FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	IoT platforma s webovým rozhraním
Student:	Martin Skalický
Vedoucí:	Ing. Jiří Mlejnek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem této práce je navrhnout a implementovat vlastní řešení IoT platformy, která bude primárně určena pro domácí kutily, kteří mohou k platformě připojit libovolné své zařízení. Prostřednictvím webového rozhraní budou moci sledovat naměřené hodnoty nebo zařízení ovládat.

1. Analyzujte existující řešení IoT platforem, srovnejte jejich výhody a nevýhody.
2. Stanovte požadavky na vlastní řešení a porovnejte je s existujícími vybranými platformami.
3. Proveďte rešerši technologií, které lze použít pro implementaci. Na základě této rešerše zvolte pro tento účel vhodné technologie a knihovny.
4. Navrhněte a implementujte vlastní řešení, dle stanovených požadavků.
5. Vytvořte ukázkové chytré zařízení a propojte je s Vámi vytvořenou platformou.
6. Celé řešení nasaďte a otestujte.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 16. listopadu 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

IoT platforma s webovým rozhraním

Martin Skalický

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Mlejnek

13. května 2021

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Skalický, Martin. *IoT platforma s webovým rozhraním.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce se zabývá porovnáním aktuálních IoT platforem na trhu pro domácí použití a následným návrhem a implementací platformy vlastní. Základním pilířem celého řešení je definice schématu, dle kterého každé zařízení popíše své schopnosti a uživateli se následně zobrazí příslušné rozhraní vygenerované na základě jeho dovedností. Platforma je určena pro domácí kutily a technické entuziasty, kteří chtějí mít svá zařízení pod jednotným rozhraním. Uživatelské rozhraní je realizováno progresivní webovou aplikací. Součástí práce je také tvorba zařízení založeného na čipu ESP8266 pro měření teploty v udírně, na kterém je demonstrováno jeho zapojení do platformy.

Klíčová slova IoT platforma, OpenSource, implementace IoT platformy, PWA, MQTT, ESP8266

Abstract

This thesis focuses on a comparison of existing IoT platforms available on the market for smart homes and subsequent design of a custom solution including its implementation. The backbone of whole solution is definition of a scheme that will be used by every connected device to describe its capabilities. User is presented to a generated interface based on the description of connected devices. Platform is designed for use by technical enthusiasts, who want to control all their devices through single interface. The user interface is realized as progressive web app. Part of the thesis also deals with a design and manufacturing of smart device based on chip ESP8266 with purpose of measuring a temperature in a smokehouse. On this smart device the thesis also demonstrates a connection of a device to the platform.

Keywords IoT platform, OpenSource, implementation of IoT platform, PWA, MQTT, ESP8266

Obsah

Seznam obrázků

Úvod

Internet věcí je velmi diskutovaným tématem posledních několika let, ale jeho vývoji předcházela spousta trnitých cest a slepých uliček. Pod kouzelnou zkratkou IoT se pro mnohé skrývá příslib pokroku od chytré domácnosti až po revoluci v průmyslu. Internet věcí je označení pro síť fyzických zařízení, která spolu dokáží komunikovat ať už napřímo nebo pomocí prostředníka. Pro efektivní správu se zařízení připojují k centrální platformě, která sbírá data z jednolivých zařízení a dle definovaných pravidel zařízením posílá příkazy. Například v domácnosti čidlo pohybu zaregistruje příchod majitele domů a platforma v reakci zapne vytápění. Současně poskytuje uživatelské rozhraní, pomocí kterého lze jednotlivá zařízení přímo ovládat a definovat scénáře, na základě kterých vykonává automatizaci jako uvedené automatické zapnutí vytápějí. Platforma je tedy nedílnou součástí světa IoT a od jejích funkcí se odvíjí možnost využití plného potenciálu.

Na trhu již dnes existují hotová řešení, ale je velmi problematické se mezi nimi zorientovat a často bývají velmi drahá. Důvodem vzniku této práce je negativní osobní zkušenost a vysoké poplatky komerčních řešení s cílem vytvoření dostupné otevřené platformy pro technické entuziasty a bastlíře, kteří si chtějí jako já vytvářet levná zařízení a jednoduše je spravovat/ovládat.

Teoretická část práce se věnuje rešerši aktuálních řešení na trhu, jejich vzájemným porovnáním a návrhem vlastního. Praktická část se zaměřuje na implementaci a nasazení vlastního řešení.

Kapitola 1

Cíl práce

Cílem této práce je navrhnout a implementovat řešení IoT platformy, která bude primárně určena pro domácí kutily, kteří mohou k platformě připojit tzv. DIY („Uděl ěj si sám“) zařízení. Platforma tedy bude muset být univerzální tak, aby umožnila připojení různorodých zařízení. Uživatelské rozhraní bude realizováno formou webové aplikace, která umožní sledování a ovládání jednotlivých zařízení.

Hlavním přínosem bude vytvoření alternativy k již existujícím řešením pro ty, kteří hledají vyšší bezpečnost a nižší cenu než často nabízí komerční řešení. Platforma bude koncipována tak, aby si uživatel přidal zařízení na jedno kliknutí a mohl ho ihned ovládat bez nutnosti často zdlouhavých konfigurací, ke kterým zpravidla potřebuje hlubší znalosti. Výsledná platforma bude uvolněna pod OpenSource licencí s možností vlastního hostingu.

Vzhledem k rozsahu Bakalářské práce není cílem nahradit existující řešení, která již obsahují velké množství funkcí, ale vytvořit možnou alternativu a demonstrovat složitost vytvoření celého řešení za použití moderních technologií.

Kapitola 2

Analýza

Tato kapitola se zabývá definicí IoT platformy, analýzou již existujících řešení a následně stanovením požadavků na řešení vlastní.

2.1 Definice IoT platformy

"IoT platforma je více vrstvá technologie, která umožňuje přímočaré zajištění, ovládání a automatizaci připojených zařízení ve světě internetu věci. Zjednodušeně propojuje Váš hardware, jakkoli rozdílný, do cloudu s možností různorodé konektivity, obsahuje bezpečnostní mechanizmy a široké možnosti pro zpracování dat. Pro vývojáře IoT platforma nabízí předpřipravené funkce, které vysoce zvyšují rychlost vývoje aplikací pro připojená zařízení a řeší škálování a kompatibilitu napříč zařízeními". (překlad autora) [5]

2.1.1 Definice pojmů

V této sekci jsou vysvětleny pojmy, které budou použity v následujících kapitolách.

- **Platforma** - Platformou se rozumí programové řešení umožňující propojení různorodých zařízení a jejich následnou obsluhu.
- **Koncové zařízení** - Zařízení, které dokáže komunikovat po síti a nabízí nějakou funkcionalitu (např. měření teploty nebo ovládání světla).
- **Bridge** - Síťové zařízení, které funguje jako prostředník mezi koncovými zařízeními a jinou sítí. Agreguje jednotlivá zařízení a nabízí rozhraní pro komunikaci s nimi.

2.2 Vlastnosti

Tato sekce se věnuje definici klíčových vlastností IoT platformy: způsobu komunikace se zařízeními, možnostmi automatizace, bezpečností a cílovou skupinou.

2.2.1 Komunikace

Komunikaci mezi zařízeními lze realizovat fyzickým propojením či bezdrátově. Drátové propojení je finančně nákladnější a především znamená obrovský zásah do stávající infrastruktury domu, využívá se nejčastěji v nové výstavbě, protože nabízí vyšší spolehlivost a odolnost. Bezdrátová komunikace se naopak používá při instalaci do stávajících domů nebo z důvodu ušetření nákladu a v této práci se jí budeme věnovat blíže. Lze ji rozdělit do dvou základních kategorií:

- **Centralizované** - Každé zařízení komunikuje pouze s jedním centrálním prvkem, přes který jde veškerá komunikace. Mezi nejznámější technologii tohoto typu patří Wifi.
- **Decentralizované** - V této síti komunikují zařízení přímo s ostatními bez jakéholiv prostředníka. Pokud nelze s cílovým zařízením komunikovat přímo, tak využijí ostatní pro předání zprávy. Síť je díky tomu mnohem odolnější vůči výpadkům, protože zde není tzv. „jediný bod selhání“ (single point of failure). Zpravidla mívá nižší datovou propustnost a je složitější pro nasazení a následnou správu. Velkou výhodou je snadnější rozšiřitelnost pokrytí, protože každé přidané zařízení rozšiřuje signál a tímto způsobem lze zařízení řetězit. Pro podrobější popis doporučuji [6].

Vzhledem k rozšířenosti Wifi, kterou dnes najdeme v každé domácnosti, se přirozeně nabízí její využití i pro internet věcí. A k tomu v posledních letech opravdu došlo. Díky extrémně levnému chipu ESP8266, který se dnes i u nás v ČR dá koupit za 70 Kč [7], došlo k masivní penetraci trhu s chytrými zařízeními využívající právě Wifi. Bohužel tato technologie má i svá negativa, největšími jsou spotřeba elektrické energie a limit maximálního počtu připojených zařízení na jeden centrální prvek (řádově desítky). Vysoká spotřeba je dána nutností časté komunikace jen kvůli udržení aktivního spojení a proto je možné provozovat zařízení na baterie pouze v jednotkách dnů, maximálně týdnů.

Pro bateriový provoz vznikly speciální sítě, které na rozdíl od Wifi umožní přenos v desítkach kb za sekundu (tisícina rychlosti běžné Wifi), ale jsou energeticky mnohem úspornější [8] (umožňují provoz až desítky let na malou baterii), mají mnohonásobně větší dosah a umožňují propojení mnohem většího počtu zařízení (stovky).

Poměrně rozšířenými z centrálně orientovaných sítí jsou u nás SigFox a LoRa. SigFox je komerční řešení, kde se platí za každé připojené zařízení [9]. Oproti tomu síť LoRa používá otevřený standard pro komunikaci LoRaWAN [10]. Protože se jedná o otevřený standard, tak kdokoliv může vytvořit a provozovat kompatibilní zařízení [10]. Samozřejmě lze také využít komerční infrastrukturu, kam lze připojit svá zařízení za poplatek, tuto službu nabízí např. České radiokomunikace [11], ale díky otevřenosti má každý možnost si za pár tisíc postavit vlastní GateWay (centrální prvek) a provozovat libovolná zařízení bez jakýchkoliv poplatků a prostředníků.

Z decentralizovaných sítí jsou na trhu poměrně rozšířené Zigbee a Z-Wave. Zigbee je otevřený standard, který dokáže pracovat, jak v pásmu 2.4GHz, tak i 900 MHz [12]. Nemá omezení na maximální počet zařízení zřetězených za sebou a dokáže vytvořit síť skládající se až ze 65 tisíc zařízení [12]. Z-Wave je naopak uzavřený standard, který funguje pouze v pásmu 800-900 MHz [13]. Limituje maximální počet přeposlání zprávy na 4 a podporuje síť o velikosti až 256 zařízení [13]. Obě sítě jsou energeticky velmi úsporné a umožňují běh zařízení na obyčejnou knoflíkovou baterii po dobu až několika let [12, 13].

2.2.2 Automatizace

Automatizace je ve světě IoT pravděpodobně nejdůležitějším tématem a každá IoT platforma by ji měla umožňovat, protože dává možnost využít zařízení úplně novým způsobem [14]. Principiálně se jedná o možnost definování reakcí na jednotlivé události. Událostí může být např. změna teploty, otevření okna nebo detekce pohybu a reakce změna stavu zařízení - zhasnutí světla nebo zapnutí televize [14]. V podstatě jediným limitem je zde lidská představivost. Modelový scénář:

Představme si moderní dům, ve kterém jsou všechny věci, které nás napadnou chytré, což s dnešními technologickými možnostmi není sci-fi, ale naopak možná realita. Majitel domu, říkejme mu Joe, přichází večer unavený domů a odemyká dveře. Vejde dovnitř a světlo na chodbě a v kuchyni již svítí. Jde přímo do kuchyně, protože po dlouhém dni v práci má hlad a usedá s jídlem ke stolu. Nemá rád ticho, tak řekne: „Alexo, zapni hudbu“ a ze sterea se spustí Beethoven, protože Alexa ví, že je to Joeův oblíbený skladatel klasické hudby. Joe cítí, jak se po místnosti rozprostřívá příjemné teplo ze zapnuté klimatizace. Po večeři odchází do druhého patra do koupelny. Samozřejmně se nemusí starat o zapnuté stereo ani světla, protože se vše samo vypne, jakmile odejde. Ve sprše pustí vodu, která má automaticky teplotu nastavenou specificky dle Joeovi preference 36 °C i přes to, že 20 min před ním se sprchovala jeho přítelkyně, která si libuje v teplejší vodě. Po sprše jde do ložnice a ulehá do postele, zatímco se kontroluje, jestli jsou všechny dveře zamčené, okna zavřená a zapíná se alarm pro případný pohyb ve spodním patře. A jak mohlo být vše uzpůsobené Joeovím preferencím a vše zapnuté ještě před jeho vstupem do

domu? Protože zvonek u dveří má kameru s rozpoznáváním obličeje - Joea tedy poznal a vše nastavil.

Takto tedy může vypadat automatizace v domácnosti, která zpříjemní život a odprostí Vás od spousty všedních věcí. Vše nastavené dle osobních preferencí a to nejen určité rodiny, ale na úrovni jednotlivců v domácnosti.

