všechny jsou pro výstupní zařízení bojler/topeni:

1. (bez výrazu)
2. ["bojler/teplota"] < 60
3. ["bojler/teplota"] < if(["bojler/topeni"], 80, 40)
4. ["fv/vykon"] > 1000 and ["bojler/teplota"] < 60
5. ["bojler/teplota"] < if(["bojler/topeni"], if(month() >= 4 and month() <= 10, 60, 80), 40)

5.2.2 Výsledky

K testování se bohužel dostavili jen 3 lidé, kteří se od začátku nezajímali o vývoji aplikace. Jde tedy pouze o předběžné testování na uživateli, kteří přišli s aplikací do styku poprvé.

| <i>Uživatel</i> | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 | <i>Skóre</i> |
|-----------------|----|----|----|----|----|----|----|----|----|-----|--------------|
| A | 4 | 4 | 4 | 3 | 5 | 1 | 2 | 1 | 4 | 3 | 67,5 |
| B | 4 | 1 | 5 | 1 | 5 | 3 | 3 | 1 | 5 | 2 | 85 |
| C | 4 | 4 | 5 | 3 | 5 | 1 | 2 | 2 | 5 | 4 | 67,5 |

Obrázek 5.2: Tabulka odpovědí testovaných uživatelů. Sloupce #1–#10 představují otázky z dotazníku SUS (viz příloha D).

Měli dobré připomínky k aplikaci, jednu z nich jsme zapracovali do aplikace (požadavek F4).

Taky můžeme s jistotou říci, že ačkoliv byli všichni uživatelé s aplikací spokojeni, projít scénář jim trvalo mnohem déle, než jsme předpokládali. Předpokládali jsme 15 min ale všichni nad tím strávili přes 40 min.

Uživatelům A i C přišel systém velmi složitý. Nutno podotknout, že uživatel B, který uvedl „zcela nesouhlasím,“ o sobě prozradil, že má bohaté zkušenosti s MS Excel, a myslí si, že mu to s řešením úloh pomohlo.

5.3 Provozní nasazení

V naší domácnosti jsme postupně vyměnili zařízení SONOFF za různá MQTT zařízení a přešli jsme na používání této aplikace.

Součástí toho byl i vývoj jednoduchých IoT teplotních čidel postavených na čipu ESP8266 s kombinovaným digitálním teploměrem/vlhkoměrem HTU-21D, nebo s digitálním teploměrem DS18B20. Program pro tyto kontrolery byl vyvinut na platformě Arduino. Kód není důležitý, jedná se jen o propojení knihoven pro obsluhu zmíněných teploměrů a pro publikování přes MQTT.

Dále, jeden z výrobků SONOFF je taky založen na čipu ESP8266. Existuje pro něj volně dostupný firmware Tasmota [22], kterým jsme přehráli původní firmware tohoto zařízení. To je dobrá ukázka toho, jak málo stačí, aby zařízení bylo kompatibilní s touto aplikací.

5.3.1 Implementace ekvitermní regulace

Zde si z běžného řešení ekvitermní regulace odvodíme teplotu, na kterou musíme ohřát přívodní otopnou vodu. Střední teplotu teplotnosné látky t_m určíme podle následující rovnice [23]:

$$t_m = t_i + \left(\frac{t_{w1,max} + t_{w2,max}}{2} - t_i \right) \left(\frac{t_e - t_i}{t_{e,min} - t_i} \right)^{\frac{1}{n}}$$

Ochlazení otopné vody Δt určíme takto [23]:

$$\Delta t = (t_{w1,max} - t_{w2,max}) \cdot \frac{t_e - t_i}{t_{e,min} - t_i}$$

kde

- t_i Cílová vnitřní teplota.
- $t_{e,min}$ Minimální venkovní výpočtová teplota.
- $t_{w1,max}$ Maximální teplota přívodu otopné vody.
- $t_{w2,max}$ Maximální teplota zpátečky otopné vody.
- n Teplotní exponent soustavy.
- t_e Naměřená venkovní teplota.

Pak cílová teplota přívodní otopné vody t_{w1} je

$$t_{w1} = t_m + \frac{\Delta t}{2}$$

neboli

$$t_{w1} = t_i + \left(\frac{t_{w1,max} + t_{w2,max}}{2} - t_i \right) \left(\frac{t_e - t_i}{t_{e,min} - t_i} \right)^{\frac{1}{n}} + (t_{w1,max} - t_{w2,max}) \cdot \frac{t_e - t_i}{2(t_{e,min} - t_i)}$$

Toto můžeme v naší aplikaci formulovat například takto:

```
["bojler/teplota"] < ["t_i"] + (  
    (75 + 55) / 2 - ["t_i"]  
) * pow( (["t_e"] - ["t_i"]) / (-12 - ["t_i"]) , 1/1.3)  
+ (75 - 55) * (  
    0.5 * (["t_e"] - ["t_i"]) / (-12 - ["t_i"])  
)
```

kde jsme konstantám stanovili tyto hodnoty

$$\begin{aligned}t_{w1,max} &= 75 & n &= 1.3 \\t_{w2,max} &= 55 & t_{e,min} &= -12\end{aligned}$$

Toto nastavení zajistí, že bude bojler ohřívat otopnou vodu na teplotu t_{w1} . Například v systému eWeLink takové pravidlo vůbec formulovat nelze. V Node-RED zase ano, ale vyžaduje to pracné sestavení tohoto výrazu z bloků, nebo vytvoření jednoduchého programu v jazyce JavaScript. Zatímco v naší aplikaci stačí výraz v podstatě jen opsat.

5.4 Další rozšíření

Grafy historie měření na přístrojové desce zobrazují všechny veličiny ve společném měřítku s jednou osou (obrázek A.1). Bylo by možné os zobrazit více, ale není jasné, jakým způsobem řešit více než dvě.