2.2.3 Bezpečnost a soukromí

Při výběru platformy by důležitým kritériem měla být bezpečnost. Na první pohled se to však nemusí zdát být důležité. Co se může stát, když bude s platformou komunikovat čidlo pohybu a někdo se dokáže dostat k těmto údajům? Například pro zloděje mohou být taková data zlatý důl, protože bude přesně vědět, kdy je dům prázdný.

Bezpečnost je potřeba zde sledovat hned na několika faktorech. Prvním je komunikační médium. Pokud zařízení komunikují bezdrátově, tak by komunikace měla být šifrovaná, aby se nedala jednoduše odposlechnout. Druhým faktorem je bezpečnost samotné platformy. Pokud je platforma dostupná pouze na interní síťi v domácnosti, tak bezpečnost na první pohled ohrožená není. Když se ale zamyslíme nad tím, kolik dnes máme doma chytrých zařízení, tedy takových, které dokáží komunikovat přes internet, tak zjistíme, že jich je velké množství. Dnes např. chytrou televizi má doma téměř každý a je otázkou, na kolik věříme výrobcům těchto zařízení, že kladou důraz na jejich bezpečnost. Stačí, aby nějaký vir napadl naší televizi či jiné zařízení a případný útočník má plný přístup k platformě pouze získáním přístupu do interní sítě. Proto by platforma měla využívat alespoň systém pro identifikaci, ideálně i autentifikaci a to nejen v případě, že je přístupná z internetu, ale i z vnitřní sítě.

2.2.4 Cílová skupina

Internet věcí lze využít napříč všemi sférami. Od jednoduché meteostanice, která měří teplotu, přes tvz. chytrou domácnost, kdy Vám lednička pošle nákupní seznam podle chybějících potravin, přes využití v průmyslu pro sběr různorodých dat a jejich následnou analýzu ať pro zvýšení kvality nebo detekci poruchy, ještě před tím než k ní dojde. Tato práce cílí na využití IoT v běžné domácnosti a implementaci platformy určené pro kutily a technické entusiasty, kteří chtějí mít svá data pod kontrolou, vytvářejí si různorodá zařízení a hledají platformu s důrazem na bezpečnost a flexibilitu.

2.3 Existující řešení

Tato kapitola se zabývá pohledem na aktuální řešení jak komerčních, tak i OpenSource. Poukazuje na výhody a nevýhody z obou světů, následně se zaměřuje na analýzu konkrétních platforem a jejich porovnáním.

2.3.1 Komerční řešení

Na trhu dnes existuje velké množství komerčních řešení od známých výrobců. Někteří jsou známí spíše výrobou hardwaru jako Philips a Xiaomi, jiní se zaměřují spíše na nabídku služeb a integraci zařízení ostatních výrobců pod svojí platformu jako Amazon nebo Google. Pro koncového zákazníka mají komerční řešení obrovskou výhodu v jednoduchosti nasazení a následné obsluhy. Stačí zakoupit centrální jednotku, libovolná zařízení od stejného výrobce a vše krásně funguje. Avšak problém nastává ve chvíli, kdy potřebují řešení škálovat či customizovat dle svých potřeb, protože si dodavatel za úpravy na „míru“ začne účtovat obrovské částky a zákazníkovi nezbývá nic jiného než platit. Sám si potřebné změny udělat nemůže, protože nemá zdrojové kódy a přechod k jinému produktu by znamenal obrovské náklady a problémy se stávajícími integracemi, protože různá řešení mívají různá rozhraní.

Aspekt bezpečnosti u uzavřených řešení bývá diskutabilní. Kvůli vysokým nákladům provádí pravidelné bezpečnostní audity málokdo. Výrobci sice vždy tvrdí, že bezpečnost je u nich na prvním místě, ale bohužel tento aspekt je v přímém kontrastu s jednoduchostí použití, což je pro výrobce mnohem důležitější, protože pokud se řešení dobře a jednoduše ovládá, tak mnohem spíše si ho zákazníci oblíbí, než pokud bude maximálně zabezpečeno, ale uživatel bude muset provádět úkony navíc čistě kvůli bezpečnosti, které mu na první pohled nepřináší přidanou hodnotu.

Od platformy očekáváme možnost vzdáleného ovládání, tedy přístup odkudkoli z internetu. Málokdo má však doma veřejnou IP adresu, aby si mohl celé řešení provozovat doma tzv. „self-hosted“. V praxi si tedy uživatel pořídí domů Bridge, který komunikuje s chytrými zařízeními v domácnosti a současně s cloudem výrobce, přes který lze přistupovat na platformu a ovládat všechny zařízení. Takové řešení se velmi osvědčilo díky jednoduchosti, protože neklade žádné nároky na uživatele jako např. veřejnou IP adresu. Problém však může nastat ve chvíli, kdy výrobce daného řešení po několika letech ukončí činnost a s tím přestane provozovat svojí cloudovou infrastrukturu, na které je závislý Bridge a vzdálený přístup z internetu. V lepším případě bude zachována funkčnost v lokální síti, v horším přestane řešení fungovat úplně. Najednou uživateli zbyde doma spousta funkčního (po fyzické stránce) hardwaru, který nemůže využívat.

Výše jsem nastínil nejhorší možný scénář, který naštěstí v poslední době již přestává platit, protože výrobci společně vytvářejí otevřené standardy pro komunikaci, které by měli zaručit kompatibilitu zařízení napříč jednotlivými výrobci. Bohužel standardů vzniká současně více a ne všichni je plně dodržují, takže nekompatibilita ještě bude delší dobu přetrvávat, i když ne v takovém měřítku jako před pár lety. Kromě rozdílných protokolů je také nekompatibilita v různých technologiích přenosu mezi něž patří Wifi, Bluethooth, LoRa, Zigbee či Sigfox.

2.3.2 OpenSource řešení

OpenSource řešení mají mezi širší veřejností špatnou reputaci, protože na rozdíl od komerčních „Plug and Play“ produktů většinou vyžadují určité povědomí o dané problematice. Je to způsobeno tím, že se snaží pokrýt celou doménu stejně jako komerční řešení, ale oproti nim se zlomkem vývojářů a financí. Následkem toho není prvotní nastavení pro laika zcela přímočaré a může se střetnout s problémy. Avšak překonání prvotních nesnází přináší následně spoustu pozitiv.

Jedním z nejatraktivnějších lákadel je zcela jistě cena. OpenSource řešení jsou zpravidla zcela zdarma, případně nabízejí placenou podporu. Mně osobně na OpenSource nejvíce zaujala komunita. Pokud se projekt dostane do určité známosti, tak kolem něho začně vznikat komunita lidí, primárně technologických nadšenců ale i lidí z IT praxe, kteří mezi sebou komunikují a spolupracují na vylepšení daného řešení, ať už přímo (napsání části funkcionality) nebo nepřímo (komunikace s vývojáři). Potom i obyčejný uživatel, který chce řešení využít, tak při objevení potíží, může požádat komunitu o pomoc a protože to jsou nadšení lidé, jsou velmi ochotní.

Pokud máme dostatečné technické znalosti, můžeme si prohlédnout přímo zdrojové kódy a sami si zhodnotit kvalitu i bezpečnost. U větších projektů to však již není tak úplně možné při desítkách tisíc řádků kódu, ale existují lidé, kteří tomu opravdu věnují čas a mohou tak objevit zranitelnosti. Dále OpenSource projekty bývají mnohem více sdílné ohledně architektury, kterou využívají a je možno se v dokumentaci dočíst, jak vlastně řešení funguje interně, na rozdíl od komerčních, kde je to tzv. „BlackBox“ (černá skříňka).

OpenSource platformy bývají postavené na systému Pluginů, tedy obsahují určitou základní sadu funkcí a dále lze funkčnost rozšiřovat pomocí instalace Pluginů. Ty mohou vytvářet přímo autoři nebo kdokoli jiný dle potřeb. Díky tomu jsou velmi robustní a podporují širokou škálu zařízení od různých výrobců napříč technologiemi a pokud ne, tak s trochou znalostí v programování si může každý dopsat plugin dle potřeb pro podporu daného zařízení.

2.3.3 Známé platformy

Tato sekce se zabývá analýzou 4 vybraných Platforem.

Blynk Blynk se označuje jako hardware-agnostic IoT platforma s white-label mobilními aplikacemi [15]. Umožnuje navrhnovat vlastní aplikace formou DragAndDrop pro ovládání zařízení, analýzu telemetrických dat a správu nasazených produktů ve velkém měřítku. Své řešení nabízejí jak pro domácí nasazení, tak i jako enterprise řešení pro větší firmy [15]. Mají 3 cenové tarify [16]:

- **Free** je omezený pouze pro osobní užití, obsahuje cloudový hosting, umožňuje připojit maximálně 5 zařízení zdarma a součástí je mobilní

aplikace pro Android a iOS.

- **StartUp** je určený pro komerční využití a cenou začíná na $415/měsíc. Součástí je deployment vlastních aplikací na AppStore/Google Play, neomezený počet zařízení a uživatelů, garantované podpora
- **Business** začíná na $1000/měsíc a nabízí navíc OTA (vzdálené) aktualizace koncových zařízení, webové rozhraní, datovou analýzou a dalších funkce.

Hardware-agnostic znamená, že nejsou omezeni pouze na určitý hardware a umožňují připojit v podstatě libovolné zařízení. Pro připojení maji definované rozhraní nad jednotlivými protokoly. Podporují vlastní TCP/IP protokol, WebSocket, HTTP a nově i MQTT (zatím k němu nemají ale dokumentaci). Dávají k dispozici knihovny pro různé hardwarové platformy, takže připojení k platformě je potom otázka dvou řádků kódu. K dispozici je velmi přehledná a detailní dokumentace. [17]

Nativní aplikace pro iOS a Android umožnuje vytvářet vlastní dashboardy pomocí již předpřipravených Widgetů, kterých je opravdu velké množství, ale jsou placené za tzv. Energii, což je měna, kterou lze dobíjet za peníze. Dále definovat vlastní widgety a upravit chování celé aplikace. Následně lze takto upravenou aplikaci vyexportovat a přímo nahrát na Google Play a AppStore pod vlastním názvem. Tento přístup nabízí elegantní možnost pro tvorbu vlastního řešení, které následně je možné nabízet jako vlastní produkt. [15]

Výhodou cloudového řešení je přístup k platformě odkudkoliv z internetu. Následně je ale funčknost odkázána na dostupnost internetového připojení a představa dat v cloudu se nemusí každému líbit. Pro tento případ je možnost hostovat si vlastní Blynk server, který je dostupný jako OpenSource server napsaný v Javě. [18]

Thingspeaks ThingSpeak™ je analytická IoT platforma od MathWorks, tvůrců známého výpočetní platformy MATLAB. Jedná se o hardware-agnostic platformu s webovým rozhraním, která se plně zaměřuje na analýzu dat. Je ideální pro lidi se zkušeností s MATLAB, protože je postavena právě na této platformě. Umožňuje v cloudu sběr dat, jejich analýzu přímo pomocí MATLAB kódu, vizualizaci dat a definování reakcí. Pro různé hardwarové platformy mají připravené knihovny a nativě podporují komunikace pomocí protokolů HTTP a MQTT. [19]

Své řešení nabízejí podle různých tarifů [20], jež jsou omezeny podle maximálního počtu zpráv, počtu kanálů, do kterého posílají zařízení zprávy a minimálního časového odstupu mezi zprávami v rámci jednoho kanálu. Dva základní tarify:

- **Free** 8 200 messages/day, počet kanálů 4, interval mezi zprávami 15s, maximální doba běhu MATLAB kódu 20s.

- **STANDARD** 90 000 messages/day, počet kanálů 250, interval mezi zprávami 60s, MATLAB maximální doba běhu 20s.

Home Assistant OpenSource domácí automatizace, která klade lokální kontrolu a soukromí na první místo - takto se prezentuje Home Assistant. Tato platforma není tolik zaměřena na koncová zařízení jako předchozí, ale funguje jako integrátor komerčních/OpenSource řešení pod jednotné rozhraní. Obsahuje systém pro tvorbu automatizace, tedy vytváření reakcí na jednotlivé akce. Dokáže se napojit buď přímo na jednotlivá zařízení nebo na jejich Bridge a umožnit ovládání všech zařízení od různorodých výrobců, kteří často vynucují použití vlastní aplikace, pod jednotné rozhraní jak webové, tak ve formě nativní aplikace. Integrace je řešena pomocí pluginárního systému, kde jeden plugin obsahuje integraci skupiny zařízení jednoho výrobce/Bridge. Většina pluginů vzniká přímo od komunity této platformy. V době psaní této práce obsahuje 1743 pluginů. [21]

Celá platforma je zdarma a pro její zprovoznění stačí Raspberry Pi, na SD kartu nahrát předpřipravený image a zapnout. Prvotním nastavením vás následně provede webové rozhraní nebo nativní aplikace, záleží na preferenci. [22]

Řešení podporuje velké množství komerčních produktů, mezi nejznámější patří Ikea TRÅDFRI, Philips Hue či Google Assistant [23]. Samozřejmě podporují i OpenSource projekty mezi nejznámější patří ESPHome [24], což je framework pro konfiguraci ESP chipů (ESP8266/ESP32), který řeší vrstvu komunikace a zapojení do platformy - stačí pouze dodefinovat chování na určité události a chytré zařízení je připravené.

OpenHAB OpenSource projekt s dlouhou historií, aktuálně ve třetí verzi, který vznikl již v roce 2010. Cílí na stejný segment jako Home Assistant, tedy propojení existujících řešení pod jednotné rozhraní a jejich automatizaci. Jedná se o hardware agnostic platformu, která komunikuje přímo s koncovými zařízeními nebo příslušným Bridge. V základu obsahuje více funkcionalit, zatímco Home Assistant je spíše minimalistický. OpenHAB je založený na systému doplňků (aktuálně 324 [25]), které rozšiřují funkcionalitu a vyvíjejí jej autoři a komunita. Od prvopočátku projektu je zde kladen velký důraz na nativní aplikace na rozdíl od Home assistantu, který dlouhou dobu žádnou oficiální aplikaci neměl. Webové rozhraní je samozřejmostí. Velkou výhodou je možnost využití cloud instance zcela zdarma, buď jako plnohodnotnou platformu nebo pouze pro přístup z internetu k vlastní instanci. [26]

Prvotní instalace je stejně jednoduchá jako u Home Assistentu. Rozdíl přichází při přidávání jednotlivých zařízení, kde je proces trochu komplikovanější. OpenHab se snaží nabídnout pokročilejší funkcionalitu, která však částečně zesložiťuje jednotlivé procesy. [27]

Dokumentace projektu je na velmi vysoké úrovni s velmi detailním popisem. Pravděpodobně díky tomu, že projekt existuje již 10 let a má silnou základnu v komunitě i přes to, že dle porovnání aktivity na GitHubu v počtu přispěvatelů (86 vs. 2 444) je oproti té, kterou má Home Assistant, mnohonásobně menší.

2.3.4 Porovnání

Jednotlivé platformy se některými funkcemi překrývají a v jiných jsou zase jedinečné. Při výběru je důležité si stanovit, na co platformu chceme využívat a jaké funkce vyžadujeme. Tabulka 2.1 obsahuje přehledné porovnání analyzovaných platforem v sedmi kategoriích:

- Podpora komerčních produktů - zda lze k platformě připojit zakoupené zařízení od výrobců jako Philips, Xiaomi a jiných.
- Vlastní zařízení - zda lze připojit vlastní tzv. DYI (vyrobená) zařízení.
- Hosting - zda lze provozovat platformu na vlastním hardwaru.
- ACL - zda platforma obsahuje systém pro nastavení oprávnění pro přístup uživatelů k jednotlivým zařízením.
- Nativní aplikace - zda platforma má oficiální nativní aplikace pro telefon (iOS či android).
- Správa zařízení - zda platforma umožňuje spravovat zařízení jako taková, ve smyslu vzdálené aktualizace, restartování, zobrazení případných chyb či přímo sledování komunikace.
- Cena - zda lze využívat řešení zcela či v omezené formě zdarma.

Blynk primárně cílí na podnikatelský segment a nejvíce se hodí firmám, které chtějí na této platformě vystavět své vlastní řešení, které následně budou přeprodávat pod svojí vlastní značkou. To díky přímé možnosti exportu aplikace na AppStore a Google Play, hromadné správě zařízení a ACL (seznam oprávnění vázaný k zařízení, který specifikuje, kdo k němu může přistupovat a jaké operace provádět).

ThingSpeaks míří primárně na zpracování dat díky svému ekosystému postaveném kolem prostředí MATLAB. Pro veškeré zpracování, analýzy a vizualizace stačí znalost MATLAB, který je světově známý a velmi oblíbený mezi akademiky.

Home Assistant je progresivní OpenSource platforma, která umožní integraci komerčních řešení pod jednotné rozhraní a domácí automatizaci s příjemným uživatelským rozhraním.

OpenHab je projekt s dlouho historií. Funkčně se velmi podobá Home Assistantu, ale snaží se uživatelům nabídnout více funkčnosti. Uživatelské rozhraní je občas trochu složitější.

Platforma	Podpora komerčních produktů	Vlastní zařízení	Hosting	ACL	Nativní aplikace	Správa zařízení	Cena
Blynk	Ne	Ano	self-hosted, cloud	Pouze Enterprise	iOS, android	Ano	Omezený Free plan
ThingSpeaks	6 dodavatelů (primárně LoRa)	Ano	cloud	Ano	Pouze pro náhled na data (android)	Ne	Omezený Free plan
Home Assistant	pomocí pluginů (1743)	3rd party knihovny	self-hosted	Ano	iOS, android	Ne	Zdarma
openHab	pomocí doplňků (324)	3rd party knihovny	self-hosted, cloud	Ne	iOS, android	Ne	Zdarma

Obrázek 2.1: Vzájemné porovnání jednotlivých Platforem

2.3.5 Závěrečný verdikt

Blynk je první platforma, se kterou jsem se střetl ve světě IoT před třemi lety a bohužel první dojem pro mě byl poměrně negativní. Mnohé se od té doby změnilo, ale nepřímá podpora MQTT protokolu a především nutnost platit řešení mě od této platformy odrazuje. Thingspeaks je hezké řešení, které splňuje většinu mých představ, ale úzká integrace s MATLAB a nutnost jeho znalosti pro zpracování dat, je pro mne překážkou, ať z hlediska, že MATLAB nepoužívám, tak více z pohledu ceny MATLAB prostředí a celého ekosystému. Sám se považuji za OpenSource zastánce a proto mě to táhne k těmto řešením. HomeAssistant je velmi progresivní a zajímáva platforma, která je ale primárně určena pro nasazení v lokální síti (nepočítá s nutností autentizace jednotlivých zařízení), zatímco já bych chtěl primárně platformu provozovat jako řešení, kde se stačí zaregistrovat a každý kutil může přidávat vlastní zařízení a veškerý tok dat bude mezi uživateli bezpečně oddělen. OpenHAB řešení mě velmi zaujalo, především možnost hostingu cloudového řešení zcela zdarma. Bohužel chybějící ACL je pro mne nepřekonatelnou překážkou, protože chci platformu využívat pro více uživatelů a tedy definovat jednotlivá oprávnění mezi uživatelem a zařízením. Proto jsem se rozhodl vytvořit si vlastní řešení, které mi dá prostor realizovat vše dle svých představ s důrazem na bezpečnost.

2.4 Vlastní řešení

Tato sekce se věnuje definici požadavků na vlastní řešení a výběru vhodných technologií.

2.4.1 Záměr

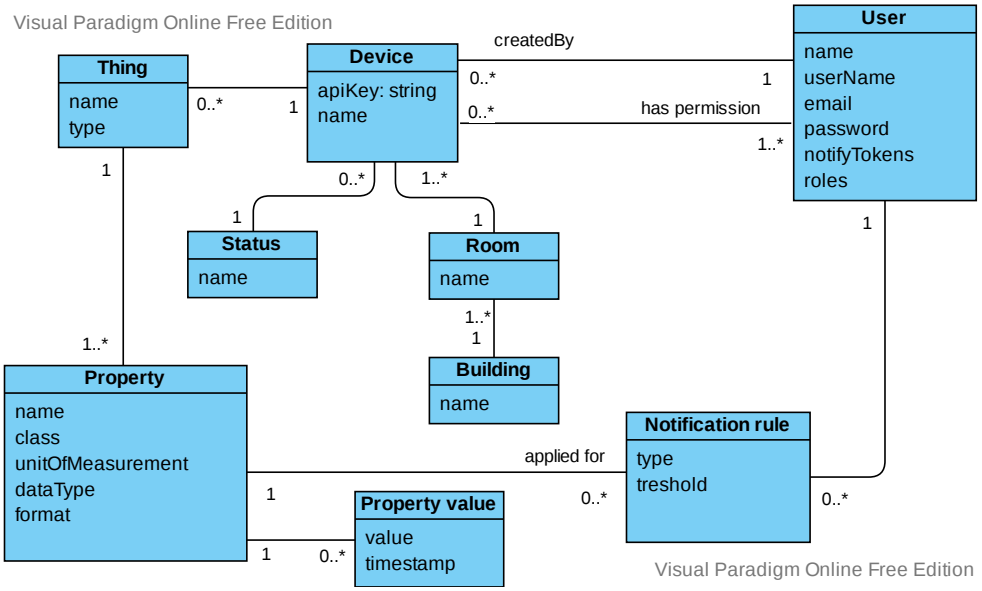
Vytvoření otevřené IoT platformy, která bude určena pro nejrůznější elektrotechnické kutily a technické nadšence. K dispozici bude zdarma veřejná instance sloužící primárně pro uživatele, kteří si chtějí platformu jednoduše a rychle vyzkoušet a nebo chtějí provozovat pouze pár zařízení. Pro ty, kteří chtějí mít plnou kontrolu nad svými daty a být nezávislí na připojení k internetu, bude k dispozici možnost hostingu celého řešení na vlastním hardwaru. K platformě půjde připojit různorodá zařízení a bude vytvořeno schéma, pomocí kterého zařízení popíší platformě vlastní funkčnost/schopnosti. Na základě těchto informací se automaticky uživateli vygeneruje webové rozhraní ke sledování a ovládání jeho zařízení.

Na bezpečnost bude kladen vysoký důraz. Primárně bude založena na uživatelských účtech, které budou mít oprávnění pouze ke svým zařízením, případně těm, ke kterým dostali oprávnění od jiných uživatelů. Každý uživatel by měl mít k dispozici vlastní izolované prostředí, v rámci kterého budou

jeho zařízení komunikovat. Uživateli bude umožněno sledovat všechny zprávy posílané mezi jeho zařízeními a platformou.

2.4.2 Popis domény

Tato kapitola obsahuje popis jednotlivých entit, se kterými platforma pracuje.



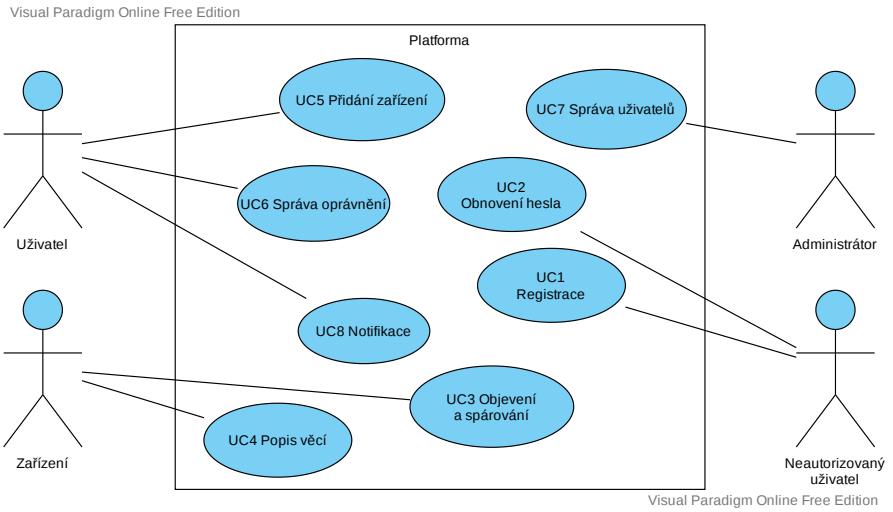
Obrázek 2.2: Doménový model

- **Uživatel (User)** - osoba, která interaguje s webovým rozhraním.
- **Zařízení (Device)** - fyzické zařízení, které komunikuje s platformou a dále se dělí na věci.
- **Stav (Status)** - v jakém stavu se zařízení nachází (např. odpojeno).
- **Věc (Thing)** - logické uskupení vlastností, např. meteostanice.
- **Vlastnost (Property)** - určitá veličina, jejíž hodnota se odesílá na platformu (např. teplota), která případně umožňuje být platformou nastavena.
- **Hodnota vlastnosti** - hodnota v určitém časovém okamžiku.
- **Budova (Building)** - místo, které se dále dělí na místnosti.
- **Místnost (Room)** - uskupení více zařízení.
- **Notifikační pravidlo** - za jaké podmínky se má uživateli odeslat notifikace.

2.4.3 Případy užití

Tato kapitola popisuje identifikované případy užití, které současně slouží jako podklad funkčních požadavků kladených na řešení. Figurují v nich následující aktéři:

- Neautorizovaný uživatel - představuje nepřihlášeného uživatele webového rozhraní
- Uživatel - přihlášený uživatel webového rozhraní
- Administrátor - autentizovaný uživatele s vyšším stupněm oprávnění
- Zařízení - koncové zařízení komunikující s platformou



Obrázek 2.3: Případy užití

UC1 Registrace uživatele Neautentizovaný uživatel vyplní registrační formulář obsahující jméno, přijmení, uživatelské jméno, heslo a email. Při zpracování požadavku na serveru bude zajištěna unikátnost uživatelského jména a emailu napříč databází. Uživatel bude informován o úspěchu/neúspěchu akce. Po úspěšné registraci bude automaticky přihlášen, pokud nezrušil ve formuláři zaškrtávátko „Automaticky přihlásit“. Následně mu bude odeslán uvítací email na zadanou emailovou adresu.

UC2 Obnovení hesla Součástí přihlašovacího formuláře bude odkaz na stránku pro obnovení zapomenutého hesla, kde bude uživatel dotázán na emailovou adresu, kterou použil při registraci. Po zadání, pokud daná emailová adresa je součástí některého uživatelské účtu, bude odeslán email s odkazem pro obnovu hesla. Na tomto odkazu bude uživatel vyzván k zadání hesla nového.

UC3 Objevení a spárování nového zařízení Zařízení, pokud ještě není spárované s platformou, po zapnutí vyzve uživatele k zadání svého uživ. jména k platformě. Toto zadání bude umožněno vytvořením Wifi přístupového bodu, na kterém poběží kaptivní portál - po připojení telefonem/počítačem se automaticky zobrazí webová stránka s formulářem pro zadání údajů. Následně se zařízení připojí k platformě, ohlásí jaké má věci, popíše jejich vlastnosti a bude ji informovat, kterému uživateli (podle zadaného uživ. jména) má zobrazit možnost přidání nového zařízení. Pokud si uživatel dané zařízení přidá (viz. UC5), platforma následně odešle zařízení API klíč, které si ho uloží a přihlásí se pomocí toho klíče k platformě (nyní je spárované).