Další možné rozšíření by bylo přidat podporu pro nějaká IoT zařízení která fungují na jiných technologiích než MQTT. Na to by byla potřeba dále zanalyzovat jaké takové technologie existují a vhodně nějakou vybrat.

Byl vznesen požadavek pro zavedení podpory pro SQLite, protože bychom se pak vyhnuli nutnosti instalovat MySQL. To je plánované rozšíření do budoucnosti.

Další dobré rozšíření by bylo rozšířit jazyk výrazů o podporu pojmenovaných konstant, aby při opravě hodnoty nějaké konstanty nebylo nutné ji měnit na deseti místech.

Závěr

Popsali jsme problém automatizace domácnosti a prozkoumali řešení, která takový problém mohou řešit. Také jsme si představili problémy spojené s událostmi řízenými systémy. Na základě toho jsme vznesli požadavky, podle kterých jsme pak navrhli a implementovali webovou aplikaci.

Aplikace nabízí jednoduché uživatelské rozhraní skrz které si i běžní uživatelé mohou nastavit i složitá regulační pravidla.

Ukázali jsme na příkladu ekvitermní regulace, že lze popsat i taková pravidla, která se v ostatních dostupných řešeních nedají formulovat bez programování nebo skriptování.

I přes malé množství testovacích uživatelů můžeme říct, že aplikace je použitelná. Ale z jejich odpovědí a z časů strávených nad testovacími úkoly je patrné, že formulovat pravidla jako přechodové funkce není snadné.

Seznam použitých zdrojů

- [1] ČEZ. Výkup elektřiny z decentralních zdrojů. Dostupné z: <https://www.cez.cz/firmy/cs/vykup-elektriny-z-decentralnich-zdroju.html>, červenec 2022.
- [2] Václav Matz. Ekvitermní regulace — princip a využití v systémech regulace vytápění. *tbzinfo — Nejnavštěvovanější odborný portál pro stavebnictví a technická zařízení budov*. Dostupné z: <https://vytapeni.tzb-info.cz/mereni-a-regulace/6294-ekvitermni-regulace-princip-a-vyuziti-v-systemech-regulace-vytapeni>, červenec 2022.
- [3] OPENJS FOUNDATION. Node-RED documentation. Dostupné z: <https://nodered.org/docs/>, červenec 2022.
- [4] Node-RED. Node-RED dashboard npm page. Dostupné z: <https://flows.nodered.org/node/node-red-dashboard>, červenec 2022.
- [5] SONOFF. Sonoff products. Dostupné z: <https://sonoff.tech/products/>, červenec 2022.
- [6] GOOGLE. Google Cloud IoT overview. Dostupné z: <https://cloud.google.com/architecture/iot-overview>, červenec 2022.
- [7] GOOGLE. Google Cloud pub/sub. Dostupné z: <https://cloud.google.com/pubsub/docs>, červenec 2022.
- [8] GOOGLE. Google Cloud monitoring. Dostupné z: <https://cloud.google.com/monitoring/dashboards>, červenec 2022.
- [9] GOOGLE. Google Cloud dataflow. Dostupné z: <https://cloud.google.com/dataflow/docs>, červenec 2022.
- [10] Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. MQTT version 5.0. Dostupné z: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html> (cit. červenec 2022), 7. března 2019. OASIS Standard.
- [11] Sandro Hawke. Rdf simple data interface protocol - level zero. Dostupné z: <https://www.w3.org/2001/sw/wiki/REST> (cit. červenec 2022), 2001. (draft).
- [12] MICROSOFT. ASP.NET Core documentation. Dostupné z: <https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0>, červenec 2022.
- [13] MICROSOFT. Entity Framework documentation. Dostupné z: <https://docs.microsoft.com/en-us/ef/core/>, červenec 2022.
- [14] Benjamin Hodgson. Pidgin. Dostupné z: <https://github.com/benjamin-hodgson/Pidgin>, červenec 2022. (software).

- [15] SMARTBEAR SOFTWARE. Swagger. Dostupné z: <https://swagger.io/>, červenec 2022. (software).
- [16] Jeffrey Fritz. Jwt validation and authorization in asp.net core. Dostupné z: <https://devblogs.microsoft.com/dotnet/jwt-validation-and-authorization-in-asp-net-core/>, červenec 2022.
- [17] MICROSOFT. Signalr. Dostupné z: <https://docs.microsoft.com/en-us/aspnet/signalr/>, červenec 2022. (software).
- [18] SOLID IT. Db-engines ranking - trend popularity. Dostupné z: https://db-engines.com/en/ranking_trend, červenec 2022.
- [19] Stefan Krause. js-framework-benchmark. Dostupné z: <https://github.com/krausest/js-framework-benchmark>, červenec 2022.
- [20] GOOGLE. Angular documentation. Dostupné z: <https://angular.io/docs>, červenec 2022.
- [21] UXLS. System usability scale. Dostupné z: <https://uxls.org/methods/system-usability-scale/>, červenec 2022.
- [22] Tasmota. Sonoff basic serial flashing. Dostupné z: <https://tasmota.github.io/docs/devices/Sonoff-Basic/>, červenec 2022.
- [23] Zdeněk Reinberk. Výpočet a graf ekvitermní křivky. Dostupné z: <https://vytapeni.tzb-info.cz/tabulky-a-vypocty/50-vypocet-a-graf-ekvitermni-krivky>, červenec 2022.
- [24] MICROSOFT. .NET 6 runtime. Dostupné z: <https://dotnet.microsoft.com/en-us/download/dotnet/6.0>, červenec 2022. (software).
- [25] ORACLE. MySQL, the world's most popular open source database. Dostupné z: <https://dev.mysql.com/downloads/installer/>, červenec 2022. (software).
- [26] ECLIPSE FOUNDATION. Eclipse Mosquitto™, an open source MQTT broker. Dostupné z: <https://mosquitto.org/>, červenec 2022. (software).
- [27] MICROSOFT. Kestrel web server implementation in ASP.NET Core. Dostupné z: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-6.0>, červenec 2022.