UC4 Popis věcí Zařízení při ohlašování definice věcí a jejich vlastností musí oznámit mimo jiné typ věci. Platforma bude podporovat kromně generického (generic) typu další 3 typy věcí, pro které bude speciální zobrazení v uživatelském rozhraní:

- **Switch** - přepínač, který se nachází ve stavu on/off. V rozhraní bude věc reprezentována dvoustavovým přepínačem, který při kliknutí odešle změnu o stavu na druhý, než ve kterém se aktuálně nachází. (využití např. vypínač světla)

- **Activator** - spínač, který má pouze jeden stav. V rozhraní bude věc reprezentována tlačítkem, které na stisk odešle aktivaci zařízení. (využití např. ovladač pojízdné brány)

- **Sensor** - v rozhraní bude reprezentován jako widget zobrazující aktuální hodnotu první vlastnosti. Po rozkliknutí se zobrazí graf vizualizující průběh hodnoty v čase za posledních 24h.

- **Generic** - obecný typ, u kterého zařízení popíše strukturu, datové typy a názvy příslušných vlastností. V uživatelském rozhraní bude věc reprezentována jako Widget, který po kliknutí zobrazí Dialogové okno umožňující zobrazení a ovládání všech vlastností dle konfigurace.

UC5 Přidání zařízení Uživateli na stránce „Správa zařízení“, v případě že bude detekováno nové zařízení, v sekci *Přidat zařízení* se zobrazí (bez nutnosti aktualizace stránky) možnost přidat nové zařízení. Při kliknutí na tlačítko přidat se zobrazí jednoduchý formulář pro zadání umístění a názvu zařízení - bude předvyplněn název, který ohlásilo zařízení. Uživatel formulář potvrdí, systém následně vytvoří dané zařízení, přidá uživateli k němu oprávnění a na stránce „Ovládání“ už bude uživatel moci sledovat aktuální stav věcí a případně je i ovládat (pokud to umožňují).

UC6 Správa oprávnění Uživatel na stránce „Správa zařízení“ v sekci *Správa* bude mít zobrazena všechna zařízení, ke kterým má uživatel oprávnění. Rozhraní umožní pro každé zařízení, ke kterému má příslušné oprávnění pro správu, jeho smazání a pomocí formuláře editaci - názvu, umístění a změnu oprávnění pro jednotlivé uživatele. Tyto oprávnění budou rozděleny na tři úrovně:

- Čtení - uživatel může zobrazit veškeré údaje o zařízení.
- Ovládání - uživatel může zařízení ovládat.
- Správa - uživatel může editovat veškeré informace o zařízení (včetně oprávnění).

UC7 Správa uživatelů Administrátor má k dispozici stránku „Správa uživatelů“, kde se mu zobrazí seznam všech registrovaných uživatelů. Jednotlivé uživatele může smazat a pomocí formuláře editovat všechny jejich osobní údaje a změnit heslo pro přihlášení.

UC8 Notifikace Rozhraní umožní uživateli nastavit pravidlo pro libovolnou věc, při kterém se odešle Web Push notifikaci (technologie umožňující serveru odeslat upozornění do prohlížeče klienta [28]) na jeho zařízení. Pro nastavení bude zobrazený formulář umožňující výběr z vlastností dané věci, po vybrání se zobrazí výběr akce, při jejímž splnění chce uživatel obdržet notifikaci (překročení hodnoty / vždy / hodnota bude rovna) a případné pole pro zadání limitní hodnoty. Dále půjde zobrazit rozšířené nastavení pro konkrétní notifikační pravidlo umožňující nastavení času a konkrétních dnů v týdnu, kdy bude pravidlo platné (ve výchozím stavu bude vždy). Těchto pravidel si bude moci nastavit libovolný počet pro každé zařízení, ke kterému má oprávnění pro čtení.

2.4.4 Nefunkční požadavky

N1 Řešení spustinelné na Linux systému Systém bude možno provozovat na Linuxovém serveru (Debian) a také na platformě Raspberry Pi (verze 3B+/4, OS Raspbian).

N2 Responzivní webové rozhraní Aplikace bude nabízet responzivní webové rozhraní přizpůsobené pro zobrazení na mobilních zařízeních i stolních počítačích. Uživatelské rozhraní bude kompatibilní s prohlížeči Mozilla Firefox verze 80, Chrome verze 80 a Safari na iOS. Dále bude implementovat tzv. PWA (Progresivní webová aplikace) - bude využívat cache pro statické soubory pro rychlé načítání, spustitelné offline a na zařízení Android půjde v aplikaci chrome přidat na plochu a následně vypadat jako nativní aplikace.

N3 Rozhraní realizováno jako SPA Single page application (SPA) je webová aplikace, která utilizuje JavaScript tak, aby při interakci v rámci aplikace se nemusela načítat celá stránka, ale pouze chytře překresluje potřebné části. Výsledkem je mnohem přijemnější uživatelský zážitek, než při čekání na stažení a překreslení celé stránka po kliknutí na odkaz.

N4 Validace Uživatel v průběhu vyplňování formulářů v rozhraní obdrží interaktivní zpětnou vazbu v případě zadání nevalidních údajů. Interaktivní zpětnou vazbou jsou myšleny následující scénáře při průchodu formuláře:

- Zadání nové hodnoty a opuštění pole - bude provedena validace a v případě nevalidního vstupu, bude uživatel vizuálně upozorněn.
- Editace již zadané hodnoty v poli - validace bude provedena po každé změně (stisknutí klávesy), uživatel bude vizuálně upozorněn v případě nevalidního vstupu.

N5 Koncová zařízení Bude specifikován protokol, pomocí kterého s platformou budou zařízení komunikovat včetně definování schématu komunikace. Platforma umožní připojení libovolného zařízení, pokud použije definovaný protokol a bude se řídit schématem pro komunikaci.

N6 Výkonnostní požadavky Systém bude stabilní a zvládne obsluhovat stovku zařízení, kde každé bude odesílat změnu stavu s periodicitou 30 vteřin. Při tomto dlouhodobém zatížení nebude docházet k pádům systému ani k výraznému zpoždění komunikace (RESTful požadavky pod 500 ms).

N7 Konfigurace systému Veškerá konfigurace týkající se externích služeb jako jméno a heslo do databáze, číslo portu pro komunikaci atd. bude konfigurovatelné pomocí promněných prostředí (env variables). Detailní popis proměnných bude obsažen v instalační příručce.

2.4.5 Vybrané technologie

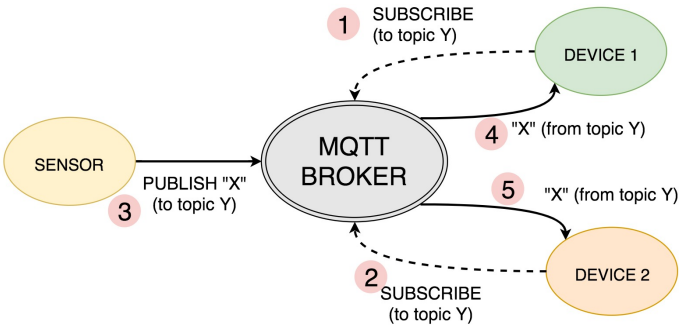
Tato sekce se věnuje výběru protokolu pro komunikaci se zařízeními a programovacího jazyka, ve kterém bude řešení implementováno včetně výběru příslušných knihoven.

2.4.5.1 Komunikační protokol

Komunikačních protokolů je velké množství a jeho výběr přímo závisí na použitém přenosovém médiu. Při použití specializovaných sítí jako LoRa nebo Zigbee, nemáme moc velkou flexibilitu ve výběru. Zařízení podporující tyto specializované sítě jsou poměrně drahá, ale umožňují běh na baterii. Zatímco

využití Wifi sítě nám dává obrovskou flexibilitu ve výběru protokolů a zařízení podporující připojení k Wifi nebyly nikdy více cenově dostupné jako dnes. Primárně z finančních nároků a možnosti využití stávající Wifi infrastruktury v domácnosti jsem se rozhodl pro Wifi jako bezdrátové médium. Současně díky přímé podpoře IP protokolu nebudu muset vytvářet bridge mezi serverem a sítí se zařízeními jako např. při použití Bluetooth či LoRa.

Z protokolů pro komunikaci je dnes nejpoužívanější HTTP, který ale nativně nepodporuje obousměrnou komunikaci, kdy zařízení může poslat zprávu serveru a stejně tak server zprávu zařízení, kterou pro ovládání zařízení budu potřebovat. Z obousměrných protokolů se velmi osvědčil WebSocket, který lze velmi snadno kombinovat s HTTP. Jedná se o protokol postavený nad TCP, který naváže spojení a obě strany mohou posílat zprávy [29]. Je to velmi pěkné řešení pro posílání zpráv, ale neumožňuje nativně systematickou filtraci nebo odběr pouze určitých zpráv a nepočítá s během na nespolehlivých zařízeních. Proto přímo pro IoT vznikl otevřený síťový protokol MQTT, kterým jsem si zvolil jako komunikační protokol, jehož specifikaci vydává nezisková organizace *OASIS*. Využívá asynchronní vzor *publish-subsribe* (v podstatě rozděluje akci na dvě události - vytvoření akce a její obsluhu) a byl speciálně navržen pro potřeby běhu na jednoduchých embeded zařízení s minimálním datovým tokem. Podrobný přehled všech protokolů využívaný pro IoT viz. [30].



Obrázek 2.4: Ukázka komunikace MQTT [1]

MQTT se vyvíjí již od roku 1999 a momentálně nejpoužívanější verzí je 3.1.1, pro kterou vznikla specifikace v roce 2014. Protokol primárně běží nad TCP/IP, ale lze využít v jakékoliv síti, kde je zaručeno správné pořadí dat, beztrátovost a obousměrnost komunikace. Protokol definuje 2 typy entit: „MQTT broker“ a „klient“. MQTT broker je server, který přijímá všechny zprávy od připojených klientů a přeposílá je příjemncům (klientům). MQTT klient je jakékoliv zařízení (od embeded až po server), které komunikuje s brokerem přes síť. [31]

Posílaná data jsou hierarchicky rozdělena do tzv. témat. Téma je textový řetězec o maximální délce 65 536 Bytů s oddělovačem lomítko (ukázka „house/bedroom/light“). Pokud klient chce odeslat (publish) data, tak pošle

zprávu brokeru s daty a tématem, do kterého zpráva patří. Broker potom zprávu odešle všem klientům, kteří jsou přihlášeni k odběru (subscribe) z daného tématu. O odběr se klient musí přihlásit a to buď přímo specifikuje plný název topicu nebo částečný s použitím zástupných znaků. MQTT počítá s případnou nespolehlivostí koncových zařízení či sítě, a proto umožňuje klientovi při přihlášení definovat „Last Will and Testament“ (LWT). Při přihlášení klient oznámí téma a zprávu, která se odešle v případě nesprávně odpojeného klienta (výpadek sítě / chyba zařízení). Takto lze notifikovat ostatní zařízení, že došlo ke ztráně spojení s daným klientem. [31]

Broker podporuje 3 třídy QoS (Quality of service), kterou lze specifikovat pro každou zprávu jednotlivě v závisloti na její důležitosti. Seřazeny jsou vzestupně dle náročnosti na systém (overhead) [31]:

- **0 - Maximálně jednou** - zpráva je odeslána pouze jednou a klient ani broker nijak nepotvrzují její přjetí
- **1 - Alespoň jednou** - zpráva je odeslaná několikanásobně, dokud není potvrzené její přijetí
- **2 - Právě jednou** - odesílatel a příjemce navazují dvoucestný handshake, aby bylo zaručeno přijmutí zprávy právě jednou

2.4.5.2 Programovací jazyk

Programovacích jazyků jsou dnes na trhu stovky a každý má své specifické pozitivní i negativní vlastnosti. Při výběru je tedy vždy potřeba zohlednit jeho přínos pro použití na daném projektu. Kromě jazyka samotného je příhodné analyzovat dostupné knihovny a frameworky, které lze v projektu použít a značně tak urychlit celkový vývoj. Pro implementaci této práce byl zvolen jazyk JavaScript konkrétně jeho nadmnožina TypeScript a to z několika důvodů.

Vzhledem k povaze zvoleného protokolu pro komunikaci se zařízeními, jenž je založený na asynchronních zprávách, je JavaScript velmi vhodný, protože je založený na asynchronní event-driven [32] architektuře. Tato architektura nabízí velice elegantní přístup pro zpracování akcí, kde se musí čekat na výsledek jako např. u síťové komunikace či právě asynchronních zpráv. V tradičním jazyce jako Java nebo C++ se toto čekání musí řešit pracným vytvořením nového vlákna, které čeká na výsledek a následným zpracováním. V NodeJS je programátor od této problematiky odstíněn a může se tak plně věnovat tvorbě aplikační logiky, aniž by měl znalosti a zkušenosti s vícevláknovým programováním.