Seznam obrázků

| | | |
|-----|---|----|
| 1.1 | Jednoduchý program pro řízení čerpadla poslouchajícího na MQTT kanálu <code>cerpadlo</code> na základě naměřené teploty bojleru vybaveného teploměrem publikujícím data na kanálu <code>bojler/teplota</code> | 7 |
| 1.2 | Přehled vygenerovaný balíčkem <code>node-red-dashboard</code> k <i>flow</i> z obrázku 1.1. | 7 |
| 1.3 | Aplikace eWeLink, obrazovka s nastavením pravidel regulace. | 8 |
| 1.4 | Aplikace eWeLink, obrazovka s přístrojovou deskou. | 9 |
| 2.1 | Komponentový diagram aplikační vrstvy. Šipky značí závislost na rozhraní jiných komponent (směřují od závislého k závislosti). Dvojitá čára značí komunikaci za hranice aplikace. | 13 |
| 2.2 | Návrh obrazovky s přístrojovou deskou | 15 |
| 2.3 | Návrh stránky s rozhraními | 15 |
| 3.1 | UML diagram databázového schéma. | 21 |
| 5.1 | Měření odezvy UI při prohlížení grafu. | 24 |
| 5.2 | Tabulka odpovědí testovaných uživatelů. Sloupce #1–#10 představují otázky z dotazníku SUS (viz příloha D). | 25 |
| A.1 | Stránka s přístrojovou deskou. | 35 |
| A.2 | Nastavení karty. | 36 |
| A.3 | Stránka se seznamem zařízení. | 37 |

Seznam použitých zkratek

DDD domain driven design

ORM object relational mapping

HTTP hyper text transfer protocol

API application programming interface

JWT JSON Web Token

UML Unified Modeling Language

MQTT Message Queuing Telemetry Transport

SUS System Usability Scale

M2M Machine to machine

IoT Internet of Things

MVC Model-View-Controller

HDO Hromadné dálkové ovládání

Přílohy

K této práci náleží kromě tištěných příloh také přílohy elektronické. U elektronických příloh je vždy uvedeno kde jsou dostupné na internetu a kde je najdete v přiloženém archivu ZIP.

Adresářová struktura archivu:

| | |
|-------------------------------|--------------------------------|
| <code>doc-backend</code> | Vývojová dokumentace backendu. |
| <code>iot-dash-app</code> | Zdrojový kód frontendu. |
| <code>iot-dash-backend</code> | Zdrojový kód backendu. |
| <code>iot-dash-v0.2</code> | Rozbalený instalační balíček. |

A. Uživatelská dokumentace

Toto je uživatelská dokumentace k webové aplikaci IoT-Dash.

A.1 O aplikaci

Aplikace slouží k regulaci jevů v inteligentní domácnosti. Umožní vám řídit MQTT zařízení pomocí výrazů podobných těm v programu MS Excel (viz A.3.3).

Také umožňuje vytvořit interaktivní vizualizace naměřených hodnot z vašich zařízení (viz A.3.1).

A.2 Jak funguje MQTT

MQTT je komunikační protokol, ve kterém chodí informace po tzv. *kanálech* (angl. *topic*). Každý účastník komunikace dostává informace jen z těch kanálů, o které se přihlásí. Z hlediska jednoho kanálu rozlišujeme dva druhy účastníků.

- Vydavatel (angl. *publisher*) na kanál posílá nové informace.
- Odběratel (angl. *subscriber*) z kanálu informace přijímá.

Komunikace v MQTT probíhá přes prostředníka (angl. *broker*). Ten se nepočítá jako účastník, pouze deleguje zprávy mezi účastníky. Pokaždé, když nějaký vydavatel posílá zprávu, sdělí prostředníkovi obsah zprávy a název kanálu, po kterém chce zprávu poslat. Prostředník pak zprávu pošle pouze těm účastníkům, kteří jsou v tu chvíli zaregistrovaní jako odběratelé tohoto kanálu.

A.3 Popis uživatelského rozhraní

Na horním okraji stránky je navigační lišta, na které jsou tyto navigační tlačítka. (viz obrázek A.1)

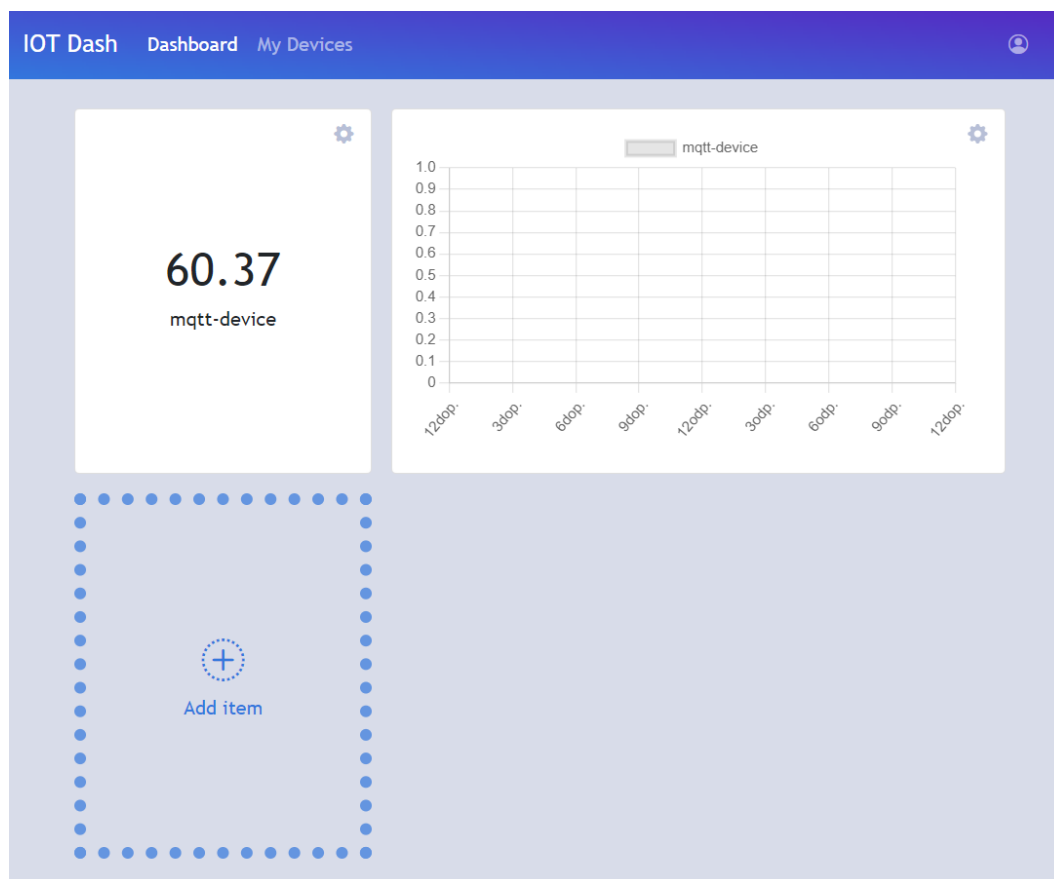
Dashboard — klepnutím zobrazíte stránku s přístrojovou deskou.

My Devices — klepnutím zobrazíte stránku se seznamem MQTT zařízení.

Ikona osoby — klepnutím zobrazíte informace o uživateli a tlačítko odhlášení.

A.3.1 Přístrojová deska

Na přístrojové desce jsou karty, které zobrazují informace o zařízeních (viz obrázek A.1). Tyto karty si můžete uzpůsobit dle vašich potřeb. Pomocí tlačítka „+ Add item“ v tečkovaném rámečku můžete přidat novou kartu. Odstranit karty můžete v jejich nastavení.



Obrázek A.1: Stránka s přístrojovou deskou.

A.3.2 Nastavení karty

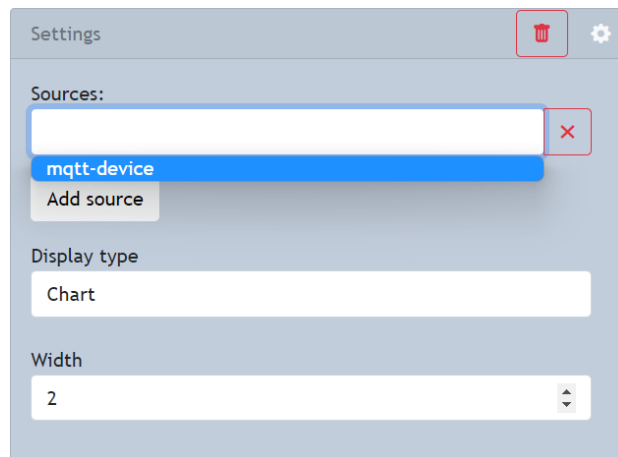
Klepnutím na ikonu ozubeného kolečka v pravém horním rohu každé karty ji přepnete do režimu nastavení (viz obrázek A.2). Každá karta může zobrazit informace z několika zařízení najednou. V kontextu karty jim říkáme *zdroje*. Klepnutím na tlačítko „Add source“ můžete přidat zdroj. Objeví se bílý rámeček s červeným křížkem na pravé straně. Klepnutím na tento rámeček můžete zvolit, které zařízení se stane tímto zdrojem karty. Klepnutím na červený křížek zdroj smažete. Klepnutím na červenou ikonku popelnice v horní liště smažete celou kartu.

Karta může zobrazit informace ze zdrojů dvěma způsoby. K tomu se váže ovládací prvek „Display type“. Buď zobrazí pouze poslední naměřenou hodnotu (volba „Gauge“, obr. A.1 vlevo) nebo zobrazí graf naměřené hodnoty v závislosti na čase (volba „Chart“, obr. A.1 vpravo).

Polem „Width“ můžete nastavit šířku karty v násobcích nejmenší velikosti karty. Podle velikosti vaší obrazovky se do jednoho řádku přístrojové desky vejde různé množství karet.

A.3.3 Seznam zařízení

Na této stránce můžete přidávat, odebírat a spravovat záznamy o zařízeních ve vašem systému (viz obrázek A.3). Každý záznam odpovídá jednomu účastníkovi MQTT komunikace (viz A.2). Klepnutím na zelené tlačítko „+ Add device“



Obrázek A.2: Nastavení karty.

přidáte nový záznam o zařízení. Klepnutím na červené tlačítko „Delete“ záznam o zařízení nenávratně smažete.

Do pole „MQTT Topic“ uveďte MQTT kanál, na který dané zařízení publikuje (odebírá) data.

Každé zařízení zde představuje jedno čidlo (vstup) nebo relé (výstup). K tomu přísluší volby „MQTT Switch“ a „MQTT Probe“ políčka „Type“.

Přepínač „Log Data“ zapne ukládání naměřených hodnot do databáze. Pokud je vypnutý, budou se vám i tak zobrazovat nové naměřené hodnoty v grafu, ale jen na dobu než zavřete tuto stránku.

U výstupních zařízení do pole „Value expression“ запиšte výraz pro přechodovou funkci. Syntaxe výrazů je popsána níže.

A.4 Syntaxe pravidel

Pravidla určující chování výstupních zařízení popisuje tzv. *přechodová funkce*. To je funkce, jejímž vstupem může být stav až všech zařízení v systému, a jejímž výstupem je hodnota, na kterou se vyhodnotí výraz. Tato hodnota je pak odeslána na MQTT kanál daného zařízení.