JavaScript je jediný programovací jazyk, který umožňuje psaní jak serverových aplikací, tak i uživatelského rozhraní formou webové stránky a jeho přímé vykonávání ve webovém prohlížeči klienta. Využití jednotného jazyka pro vývoj serveru i uživ. rozhraní přináší obrovskou výhodu v podobě možnosti sdílet nejenom definice pro objekty, ale i přímo části kódu. Toto je velmi

vhodné například pro jednotné validace formulářů, různé datové transformace a sdílení aplikační logiky pro frontend „optimistické aktualizace“ (aktuální trend, nečekat na potvrzení požadavku ze serveru, ale rozhraní aktualizovat, jako by požadavek byl úspěšný a pouze v případě neúspěchu zobrazit stav ze serveru). Dále jednotný jazyk umožňuje programátorům při vývoji v případě potřeby pohodlně pracovat na obou částech aplikace, aniž by se museli učit nový jazyk.

TypeScript Je nadmnožina JavaScriptu, která navíc přidává komplexní typový systém [33] a rozhodl jsem se ho využít jako hlavní programovací jazyk jak pro backend tak i frontend. Jedná se o OpenSource jazyk vyvíjený společností Microsoft, který jeho vznikem chtěl usnadnit přechod C# a .NET vývojářům k webovým aplikacím [33]. Mnoho lidí z JavaScript komunity považuje TypeScript jako kontroverzní počin, protože přidává složitost k velmi elegantnímu jazyku a zvyšuje časovou náročnost vývoje. Já jsem dlouhou dobu tento názor také zastával, ale v posledních letech při práci na větších projektech a díky zkušeností z jiných jazyků (včetně striktně typových jako C++ a Java), jsem změnil svůj názor ve prospěch TypeScriptu. Souhlasím, že na první pohled prodlužuje dobu vývoje. Programátor musí psát věci navíc oproti čistému JavaScriptu, ale v dlouhodobém životním cyklu projektů se tato práce „na víc“ mnohonásobně vrátí. A to v podobě statické kontroly typů, která minimalizuje riziko pádu aplikace a umožňuje lepší statickou analýzu kódu, a dále jako největší přínos pro mne jako programátora TypeScript přináší funkční „našeptávání“ ve vývojovém prostředí, které pro JavaScriptu i přes veškeré snahy bohužel funguje ve velmi omezené míře.

2.4.5.3 Server

Pro běh JavaScript na straně serveru existuje několik prostředí např. SpiderMonkey, NodeJS či Rhino. Pro realizaci bylo zvoleno prostředí NodeJS, které má pravděpodobně aktuálně největší a nejaktivější komunitu ze všech serverových prostředí pro běh aplikací napříč programovacími jazyky. Pro správu knihoven používá balíčkovací systém npm (jsou i jiné alternativy), ze kterého se stal největší ekosystém na světě, který je zastřešený neziskovou společností „npm, Inc.“ provozující centrální repozitář se všemi dostupnými moduly pro NodeJS. Díky sve centralizaci je velmi jednoduchý na používání, ale v posledních letech, kdy se JavaScript zpopularizoval a nyní je jedním z nejoblíbenějších jazyků [34], ukázala se centralizace jako poměrně nešťastné řešení kvůli vysokým nákladům na provoz infrastruktury. Pro představu velikosti ekosystému: npm v roce 2020 obsahoval 1$\tilde{2}$00$\tilde{0}$00 modulů a druhý největší systém RubyGems „pouhých“ 350$\tilde{0}$00 [35]. Všechny moduly jsou k dispozici zcela zdarma a díky takto aktivní komunitě lidí, kteří dávají k dispozici své knihovny ostatním, je vysoce pravděpodobné, že pokud chceme řešit nějaký problém, tak na něj již existuje knihovna.

ExpressJS Platforma bude implementovat RESTful webové rozhraní a pro jeho implementaci byl zvolen minimalistický framework ExpressJS. První jeho verze vznikla již v roce 2010 a dodnes je mezi vývojáři velmi oblíbený a v mnohém ovlivnil směr vývoje většiny frameworků. Jeho největší výhoda je vysoká flexibilita. Nabízí pouze základní definici způsobu pracování s HTTP požadavky a možnost registrovat tzv. middleware - software, který rozšiřuje funkcionalitu. Veškerá funkcionalita je dodávána pomocí middlewarů, které jsou k dispozici jako moduly. Vývojář si tedy může výsledný server poskládat přesně dle svých představ, kterých existují desítky, vytvořených přímo od autorů a další stovky od komunity. [36]

AgendaJS Knihovna pro perzistentní plánování úkolů pro NodeJS [37]. Umožňuje zpracování/plánování/perzistenci úkolů a jejich opětovné spouštění v případě chyby [37]. Tato knihovna bude primárně využita pro zajištění odeslání emailů a pro spouštění případných plánovaných akcí. Proč v souvislosti s odesláním emailů? Jejich zpracování je závislé na třetí straně - emailovém serveru, který nemusí být vždy dostupný. Pokud systém bude mět odeslat email, tak tímto způsobem bude zajištěno, že i v případě selhání bude email opětovně odeslán, jakmile to bude možné.

Socket.IO Pro zajištění aktualizace rozhraní v reálném čase bude využita knihovna SocketIO umožnující navázání obousměrného spojení pro real-time komunikaci. Jedná se o velice populární a časem ověřené řešení, které zajišťuje kompatibilitu i s prohlížeči nepodporujícími moderní technologii WebSocket.

2.4.5.4 Uživatelské rozhraní

Prvním bezesporu světoznámým průkopníkem ve světě JavaScriptu pro tvorbu uživatelského rozhraní byla knihovna *jQuery*, která existuje dodnes, ale spíše se již považuje za přežitek doby. Dnes existuje obrovské množství Frameworků a knihoven pro tvorbu frontendu, ať pro tvorbu na straně serveru nebo přímo na straně uživatele v prohlížeči. Trend dnešní doby je přesouvat generování rozhraní na stranu uživatele, jak kvůli snížení výkonnostních nároků na server, tak spíše kvůli lepší odezvě a uživatelskému zážitku. Mezi nejznámější JavaScriptové frameworky patří bezpochyby Angular, Vue.js, Svelte a nesmím zapomenout na React, který je sice knihovna, ale řadí se na stejnou úroveň. Já jsem si zvolil jako hlavní prostředek pro tvorbu rozhraní React právě proto, že se jedná o knihovnu. Framework se vyznačuje tím, že vynucuje určité problémy řešit jistým způsobem bez možnosti volby. Má to své výhody a nevýhody a do větších týmů bych rozhodně volil raději framework. Tento projekt ale budu vytvářet primárně sám a mám velice rád flexibilitu a možnost volby. V začátcích to bývá časově náročnější, ale vidím v tom obrovskou možnost osobního růstu, protože při každé volbě musím hodnotit výhody/nevýhody a nakonec retrospektivně vidím následky svých rozhodnutí. Mimo to za vývojem

Reactu stojí Facebook a je používán největšími technologickými spočnostmi světa (Yahoo!, Nextflix a Airbnb [38]), takže je jistá jeho dlouhodobá podpora a od roku 2013, kdy byla vydána první verze, je dobře odladěný a ověřený.

React Je deklarativní, efektivní a flexibilní knihovna pro tvorbu rozhraní. Kód dělí do malých izolovaných částí kódu nazvaných „komponenty“, které se skládají do sebe a mohou tvořit komplexní uživatelská rozhraní. Pro vysoký výkon využívá techniku virtuálního DOM - nejprve si vytvoří virtuální strom podoby rozhraní v paměti, který následně porovná s aktuální podobou vykreslenou v prohlížeči a zmanipuluje pouze ty části, které se od posledního vykreslení změnily. Díky tomu je velice efektivní a dokáže vykreslovat komplexní stránky s obrovským množstvím dat. [39]

Kapitola 3

Návrh a realizace

Tato kapitola podrobně popisuje návrh celého řešení a jeho realizaci. Nejprve se věnuje popisu architektury serverové části včetně implementovaným bezpečnostním mechanismům. Dále definicí MQTT schématu, který platforma používá pro komunikaci se zařízeními, následované popisem implementace uživatelského rozhraní. Závěr kapitoly je věnován zamyšlení nad automatizací, nasazení celého řešení a tvorbě knihovny pro koncová zařízení a její následné demostraci při výrobě ukázkového zařízení.

3.1 Serverová část

Tato sekce popisuje návrh a realizaci serverového řešení včetně architektury, implementovaných bezpečnostních mechanizmů a definici schématu pro komunikaci s platformou na protokolu MQTT.

3.1.1 Architektura

Systém implementuje relaxovanou **Třívrstvou architekturu** obohacenou o vzor **Publish subscribe**. Základní myšlenkou Třívrstvé architektury je oddělení zodpovědnosti do tří vrstev, kde každá vrstva by měla být zodpovědná za určitou činnost a nepřesahovat svým rozsahem do zodpovědnosti jiné. Jednotlivé vrstvy:

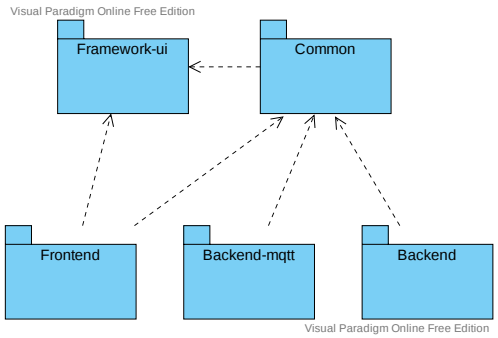
- Controller (řadič) - prostředník pro komunikaci mezi uživatelským rozhraním a služeb, které systém nabízí.
- Service (služba) - část systému obsahující aplikační logiku, jejímž úkolem je vykonání určitých úkolů (např. vytvoření uživatele).
- Data access layer (vrstva pro přístup k datům) - vrstva zapouzdřující přístup do databáze, která vytváří dle požadavků příslušné dotazy a

výsledky mapuje na objekty, se kterými se pracuje lépe než s textovými záznamy.

Vzor **Publish subscribe** přidává navíc asynchronní zpracování událostí. Toto řešení je velmi užitečné v případě, pokud je potřeba navázat nějaké akce (nejčastěji volání třetí strany např. kvůli analytickým údajům). Například pokud vytvoříme řadič pro registraci uživatele, tak od něj očekáváme, že vytvoří uživatele a to je vše. Jsou ale situace kdy potřebujeme navázat další akce, které nemají žádnou přímou spojitost s danou odpovědností (např. zalogování do google Analytics nebo odeslání emailu). Přidáním této akce přímo do kontrolleru bychom porušili *Princip jedné odpovědnosti* (Single-responsibility principle), který říká, že každý objekt (v tomto případě řadič) by měl vykonávat pouze činnost, která se od něj očekává. V tuto chvíli se hodí vzor *Publish subscribe*, který umožní v řadiči vyslání (emit) události, že byl registrován uživatel a veškeré nesouvisející akce se vykonají v obsluze dané události v jiné části systému. Díky tomu nedojde k porušení Principu jedné odpovědnosti a kód řadiče dělá přesně to, co se od něj očekává a nic víc.

Serverová část je rozdělena na dva separátní procesy kvůli zvýšení odolnosti, aby v případě pádu jedné z části buď fungovalo uživatelské rozhraní nebo část obsluhující komunikaci se zařízeními.

- **Backend** - dává k dispozici RESTful rozhraní pro kompletní ovládání platformy, dále vykonává naplánová akce jako odesílání emailů.
- **Backend-mqtt** - interaguje se zařízeními přes MQTT broker a odesílá real-time změny na frontend.



Obrázek 3.1: Diagram rozdělení balíčků

3.1.2 Databáze

Databáze je kritická část každého systému, protože se stará o perzistentní uchování dat, bez kterého bychom ztratili veškerá data při restartu či výpadku elektrické energie. Databáze lze obecně rozdělit do dvou skupin. První je

označována jako SQL, která je založená na předpokladu, že objekty (reálného či virtuálního světa) lze přesně definovat a zpravidla mají mezi sebou vztah (relaci) např. potraviny a nákupní košík - vložení potravin do košíku lze reprezentovat vztahem, který je mezi danou potravinou a košíkem (je vložena / není). SQL databáze se orientují právě na zachycení vztahů mezi strukturovanými daty. Naopak druhá skupina NoSQL (Not only SQL) se orientuje na ukládání nerelačních nestrukturovaných dat. Problematika různých typů databází je velice rozsáhlá a více se jí zde věnovat nebudeme. Pro podrobnější popis doporučuji [40].

Oba přístupy mají své výhody a nevýhody. Je potřeba vždy volit databázi na základě druhu dat, která budou uchovávat. Pro mé řešení jsem zvolil NoSQL databázi z následujících důvodů. Vzájemná provázanost dat bude naprosto minimální viz. sekce s doménovým modelem 2.2. Dále zařízení mohou produkovat potencionálně obrovské množství dat, pro což se NoSQL databáze hodí lépe, protože umožnují tzv. horizontální škálování (rozložení zátěže na více fyzických strojů), které pro SQL databáze lze efektivně využít pouze pro čtení nikoliv zápis. NoSQL databáze také umožňuje mnohem dynamičtější vývoj, protože SQL databáze vyžadují pevnou strukturu dat a i malá změna struktury znamená často velmi složitou migraci všech dat, zatímco NoSQL databáze umožňují ukládání dat bez nutnosti definovat jejich strukturu.

Specificky byla zvolena NoSQL databáze MongoDB, která je velmi populární mezi vývojáři, má skvělou podporu ze strany knihoven v NodeJS světě a pro uchovávání dat používá formát podobný formátu JSON (MongoDB nazývá formát BSON [41]), což se velmi snadno kombinuje s jazykem JavaScript, který JSON používá pro nativní objekty.