Výraz přechodové funkce se zapisuje do pole „Value expression“. Ve výrazech můžete použít čísla a běžné aritmetické operátory, odkazy na jiná zařízení, a předdefinované funkce. Syntaxe jazyka je podobná jako výrazy v programu MS Excel.

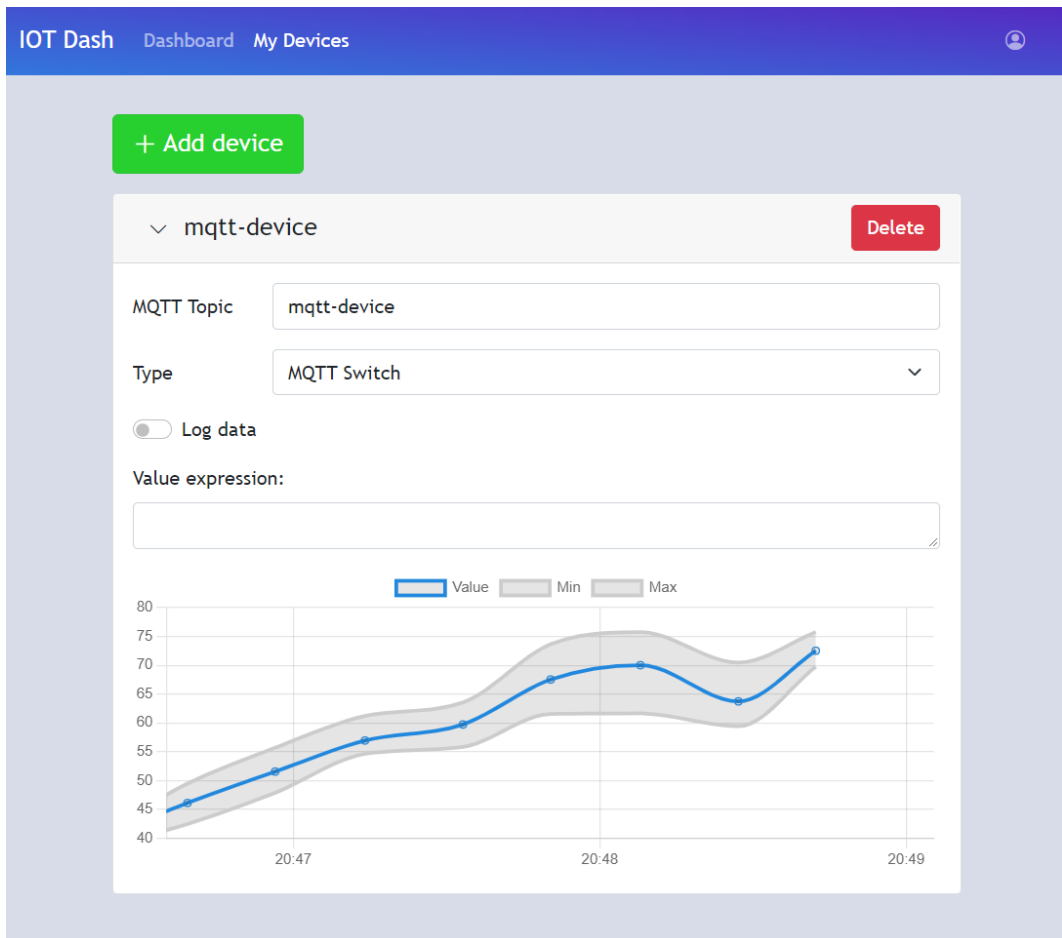
Na hodnotu nějakého kanálu se odkážete napsáním jeho názvu do hranatých závorek, například ["termostat01"].

A.4.1 Příklady

Pro příklad si představme že vyplňujeme „Value expression“ zařízení na kanálu `switch`. Toto zařízení přijímá dvě hodnoty, 1: zapnuto, 0: vypnuto.

1

Výraz se vždy vyhodnotí na 1. Zařízení je vždy zapnuté



Obrázek A.3: Stránka se seznamem zařízení.

```
time_of_day() >= 9 and time_of_day() < 12
```

Výraz se vyhodnotí na 1 pokud je čas mezi 9 a 12. Zařízení je zapnuté od 9:00 do 11:59.

```
["temp01"] < 50
```

Výraz se vyhodnotí na 1 pouze pokud je hodnota kanálu `temp01` menší než 50. Jinak se vyhodnotí na 0. Zařízení je zapnuté pouze pokud je hodnota `temp01` menší než 50.

```
["temp01"] < if(["switch"], 80, 60)
```

`switch` je název kanálu, ve kterém je tento výraz zapsaný. Když teplota klesne pod 60, výraz se vyhodnotí jako 1. Tím se ale změní hraniční hodnota z 60 na 80, takže se vyhodnotí jako 0 až při stoupnutí nad 80. Toto nastavení odpovídá termostatu s hysterezí.

A.4.2 Přehled všech operátorů a funkcí

Operátory jsou uvedeny po řádcích podle přednosti, sestupně:

| | |
|-----------|---|
| - | unární mínus |
| *, /, mod | násobení, dělení, zbytek po dělení |
| +, - | sčítání, odčítání |
| =, >, < | rovno, větší, menší, větší nebo rovno, menší nebo rovno |
| >=, <= | (výsledek výrazu je 1 pokud je podmínka splněna, 0 jinak) |
| and | logická konjunkce („a zároveň“, výsledek výrazu je 1 pouze pokud jsou oba operandy nenulové, 0 jinak) |
| or | logická disjunkce („nebo“, výsledek výrazu je 1 pouze pokud je alespoň jeden operand nenulový, 0 jinak) |

V jazyce jsou zabudované následující funkce:

`if(a, b, c)` je podmíněný výraz. Pokud a není 0, pak je výsledek b . Jinak je výsledek c .