Mongoose je knihovna pro NodeJS, která vytváří nad MongoDB objektovou abstrakci a spoustu dalších užitečných funkcí jako validace a type cast [42]. Základním prvkem je definování schéma pro jednotlivé dokumenty, což může vypadat jako návrat do striktního schématu u SQL databází, ale zde je schéma definované pouze na úrovni Mongoose, tedy mnohem flexibilnější a méně restriktivní. MongoDB nabízí oficiální knihovnu pro přístup do databáze, ale preferuji Mongoose, protože díky definici schémat mám obecně větší kontrolu nad daty, která se dostanou do databáze a mají výbornou rozsáhlou dokumentaci.

3.1.3 Souborová struktura

Oba procesy jsou rozděleny do separátních balíčků *backend* a *backend-mqtt*, které na sobě nejsou nijak závislé a pro sdílení společných částí kódu, primárně databázových modelů, je využíván balíček **common**. Následující souborová struktura pro oba procesy byla inspirována článkem *Bulletproof node.js project architecture* [43], která se mi velmi osvědčila v jiných projektech.

`packages`

- backend — obsluha RESTful rozhraní a úkolů
  - src
    - api — zdrojové kódy pro jednotlivé endpointy
      - device
      - discovery
      - user
    - jobs — definice úkolů pro AgendaJS
    - loaders — rozdělený startovací proces do modulů
    - middleware — definice vlastních middleware
    - services — zdrojové kódy služeb
    - subscribers — obsluha akcí na asynchroní události
- backend-mqtt — oblsuha MQTT a Socket.IO
  - src
    - api
      - auth — endpoint pro RabbitMQ autentifikaci
      - actions — endpoint pro interní komunikaci s Backend
    - services — zdrojové kódy služeb
      - firebase — odesílání notifikací
      - mqtt — napojení na MQTT broker a obsluhu zařízení
      - websocket — real-time komunikace pomocí Socket.IO
    - subscribers — definice akcí na asynchroní události
- common — obsahuje sdílené části kódu
  - src
    - config — konfigurace načtená z promněných prostředí
    - models — definice mongoose schémat a typů
    - utils

3.1.4 Proces Backend

Tento proces s využitím frameworku ExpressJS implementuje webový server nabízející RESTful rozhraní pro vytváření, editaci a mazání všech entit platformy. Data se odesílají a konzumují ve formátu JSON. Proces dále implementuje odesílání emailů.

3.1.4.1 Popis rozhraní

Všechny endpointy, až na přihlášení a registraci uživatele, vyžadují autentizační token v hlavičce požadavku. Odpověď se potom liší podle oprávnění daného uživatele. Ukázka rozhraní:

- POST /user - vytvoří nového uživatele
- GET /device - vrací seznam zařízení
- DELETE /device/:deviceId - odstraní dané zařízení

- PATCH /device/:deviceId - aktualizuje zařízení
- GET /device/:deviceId/thing/:thingId/history?from=&to - vrací historická data pro specifikovanou věc, parametry specifikují časový rozsah
- PUT /device/:deviceId/thing/:thingId/notify - nastaví notifikační pravidla pro určitou věc
- PATCH /device/:deviceId/thing/:thingId - aktualizuje state pro danou věc

Server na požadavky odpovídá následujícími HTTP kódy (tělo odpovědi obsahuje podrobnější chybovou hlášku):

- 200 - v pořádku, součástí těla odpovědi jsou data
- 204 - v pořádku, tělo odpovědi je prázdné
- 400 - chybný požadavek
- 403 - nedostatečné oprávnění
- 404 - požadovaný zdroj nebyl nalezen
- 500 - chyba serveru

3.1.5 Proces Backend-mqtt

Tento proces implementuje autentizační rozhraní, které používá MQTT broker pro autentifikaci jednotlivých zařízení. Je přihlášen k odběru všech zpráv z MQTT brokeru, na které příslušně reaguje - vytváří nově objevená zařízení, ukládá stav zařízení, změny stavu jejich vlastností a odesílá real-time změny na rozhraní pomocí knihovny Socket.IO. Dále implementuje RESTful rozhraní pro interní komunikaci s procesem Backend, které umožňuje nastavení změnu stavu vlastnosti a inicializaci párování nového zařízení. Toto rozhraní je z důvodu oddělení odpovědností, kdy tento proces řeší obsluhu zařízení přes MQTT, zatímco proces Backend výhradně komunikaci s uživatelským rozhraním.

3.1.6 Bezpečnost

Platforma zpracovává uživatelská data a proto je nutné tato data chránit (s ohledem na soukromí uživatelů a i z pohledu zákona). Následující text pojednává o implementovaných bezpečnostních prvcích.

RESTful rozhraní je přímo dostupné z internetu a proto je velice důležité ho správně zabezpečit proti zneužití. Proti velkému množství útoků se lze bránit správným nastavením HTTP hlaviček. Toto sice nechrání před přímými

útoky, protože útočník může hlavičky ignorovat, ale chrání uživatele tak, že informují jejich prohlížeč o povoleném chování stránky např. odkud je bezpečné stahovat kód. Tímto způsobem lze primárně předejít útokům typu *Cross-site scripting* (vložení/podstrčení cizího JavaScript kódu do stránky) a jiným druhům *Cross-site infections* (vkládání html, stylů a jiných objektů) a *Clickjacking* (překrývání klikacích prvků jinými s úmyslem propagace kliknutí na prvek, který chce útočník). Pro nastavení hlaviček jsem využil middleware „helmet“, který je doporučený projektem OWASP (Open Web Application Project) [44], který se přímo zabývá bezpečností webových aplikací. Pro ochranu přihlášení před útokem typu *Brute-force* (hrubou silou) jsem použil middleware „express-rate-limit“ s omezením množství požadavků na jednu ip adresu za časový úsek. V případě potřeby je ale velmi snadno rozšiřitelný o komplexnější omezení (např. limit pokusů pro kombinaci uživatelského jména a ip adresy).

Pravděpodobně nejznámější typ útoku je *SQL Injection*. Díky zvolení databáze typu NoSQL systém touto zranitelností netrpí přímo, ale jejím ekvivalentem v podobě *NoSQL Injection*. Princip tohoto útoku je velmi jednoduchý. Zranitelnost vychází ze všech uživatelských vstupů, která se používají pro vytváření dotazů do databáze. Při sestavování těchto dotazů lze chytře využít možnost daného dialektu databázového jazyka a kompletně modifikovat jeho význam. Způsob ochrany spočívá v odstranění či nahrazení všech potencionálně zneužitelných znaků z uživatelstkého vstupu. Konkrétně jsem využil middleware „express-mongo-sanitize“, který odstraní ze všech potencionálně zneužitelných pozic v datech všechny znaky „$“ a „.“, které lze v případě MongoDB využít právě pro *NoSQL Injection*.

Do systému mohou přistupovat klienti pod různými uživatelskými účty, které mají různá oprávnění vázaná k určitým zařízením. Proto je nutné zamezit přístup pouze k takovým zdrojům serveru, ke kterým má daný přihlášený uživatel přístup. Jak vlastně server pozná, kdo inicioval daný požadavek? V případě nestavového protokolu HTTP, musí iniciátor přiložit ke každému požadavku token, kterým prokáže svoji totožnost. K tomuto účelu se nejčastěji využívaly dlouhé náhodné unikátní identifikátory, které byly uloženy v databázi u daného uživatele. Toto řešení ale vyžadovalo dotaz do databáze při každém požadavku pro ověření identity, což zbytečně zvyšuje zátěž na server. Proto byl vytvořen standard využívající asymetrickou kryptografii JWT (JSON Web Token, [45, RFC 7519]), který umožňuje aby po přihlášení server odeslal klientovi řetězec obsahující informace (např. id, jméno, příjmení, úrověň oprávnění) s cryptografickým podpisem a asymetrická kryptografie zaručuje, že nelze tyto informace modifikovat bez poškození integrity. Díky tomu odpadá nutnost pokaždé se dotazovat do databáze, protože stačí ověřit integritu tokenu. Tento postup má samozřejmě, ale i své negativní vlastnoti, kterými se zde však nebudeme zabývat. Pro detailnější vysvětlení doporučuji [46];

Použití HTTPS (HTTP spolu se šifrováním SSL nebo TLS) je dnes sa-

mozřejmností v případě, že se na stránce zadávají jakékoli údaje. NodeJS přímo podporuje šifrované HTTP, ale správa certifikátu i nastavení není úplně přímočaré. Proto jsem NodeJS použil jako HTTP server s reverzním proxy serverem Nginx, který s klienty již komunikuje pomocí zabezpečeného spojení. Nginx je široce podporován různými nástroji pro správu webů mimo jiné nástrojem „Certbot“, který umožňuje automatické získání bezplatného certifikátu nutného pro provoz HTTPS. Lze velmi snadno konfigurovat a nabízí pokročilé funkce jako *load balancing*.

Veškerá komunikace mezi koncovými zařízeními a platformou probíhá přes protokol MQTT a proto zabezpečení tohoto kanálu je nezbytné. Není žádoucí, aby třetí strana mohla posílat požadavky pro změnu stavu zařízením, i odposlouchávání zpráv je bezpečnostní riziko (z odposlechu pohybových čidel lze zjistit, zda je někdo doma, zlatý důl pro zloděje). Šifrování MQTT dokáže zajistit využitím protokolu nižší vrstvy TLS (označováno jako *mqtts* nebo *mqtt over tls*). Toto řešení zamezí odposlechnutí komunikace mezi Brokerem a zařízením. Většina existujících řešení pro domácnost další bezpečností prvky neimplementuje a následkem toho sice nelze odposlechnout komunikace, ale lze se jednoduše přihlásit k Brokeru k odběru všech zpráv. Já však považuji toto řešení jako nedostatečné a proto implementuji systém pro autentizaci (ověření identity) i autorizaci (kontrola oprávnění pro přístup k danému zdroji). MQTT specifikace umožňuje přihlášení pomocí uživ. jména a hesla nebo certifikátu. Vzhledem k omezenému výpočetnímu výkonu ESP8266 jsem nucen využít kombinaci jména a hesla, protože rozumný výpočet asymetrických šifer je za hranicí jeho možností. Jako MQTT Broker využívám RabbitMQ, který podporuje definici vlastního backendu pro kontrolu oprávnění přes RESTful rozhraní. Toto řešení mi umožňuje kontrolovat přihlášení jednotlivých zařízení a i následně jednotlivé požadavky k publikaci a odběru zpráv. Součástí dat, které předává RabbitMQ autentizačnímu serveru je i název tématu do kterého chce zařízení zapisovat nebo z něho číst. Mohu tak přesně specifikovat, jestli danému zařízení bude umožněn přístup do daného tématu či nikoliv.

3.1.7 Validace

Webové aplikace se stávají stále více komplexní se složitější datovou strukturou. Z pohledu systému je důležité validovat veškerá data, která přijdou od třetí strany před tím, než s nimi začneme pracovat, protože se nelze pouze spoléhat, že nám je někdo poslal ve správném formátu, v horším případě útočník bude záměrně posílat chybná data s nějakým postranním úmyslem. Dále z pohledu uživatele, je pro něj důležité, aby formulář byl intuitivní a na případné chybně zadané hodnoty byl ihned upozorněn - z vlastní zkušenosti vím, že není nic horšího než vyplnit dlouhý formulář, který se zdá naprosto validní, stisknout tlačítko odeslat a následně vidět nic neříkající hlášku „Nevalidní formulář“ bez jakýchkoliv upřesňujících informací.

Využití stejného programovaícho jazyka jak na backendu, tak frontendu mi

umožňuje využít stejný způsob validace dat na obou stranách, který poskytne uživateli okamžitou odezvu bez nutnosti duplikace validační logiky. Implementoval jsem si proto vlastní framework, který je založen na vzoru „Field and Descriptor Field“ [47]. Podle tohoto vzoru vstupují do systému dva separátní vstupy. Field obsahuje reálnou hodnotu pole a deskriptor popisuje správný formát hodnoty pole. Systém při validaci určité hodnoty si nejprve načte příslušný deskriptor a následně aplikuje validační logiku dle konfigurace v deskriptoru na hodnotu.

Pomocí deskriptorů se definují struktury dat celých formulářů. Většina RESTful endpointů konzumuje v těle požadavku formulář, který je předem systémem zvalidován (pomocí middlewaru). Výstupem validace je seznam všech polí, která nejsou validní a chybové hlášky obsahující přesný popis, proč validace selhala.

3.1.7.1 Field deskriptor

Deskriptor se definuje pro každé formulářové pole, společně tvořící deskriptor pro celý jeden formulář. Datovou strukturu reprezentující formulář lze libovolně zanořovat a struktura odpovídá 1:1 struktuře držící samotná formulářová data. Ukázka deskriptoru pro formulář obsahující jedno pole:

```
{
    FORM_NAME: {
        userName: {
            // sebereflektivní cesta k deskriptoru
            deepPath: "FORM_NAME.userName",
            label: "Název který se zobrazí u pole v UI",
            // pokud vrátí false, tak required bude ingorováno
            when: (formData) => formData.selected === "user",
            // povinost vyplnění pole
            required: true/false,
            // seznam validací, kterými se má validovat hodnota
            validations: [
                validationFactory('isString', {min: 3, max: 10})
            ],
        }
    }
}
```

3.1.8 MQTT schéma

MQTT je zvolený komunikační protokol umožňující odesílání a přijímání asynchronních zpráv identifikovaných tématem (popsáno v 2.4.5.1). Jedná se tedy o definici obecné komunikace a pro účely této platformy je potřeba vytvořit specifikaci, podle které se budou jednotlivá zařízení řídit při odesílání zpráv.