`floor(x)` zaokrouhlí x na nejbližší nižší celé číslo.

`ceil(x)` zaokrouhlí x na nejbližší vyšší celé číslo.

`abs(x)` je absolutní hodnota z x .

`time_of_day()` vrací počet sekund od začátku (půlnoci) dnešního dne.

`day_of_week()` vrací číslo aktuálního dne v týdnu (1: pondělí, 7: neděle).

`day_of_year()` vrací číslo aktuálního dne v roce (1...366).

`month()` vrací číslo aktuálního měsíce (1...12).

`pow(x, y)` spočte mocninu x^y .

`log(x, b)` je logaritmus x při základu b . Výsledek výrazu je $\log_b(x)$.

`log(x)` je přirozený logaritmus x . Výsledek výrazu je $\log_e(x)$.

`exp(x)` je exponenciála. Výsledek výrazu je e^x .

B. Instalační příručka

Tato část popisuje, jak nasadit aplikaci do vašeho systému, jak ji nastavit podle vašich potřeb, a jak ji provozovat.

B.1 Požadavky

Aplikaci můžete spustit na běžných operačních systémech, jako je Linux, iOS nebo Windows. Nebo na jakýchkoliv jiných, které podporuje platforma .NET 6 [24].

Pro spuštění aplikace je nutné nejprve nainstalovat a spustit některé služby třetích stran, jejich seznam je uveden níže.

B.1.1 MySQL

Aktuálně podporujeme jediný databázový backend, a to je MySQL. Pokud už nemáte běžící instanci MySQL dostupnou z počítače na kterém poběží IoT-Dash, nainstaluje ji prosím prostřednictvím vašeho oblíbeného balíčkovacího systému, nebo pomocí oficiálního instalačního balíčku [25].

Po té co MySQL úspěšně nainstalujete, vytvořte novou databázi pro aplikaci, například `iot-dash`. Doporučujeme vytvořit i uživatelský účet, který aplikaci zpřístupní pouze tuto jedinou databázi. Informace o něm a o databázi pak musíte uvést v konfiguraci aplikace (viz B.3).

B.1.2 MQTT broker

Další služba, kterou IoT-Dash potřebuje ke své funkci, je MQTT broker. Pro lokální hostování si můžete nainstalovat např. Eclipse Mosquitto™ [26].

B.1.3 .NET runtime

Abyste aplikaci mohli spustit na nějakém počítači, potřebujete na něm mít nainstalované běhové prostředí .NET 6 [24] (neplést s .NET Core a .NET Framework).

B.2 Instalace IoT-Dash

Sestavenou aplikaci připravenou k nasazení můžete získat z <https://github.com/Muph0/iot-dash-backend/releases> nebo z příloženého archivu ZIP (viz Přílohy). Release archiv obsahuje spouštěcí skripty pro různé platformy. Na Linuxu nebo iOS spusťte `run.sh`, na Windows `run.bat`.

B.3 Konfigurace

Aplikace čte konfiguraci ze souboru `appsettings.json`. Tato část je rozdělena do několika témat týkajících se různých částí konfiguračního souboru. Postupně

jsou uvedeny klíče a hodnoty ve výchozí podobě.

B.3.1 Webové rozhraní

```
"Kestrel": {
  "Endpoints": {
    "Http": {
      "Url": "http://*:8080"
    }
  }
},
```

Hodnota klíče `Url` představuje URL, na které bude server poslouchat.

Kestrel HTTP server umožňuje široké možnosti nastavení hostování, včetně HTTPS. Detailní popis tohoto nastavení je k dispozici s příklady v dokumentaci ASP.NET Core [27, Endpoints].

B.3.2 Připojení k databázi

```
"ConnectionStrings": {
  "DefaultConnection": "server=localhost;port=3306;database=my-db;
                        uid=iot-dash;password=iot-dash"
},
```

`DefaultConnection` určuje způsob připojení k MySQL databázi. Sestává z seznamu klíčů a hodnot oddělených středníkem.

- `server` je hostname nebo IP adresa počítače s MySQL serverem.
- `port` je číslo portu, na kterém MySQL server poslouchá.
- `database` je název databáze, ke které bude aplikace používat.
- `uid` je jméno MySQL uživatele, kterým se aplikace přihlašuje do databáze.
- `password` je heslo MySQL uživatele.

B.3.3 Nastavení JWT

```
"Jwt": {
  "Secret": "secret-string",
  "TokenLifetime": "00:14:30",
  "RefreshTokenLifetime": "180.00:00:00",
  "Algorithm": "HS256"
},
```

Klíč `Jwt` drží nastavení uživatelské relace:

- `Secret` je heslo použité k podepisování tokenů. Toto heslo doporučujeme změnit.

- `TokenLifetime` je doba platnosti JWT tokenu, tedy délka trvání uživatelské relace.
- `RefreshTokenLifetime` je doba po kterou se již přihlášená aplikace může znovu připojit bez nutnosti chtít po uživateli heslo.
- `Algorithm` je algoritmus hešování JWT tokenu.

B.3.4 Nastavení MQTT

```
"Mqtt": {
  "Broker": {
    "Host": "localhost",
    "Port": 1883,
    "MaxReconnectionAttempts": 3
  },
  "Credentials": {
    "UserName": "my-username",
    "Password": "my-password",
  }
},
```

Klíč `Broker` obsahuje nastavení připojení k MQTT brokeru.

- `Host` je hostname nebo IP adresa počítače, na kterém běží MQTT Broker.
- `Port` je port, MQTT Broker poslouchá.
- `MaxReconnectionAttempts` je maximální počet pokusů o připojení k brokeru. Pokud je tento počet překročen, aplikace se ukončí. Pro neomezený počet uveďte 0.

Klíč `Credentials` obsahuje jméno a heslo pro zabezpečené připojení k MQTT. Pokud nepoužíváte zabezpečené MQTT, tento klíč můžete odstranit.

- `UserName` je uživatelské jméno pro přihlášení k brokeru.
- `Password` je heslo.

B.3.5 Nastavení logování

```
"Logging": {
  "LogLevel": {
    "Default": "Warning",
    "IotDash.*": "Information",
    "Microsoft.Hosting.Lifetime": "Information",
    "Microsoft.*": "Warning"
  }
}
```

Tato sekce umožňuje nastavit, které zprávy se za běhu programu zobrazují v okně terminálu. V logování je celkem šest úrovní závažnosti zpráv, vzestupně **Trace**, **Debug**, **Information**, **Warning**, **Error** a **Critical**. Každá zpráva je opatřena úplným názvem třídy ze které pochází (včetně jmenných prostor).

V sekci **LogLevel** jednotlivé klíče představují šablony názvů tříd. Pro každou zprávu se rozhodne jestli se zobrazí následujícím způsobem:

- Pokud je nějaký klíč v sekci **LogLevel** předponou úplného názvu třídy, ze které zpráva pochází, zobrazí se zpráva pouze tehdy, je-li její úroveň závažnosti větší nebo rovna hodnotě uvedené u tohoto klíče.
- Jinak se zpráva zobrazí pouze tehdy, je-li její úroveň závažnosti větší nebo rovna hodnotě uvedené u klíče **Default**.

C. Vývojová dokumentace

Tato příloha je určena pro programátory, kteří chtějí přispívat do kódu aplikace IoT-Dash.

C.1 Aplikační vrstva

Na adrese <https://muph0.github.io/iot-dash-backend/> je volně dostupný *Programmer's manual*, detailní přehled architektury a implementačních detailů jednotlivých komponent aplikace v anglickém jazyce. Také jako webová stránka v adresáři `doc-backend` v příloženém archivu ZIP.

Aby projekt mohl být přístupný programátorské veřejnosti, je nutné aby detailní dokumentace byla dostupná na internetu v anglickém jazyce. Navíc pak může využívat hypertextových odkazů a odkazů do kódu, které by v tištěné podobě ztrácely význam. Proto ta dokumentace není zde.

C.1.1 Sestavení

Pro práci na aplikační vrstvě budete potřebovat mít nainstalované vývojové prostředí .NET 6 SDK. Všechny závislosti získáte provedením příkazu `dotnet restore` v repozitáři.

Příkazem `dotnet build` aplikaci sestavíte. Před sestavením přesuňte sestavenou webovou aplikaci prezentační vrstvy do adresáře `wwwroot`. Její hostování pak probíhá přes HTTP server backendu.

C.1.2 Dokumentace HTTP API

Dokumentace HTTP rozhraní aplikační vrstvy je dostupná v příloženém archivu ZIP v souboru `doc-backend/rest.html`. Ve formátu OpenAPI je pak v souboru `iot-dash-backend/swagger.yaml`.

Když aplikaci sestavíte v konfiguraci `Debug`, přibude na ní navíc trasa `GET /swagger/index.html`. Tam se nachází užitečná, v reálném čase vygenerovaná dokumentace HTTP rozhraní podle aktuálních MVC kontrolérů. Z ní lze interaktivně provádět HTTP požadavky přímo na běžící aplikaci.

Také tam získáte čerstvě vygenerovanou specifikaci ve formátu OpenAPI, provedete-li nějaké změny.

C.2 Prezentační vrstva

Prezentační vrstva je vyvinuta ve frameworku Angular 13. Většinu části kódu tvoří uživatelské rozhraní.

C.2.1 Sestavení

Pro práci na prezentační vrstvě budete potřebovat balíčkovací systém `npm`. Všechny závislosti získáte provedením příkazu `npm install`.

Aplikaci sestavíte příkazem `npx ng build`.

Aplikaci spustíte ve vývojovém režimu příkazem `npx ng serve`.

Ze souboru `swagger.yaml` umístěného v kořenovém adresáři můžete vygenerovat webového klienta pro přístup k backendu příkazem `npm run oapi-gen-v1`.

C.2.2 Architektura

Tato část popisuje hierarchii komponent v uživatelském rozhraní a služby které běží na pozadí aplikace.

Komponenty

V frameworku Angular se uživatelské rozhraní skládá z komponent. Každá komponenta se skládá z kódu (soubor `<název>.component.ts`), šablony, která je někdy součástí souboru s kódem (soubor `<název>.component.html`) a ze stylů (soubor `<název>.component.scss`) Uživatelské rozhraní je tvořeno touto hierarchií komponent.

app : kořenová komponenta aplikace.

Aplikace používá běžné routování dostupné v Angular 13. Další komponenty se zobrazují přes router.

login : Přihlašovací stránka.

user-detail : Stránka s detailem uživatele a s tlačítkem odhlášení.

interface-list : Stránka se seznamem zařízení.

interface-detail : Karta s informacemi o zařízení.

form-error-list : Karta s informacemi o zařízení.

history-chart : Graf naměřených hodnot jednoho zařízení.

dashboard : Stránka s přístrojovou deskou.

app-card : Karta na přístrojové desce.

card-gauge : Ukazatel hodnoty zařízení.

card-chart : Ukazatel grafu naměřených hodnot.

card-src-box : Rámeček pro výběr zdroje dat pro kartu.