Tímto bude jasně zadefinované, jak platforma bude reagovat na jednotlivé zprávy a půjde implementovat bezpečnostní mechanizmy.

První prototyp jsem založil na vlastní struktuře témat na MQTT protokolu, na kterém zařízení oznamovala v jakém stavu se nachází a naslouchala pro případné požadavky na změnu. Uživatel při vytváření zařízení musel nadefinovat veškeré jeho vlastnosti pomocí poměrně rozsáhlých formulářů, následně byl vygenerován api klíč, který bylo nutné zadat do zařízení pro jeho úspěšné přihlášení k platformě. Toto řešení se ukázalo jako nešťastné, protože uživatel pro přidání zařízení musel mít rozsáhlé znalosti dané problematiky a tento proces byl velice časově náročný. Proto jsem se rozhodl, že místo aby uživatel definoval ručně každé zařízení, zavedu automatickou detekci (auto discovery) nových zařízení, které budou sama propagovat platformě jaké věci a vlastnosti podporují.

Po mnohých experimentech jsem nakonec své řešení založil na konvenci **Homie** [48] specifikující strukturu MQTT témat pro automatickou detekci, konfiguraci a používání zařízení. Tento základ jsem obohatil mimo jiné o výměnu párovacího klíče a z důvodu, že konvence počítá s globálním unikátním identifikátorem pro každé zařízení, tak jsem přidal navíc unikátní prefix pro MQTT téma, které označuji jako „realm“ - tento prefix je unikátní pro každého uživatele a všechny jeho zařízení komunikují v tomto realmu. Toto mi umožňilo přesunout restrikci identifikátoru z globální na úroveň uživatele a zároveň dává každému uživateli vlastní prefix témat pro jeho zařízení či jiné využití a pro monitoring komunikace mu stačí se přihlásit k odběru všech zpráv z daného tématu. Moje řešení neimplementuje kompletní konvenci, ale pouze následující část.

Základem *Homie* konvence je, že každé zařízení obsahuje uzly (věci) a ty mají vlastnosti. Příklad - zařízení auto má věc motor, které má vlastnosti teplotu a tlak. Každé zařízení komunikuje v tématu obsahující nějaký prefix a v další úrovní tématu id daného zařízení (ukázka „homie/esp-1919/“). Homie specifikuje základní atributy tématu, které se používají pro konfiguraci a začínají symbolem „$“:

- $name - člověkem čitelný název
- $state - v jakém stavu se dané zařízení nachází
  - Init - zařízení je připojené, ale ještě není plně připraveno
  - Ready - je připraveno a plně funkční
  - Sleeping - zařízení je uspané a momentálně nekomunikuje
  - Alert - zařízení vyžaduje pozornost
  - Disconnected - odpojeno
  - Lost - ztráta spojení (zařízení je poviné toto zadefinovat jako Last Will Testament)

- $nodes - seznam id věcí, které zařízení obsahuje oddělené čárkou

Pro prefix tématu „homie/deviceId/nodeId“ lze odeslat následující atributy:

- $name - člověkem čitelný název věci
- $type - typ věci
- $properties - seznam id vlastností, které daná věc obsahuje oddělené čárkou

Pro prefix tématu „homie/deviceId/nodeId/propertyId“ lze odeslat atributy:

- $name - člověkem čitelný název vlastnosti
- $datatype - datový typ hodnoty vlastnosti (string, float, integer, boolean, enum)
- $unit - jednotka hodnoty vlastnosti [volitelný]
- $format - specifikace omezení pro daný datový typ. Pro float, integer lze uvést rozsah hodnot ve formátu minimální a maximální hodnota oddělená dvojtečkou (např. „10:20“). Pro enum se specifikuje výčet možných hodnot oddělených čárkou (např. „red,blue,orange“) [volitelný]
- $settable - informace zda lze hodnotu vlastnosti nastavit (true, false) [volitelný, výchozí false]

3.1.8.1 Upravená specifikace

Rozdíl v mém řešení oproti „Homie“ konvenci primárně spočívá v použití dvou rozdílným prefixů témat v závislosti, jestli zařízení již bylo spárováno (uživatel si dané zařízení přidal) či nikoliv. Toto opatření je z důvodu vyšší bezpečnosti, abych mohl oddělit již spárovaná zařízení od ostatních. Při prvním připojení se zařízení ohlásí v prefixu „prefix/“ následovaný identifikátorem zařízení stejně jako v případě *homie* konvence. Oznámí svůj status, název a všechny své schopnosti (věci a vlastnosti). Toto téma je veřejně přístupné pro všechny zařízení, která se k MQTT brokeru přihlásí uživatelským jménem shodujícím se s identifikátorem zařízení na rozdíl od druhého prefixu „v2/realm“, který je přístupný pouze zařízení, která se přihlásí pomocí platného api klíče s přístupem do daného realmu. Přidané atributy:

- $realm - do kterého realmu zařízení chce patřit (uživ. jméno uživatele)
- $config/apiKey/set - tento atribut využívá platforma pro odeslání api klíče. Zařízení je zodpovědně za přihlášení odběru daného tématu.

- $cmd/set - využívá platforma pro odeslání příkazů, jmenovitě restart a reset.

Restrikce na typ věci:

- sensor - senzor vyžadující mít první vlastnost číselného typu
- switch - přepínač vyžadující mít první vlastnost datového typu boolean
- activator - tlačítko vyžadující mít první vlastnost datového typu enum s jednou hodnotou
- generic - obecný typ

Přidán atribut pro téma definující vlastnosti:

- $class - do jaké třídy hodnota vlastnosti patří, možnosti: humidity, temperature, voltage, pressure. Tato hodnota ovlivňuje pouze uživatelské rozhraní, specificky jaká ikonka se u hodnoty zobrazí. [volitelné]

Následuje ukázka komunikace automatické detekce zařízení umožňující zapnutí/vypnutí světla, měření teploty s identifikátorem „light“ a hlásící se k uživateli s uživ. jménem „pepa“:

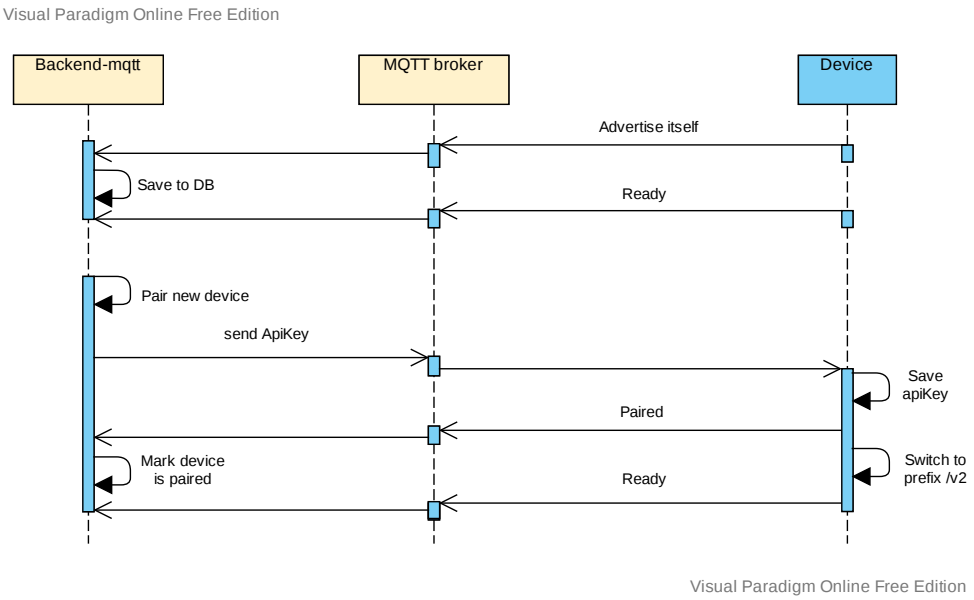
```
prefix/light/$status   -> init
prefix/light/$name   -> Světlo
prefix/light/$nodes  -> sensor,switch
prefix/light/$realm  -> pepa
prefix/light/sensor/$name    -> Senzor
prefix/light/sensor/$type    -> sensor
prefix/light/sensor/$properties -> temperature
prefix/light/sensor/temperature/$name -> Teplota
prefix/light/sensor/temperature/$datatype -> float
prefix/light/sensor/temperature/$unit -> °C
prefix/light/sensor/temperature/$class -> temperature
prefix/light/switch/$name   -> Lustr
prefix/light/switch/$type   -> switch
prefix/light/sensor/$properties -> power
prefix/light/sensor/power/$name -> Lustr
prefix/light/sensor/power/$datatype -> boolean
prefix/light/$status    -> ready
```

Pokud se detekované zařízení nachází ve stavu „ready“, tak je zobrazeno příslušnému uživateli pro přidání. Pokud si ho uživatel přidá, tak platforma odešle api klíč danému zařízení, který jeho přijetí potvrdí, klíč si uloží a následně se přepne do prefixu témat „v2/realm/“. Pokračování ukázky:

```
prefix/light/$apiKey/set
    -> XXXXXXXXXXXXXXXX (odesláno platformou)

prefix/light/$status    -> paired
v2/pepa/light/$status   -> ready

v2/pepa/light/sensor/temperature   -> 20.05
v2/pepa/light/switch/power         -> true
```



Obrázek 3.2: Diagram spárování zařízení

3.2 Uživatelské rozhraní

Tato sekce se zabývá architekturou a implementací uživatelského rozhraní formou webové stránky. Implementace je rozdělena do dvou balíčků:

- Framework-ui - obsahuje komplexní řešení pro validace formulářů včetně integrace s Redux (napojení na state, definice akcí a reducerů), implementuje znovu použitelné React komponenty, včetně komponenty řešící přímé napojení formulářového pole na state a obsluhu jeho validací (FieldConnector). Dále jazykovou lokalizaci pro systémové hlášky a nádstavbu nad základní rozhraní pro odesílání HTTP požadavků, která řeší zobrazení chybových hlášek a definuje flexibilnější rozhraní.

- Frontend - implementace samotné aplikace

3.2.1 Frontend

Uživatelské rozhraní je realizováno jako SPA - průchod celou aplikací je plynulý a nikdy nedochází k přenačítání celé stránky, ale pouze k překreslení potřebných částí. Toto řešení zlepšuje uživatelský zážitek, protože stránka zůstává pořád plně aktivní a při čekání na vyřízení požadavku uživatel může pokračovat v interakci. Také je zde implementován standard PWA - v podporovaných systémech jako je např. android lze aplikaci tzv. „Přidat na plochu“, potom při otevření vypadá jako nativní aplikace (nemá zobrazený url bar). Statické soubory jsou v cache, díky čemuž je minimalizován datový přenost a stav aplikace je perzistentně ukládán, takže při zavření aplikace a následném otevření je stav plně obnoven a uživatel pokračuje přesně tam, kde skončil naposledy a to i v případě bez přístupu k internetu - aplikace je kompletně načtena, ale následná interakce je již závislá na komunikaci se serverem.

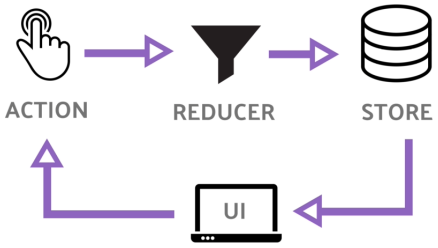
3.2.1.1 State management

React je postavený na předávání dat mezi jednotlivými komponenty, které tvoří stromovou strukturu, data lze však předávat primárně z vrchu dolů a proto je poměrně složité ze spodní komponenty předat změnu do vrchní. Navíc dnešní aplikace pracují s obrovským množstvím dat, ke kterým potřebují přistupovat různé části aplikace a ideálně transparentní cestou data modifikovat a o této modifikaci se musí dozvědět všechny komponenty pracující s danými daty, aby dostali jejich aktuální verzi. Dnes již React obsahuje obecné řešení pro předávání dat, ale já jsem se rozhodl pro využití komplexního řešení s centrálním úložištěm stavu aplikace v podobě knihovny Redux, které nabízí více funkcí a je odzkoušené časem. Redux je založený na „Observer pattern“. Tento návrhový vzor definuje dvě entity - předplatitel (pozorovatel) a vydavatel (pozorovaný). V praxi se jednotlivé komponenty přihlásí k odběru určitých dat („předplatí si je“) a v případě, že někdo chce data modifikovat, tak se stane vydavatelem a řekne, co modifikuje a jak vypadají nová data. Redux zajistí, že všichni předplatitelé dat, kterých se modifikace týká, dostanou jejich nejnovější verzi. V Reduxu se používá následující terminologie:

- store - centrální úložiště dat (tvz. „jediný zdroj pravdy“)
- akce - popis k jaké změně dojde
- dispatch akce - odeslání určité akce k vykonání
- reducer - reaguje na odeslanou akci, obsahuje logiky pro modifikaci dat

3.2.1.2 Vzhled

V kombinaci s Reactem jsem použil knihovnu Material-ui, která obsahuje velké množství nastylovaných komponent a komplexní řešení pro stylování React



Obrázek 3.3: Redux tok dat [2]

komponent. Tuto knihovnu používám již od prvopočátku jejího vzniku a velmi jsem si ji oblíbil kvůli detailní dokumentaci. Obsahuje vestavěné řešení pro **CSS-in-JS**, které mi jako programátorovi velmi vyhovuje, protože překlenuje spoustu limitací přímého použití CSS. Umožňuje definovat vzhled ve stejném souboru i jazyce jako samotnou React komponentu, díky tomu nemusí programátor přepínat kontext mezi různými jazyky a může se plně soustředit na vývoj uživatelského rozhraní.