Služby

Služby jsou k dispozici v DI kontejneru. Každá služba sídlí v souboru nesoucím její název, tj. `<název>.service.ts`. V kódu těchto třídám zpravidla odpovídají třídy ve tvaru `<název>Service`.

status : Chybové hlášky, které je potřeba zobrazit uživateli jsou poslány na tuto službu. Pokud se přeruší spojení se serverem, tato služba by se to měla dozvědět jako první.

api-v1-provider : Stará se o správnou inicializaci a konfiguraci HTTP klienta vygenerovaného z OpenAPI. Dále tohoto klienta nabízí ostatním službám v DI.

identity : Rozhraní pro službu, která spravuje uživatelskou relaci.

api-v1-identity : Implementace služby **identity**.

persistency : Stará se o ukládání a načítání perzistentních dat.

dashboard : Spravuje model karet na přístrojové desce.

interface : Spravuje model MQTT zařízení v backendu.

mediator : Přijímá a rozesílá události. Jiné služby přes něj mohou odebírat nebo publikovat události.

random-provider : Vytváří náhodné tokeny.

server-event : Spravuje klienta pro SignalR. Přijímá z backendu události o nových naměřených hodnotách.

Výjimkou, co se jmenných konvencí týče, je služba **authentication.mw**. Ta slouží jako middleware do vygenerovaného OpenAPI klienta. Ze služby **identity** si získá uživatelskou relaci, a pokud je uživatel přihlášený, opatří každý požadavek hlavičkou **Bearer**.

D. Dotazník pro testování

Tento dotazník slouží k vyhodnocení použitelnosti reprezentace pravidel jako přechodových funkcí v aplikaci pro automatizaci dějů v domácnosti vytvořenou v rámci bakalářské práce na MFF UK.

Scénáře a vyplnění dotazníku by vám nemělo zabrat více než 30 minut.

D.1 Popis systému

V následujících scénářích budete vytvářet pravidla pro regulaci topení v rodinném domě s elektrickým bojlerem a solární elektrárnou. Níže je popis MQTT kanálů po kterých zařízení komunikují.

`bojler/teplota` Teplota v bojleru (°C).

`bojler/topeni` Stav topení (0: vypnuto, 1: zapnuto).

`fv/vykon` Okamžitý výkon fotovoltaické elektrárny (W).

D.2 Scénáře

Než začnete, přečtěte si prosím uživatelskou dokumentaci A, zejména část A.3.3. Poté si na vašem zařízení otevřete webový prohlížeč, připojte se k webové aplikaci na `http://192.168.6.13:8080/` a přihlaste se. Jméno: `dotaznik`, heslo: `Dotaznik+1`.

Po přihlášení prosím zkuste projít následující scénáře. Pokud se vám jakýkoliv z nich nepodaří splnit, můžete jej přeskočit.

1. V navigační liště klepněte na „My Devices“ a tím přejděte do seznamu zařízení. Přidejte zařízení na MQTT kanálu `bojler/teplota`, zapněte u něj ukládání historie a prohlédněte si přibývajících naměřené hodnoty.
2. Přidejte zařízení na kanálu `bojler/topeni` a nastavte jej tak, aby bylo zapnuté, pokud je teplota vody v bojleru nižší než 60 °C.
3. Nastavte systém tak, aby zapnul topení, pokud teplota vody v bojleru klesne pod 40 °C a aby vypnul topení až když překročí 80 °C.
4. Nastavte systém tak, aby platilo nastavení z 2. úlohy, ale pouze tehdy, když výkon solární elektrárny přesáhne 1 kW. Jinak musí být topení vždy vypnuté.
5. Nastavte systém tak, aby zapnul topení, pokud teplota vody v bojleru klesne pod 40 °C. Teplota, při které se má topení vypnout je 60 °C od dubna do října, 80 °C jinak.

D.3 Dotazník

Vyplntě prosím interaktivní dotazník dostupný na této adrese:

https://docs.google.com/forms/d/e/1FAIpQLSfUIgcYk3XUp-8c4IcJ0hoQX2UfUpL4ui7fFaIx6GZIZSSrYA/viewform?usp=sf_link

Pro tištěnou verzi je zde přepis dotazníku (na každou z otázek může uživatel odpovědět 1–5 prostřednictvím ovládacích prvků):

Tento dotazník slouží k vyhodnocení použitelnosti uživatelského rozhraní aplikace pro automatizaci dějů v domácnosti vytvořené v rámci bakalářské práce na MFF UK.

Vyhodnocení probíhá dle jednoduché metodiky System Usability Scale (SUS): <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>.

Vyplnění dotazníku by vám nemělo zabrat více než 5 minut.

Legenda odpovědí:

1 — Zcela nesouhlasím, 2 — Spíše nesouhlasím, 3 — Nevím, 4 — Spíše souhlasím, 5 — Určitě souhlasím

1. Myslím, že bych chtěl tento systém často používat.
2. Systém mi přišel příliš složitý.
3. Systém mi přišel snadno použitelný.
4. K tomu, abych mohl(a) používat tento systém bych potřeboval(a) podporu technického personálu.
5. Přišlo mi, že různé funkce tohoto systému jsou dobře integrovány.
6. Přišlo mi, že je systém příliš nekonzistentní (rozdílné názvy pro stejné věci, rozdílné ovládání podobných prvků...).
7. Myslím, že většina lidí by se naučila s tímto systémem zacházet velice rychle.
8. Systém mi přišel velice těžkopádný.
9. Při používání systému jsem se cítil(a) že vím co dělám.
10. Před použitím systému jsem se musel(a) naučit hodně věcí.

Děkujeme za vyplnění dotazníku

Konec přepisu.

E. Zdrojový kód aplikace

Tato příloha obsahuje zdrojový kód aplikace v podobě dvou repozitářů Git. Snapshot obou repozitářů je k dispozici ve stejnojmenném adresáři v přiloženém archivu ZIP.

- `iot-dash-backend` je repozitář se zdrojovým kódem aplikační vrstvy. Volně dostupný z <https://github.com/Muph0/iot-dash-backend>.
- `iot-dash-app` je repozitář se zdrojovým kódem prezentační vrstvy. Volně dostupný z <https://github.com/Muph0/iot-dash-app>.