3.2.2 Souborová struktura

```
packages
├── framework-ui
│   └── src
│       ├── api ..... implementace rozhraní pro odesílání požadavků
│       ├── Components ..... React komponenty
│       ├── localization ..... řešení pro lokalizaci systémových hlášek
│       ├── privileges ... pomocné funkce pro oprávnění a jejich dědičnost
│       ├── redux ..... implementace akci a reducerů, primárně pro správu
│       │                formulářových dat
│       └── validations ..... implementace validací
└── frontend
    ├── public
    └── src
        ├── Pages ..... jednotlivé stránky rozhraní
        ├── api ..... definice RESTful volání
        ├── components ..... sdílené komponenty napříč stránkami
        ├── containers ..... React kontejnery obalující celou aplikaci
        ├── firebase ..... služba pro registraci a obsluhu notifikací
        ├── store ..... inicializace Redux, definice akcí a reducerů
        └── webSocket ..... služba pro inicializaci Socket.IO
```

3.2.3 Validace

Obecný princip fungování validací je popsán v sekci Backend. Na Frontendu se validace používají pro upozornění uživatele na případně chybně zadanou hodnotu. Samotná formulářová data jsou spravována knihovnou redux, stejně jako celý state aplikace. Jejich struktura je shodná s deskriptory formuláře - deepPath se využívá pro získání příslušné hodnoty.

Pro vykreslení formulářového pole jsem připravil komponentu *FielConnector*, které se předá typ pole (text, email, select atd.) a deepPath, podle které si nalezne příslušný deskriptor, ze kterého získá potřebné atributy a validace, které se provolávají v následujících případech:

- uživatel poprvé zadává hodnotu a vyklikne (událost „onBlur“);
- uživatel edituje již zadanou hodnotu, potom se validace spouští po každé změně (událost „onChange“).

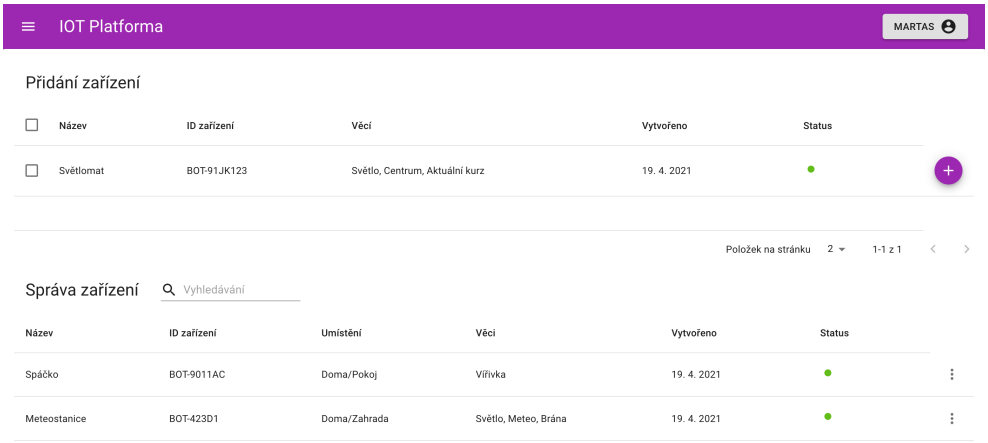
3.2.4 Rozhraní

Aplikace nabízí jednoduché uživatelské rozhraní s jedním menu pro navigaci mezi jednotlivými stránkami a tlačítko pro přihlášení, které otevře přihlašovací dialog.

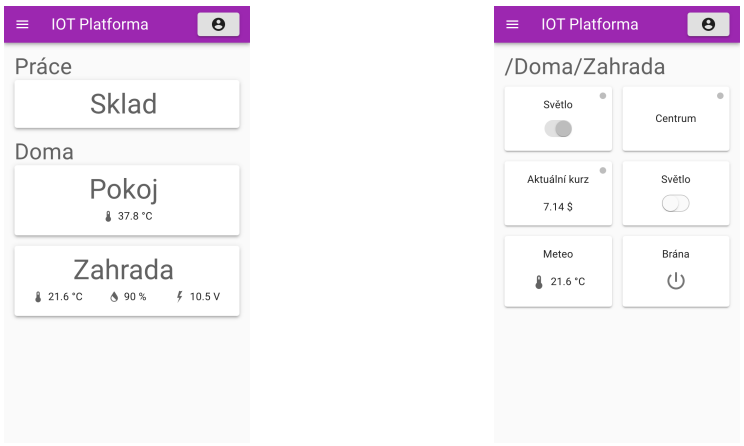
Nepřihlášenému uživateli je k dispozici stránka pro registraci obsahující formulář. Po registraci je automaticky přihlášen a má k dispozici stránku pro správu zařízení a stránku pro ovládání a sledování jednotlivých věcí. Správa zařízení je rozdělena na dvě sekce, první obsahuje tabulku s detekovanými zařízeními, které lze přidat na dvě kliknutí. Druhá část obsahuje tabulku se zařízeními, ke kterým má uživatel oprávnění pro čtení a v případě oprávnění pro zápis, může editovat jejich informace, měnit oprávnění pro jednotlivé uživatele, odeslat příkaz pro restart nebo zařízení odstranit.

Stránka pro ovládání zobrazuje widgety pro jednotlivé místnosti seskupené podle budov. Pokud místnost obsahuje nějakou věc typu senzor, tak aktuální hodnota je zobrazena na tomto widgetu. Po rozkliknutí místnosti jsou rozbrazeny všechny věci, která daná místnost obsahuje. S některými věcmi lze přímo interagovat (přepínač a aktivátor) a u senzoru je zobrazena aktuální hodnota. Pokud se dané zařízení nenachází v připraveném stavu, tak je uživatel upozorněn barevným kolečkem. Po kliknutí na název věci je zobrazeno dialogové okno, které se částečně liší v závislosti na typu věci:

- **Generic** zobrazí aktuální hodnoty vlastností a je umožněna případná interakce s nimi.
- **Sensor** zobrazí stejný obsah jako pro *generic* a navíc na začátku vykreslený graf vizualizující průběh hodnoty v čase za posledních 24 hodin.
- **Switch** zobrazí stejný obsah jako pro *generic* a navíc časovou osu vizualizující aktivitu vlastnosti.



Obrázek 3.4: Ukázka rozhraní - správa zařízení



(a) Zobrazení místností (b) Zobrazení místnosti

Obrázek 3.5: Ukázka rozhraní - výpis místností a jednotlivých věcí v místnosti

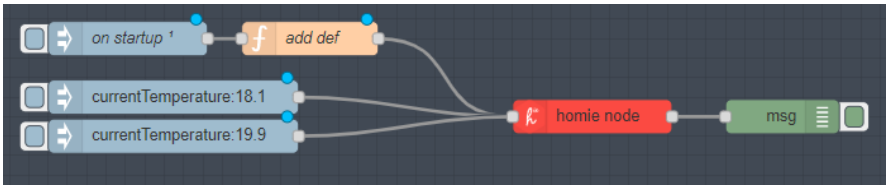
Přihlášený uživatel má dále k dispozici menu s možností editace vlastního uživatelskéhoo účtu a odhlášení.

Pokud má uživatel administrátorské oprávnění, tak má navíc k dispozici stránku pro správu uživatelů, ve které je tabulka zobrazující všechny uživatele s možností jejich editace a odstranění.

3.3 Automatizace a hlasové ovládání

Implementované řešení umožňuje automatizaci ze strany zařízení, která se mohou přihlásit k odběru zpráv z ostatních zařízení a příslušně reagovat.

Možnost flexibilní automatizace ze strany platformy vyžaduje nasazení velmi komplexního řešení, které z důvodu náročnosti nebylo implementováno. Nabízí se však využití nástroje **Node-Red**, který umožňuje vytvářet rozsáhlé datové toky pomocí vizuálního programování [49]. Obsahuje intuitivní grafický nástroj pro vytváření složitých podmínek a následných akcí.



Obrázek 3.6: Node-Red ukázka definice akce [3]

Momentální řešení uživatelům umožňuje napojení vlastního nástroje, který bude přihlášen k odběru MQTT zpráv z jejich zařízení a mohou si tedy vytvářet datové toky v něm. Pro integrování řešení přímo do platformy by bylo nutné analyzovat možnost použití Node-Red v tomto více uživatelském prostředí. Současně s tímto řešením by šlo podmiňovat notifikace složitějšími výrazy než pouze v závislosti na aktuální hodnotě dané vlastnosti, ale i v závislosti na jiných složitějších podmínkách.

Pro využití hlasového ovládání pomocí současných asistentek jako *Siri* nebo *Google Assistent* by bylo potřeba doimplementovat možnost vytváření tokenů pro přístup k API rozhraní pro jednotlivé uživatele a jejich správu. Následně by bylo možné využít v asistentkách možnosti definovat akci na hlasový povel, která by odeslala zadefinovaný požadavek na změnu jehož součástí by byl přístupový token pro autorizaci. Takto by bylo možné realizovat jednoduché povely pro změnu jedné vlastnosti. Pro definici složitější akce na hlasový povel by bylo možné využít webovou službu **IFTTT** [50] v kombinaci s asistentkou, která umožňuje vytvářet podmíněné příkazy a obsahuje integraci s velkým množství služeb třetích stran.

3.4 Nasazení

Celé řešení jsem nasadil na Linuxový server s operačním systémem Debian. Pro zjednodušení jsem se rozhodl celý proces automatizovat. Využil jsem Open-Source automatizační nástroj **Jenkins** [51], který umožňuje psaní skriptů, které automaticky spouští při změně kódu (pro jeho verzování jsem využil nástroj GIT). Toto řešení mi umožňilo plně automatizovat sestavení celé aplikace a její následné nasazení na server pouhým uložením nové verze kódu. V případě výskytu chyby při sestavování jsem na ni byl ihned upozorněn a mohl ji opravit. Důvod využití právě nástroje Jenkins oproti alternativám např. *GitLab CI/CD* či *CruiseControl* spočívá primárně v rozsáhlému systému plu-

ginů, kterých má obrovské množtví na rozdíl od mladších konkurentů a lze provozovat zcela zdarma na vlastní infrastruktuře.

Pro spuštění platformy na systému Linux je připravena instalační příručka, která je umístěna na médiu (viz. příloha D). Pro případnou distribuci celého řešení uživatelům, by byla možnost využít nástroj **Docker**, který umožňuje spouštět virtualizované kontejnery. Takové řešení umožní spuštění celého serveru jedním příkazem. Automatizované sestavení s nástrojem Jenkins je již funkční a s minimální úpravou lze připravit obraz pro Docker, který by spustil celé řešení. Podrobnější popis problematiky kontejnerů naleznete v [52].

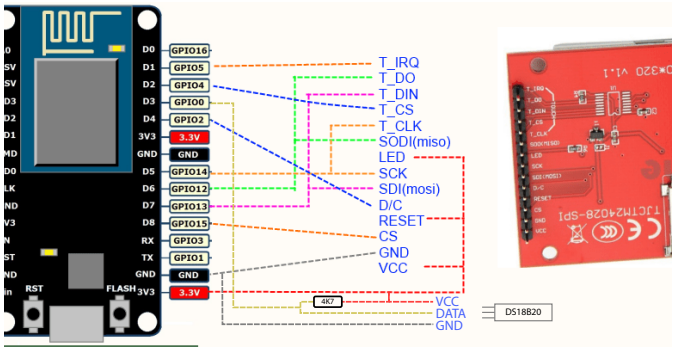
3.5 Chytrá udírna

Pro demonstraci vytvoření zařízení jsem si zvolil jedno, které jsem vytvořil pro svého otce a mělo velmi pozitivní přínos, jak po stránce úspory času, tak i chuťových buněk. Dal jsem si za cíl vyřešit vzdálené monitorování teploty udírny v průběhu uzení, protože u udíren se používá klasický ručičkový teploměr, který se musí každých 15 minut kontrolovat kvůli správnému udržení teploty v rozmezí přibližně deseti stupňů. Naše udírna je umístěna v rohu velké zahrady a v zimním období je velice nepříjemné neustále kontrolovat teplotu v celém průběhu uzení, které trvá i deset hodin.

Proto jsem se rozhodl vytvořit zařízení, které bude měřit teplotu, zobrazí ji na displeji a současně ji odešle do platformy pro možnost sledování z pohodlí domova. Tímto se celý proces uzení zpříjemní a s nastavenými notifikacemi nebude potřeba ani teplotu aktivně sledovat, protože platforma sama upozorní na problematickou teplotu.

3.5.1 Zapojení

Jako hlavní řídící desku s čipem ESP8266 jsem zvolil *Wemos D1 mini*, se kterou mám dobré osobní zkušenosti, ale vyhovovala by jakákoli jiná s dostatkem vyvedených pinů. Pro monitorování teploty použiji senzor DS18B20 (maximální teplota 110 °C pro uzení plně dostačuje), který využívá digitální komunikaci a umožňuje sdílení datového kabelu s více senzory, vhodné pro jednoduché přidání dalších senzorů v případě potřeby pro monitoring různých segmentů udírny. Pro zobrazení informací použiji displej „2.2”240x320 SPI TFT - ILI9341“, který je i dotykový (do budoucna počítám s možností jeho využití pro další funkce). Na zapojení může vypadat zvláštně zapojení rezistoru mezi datový a napájecí pin, ale toto je vyžadováno komunikační sběrnicí „1-Wire“ [53], kterou zvolený senzor využívá. Danou hodnotu rezistoru jsem převzal z datasheetu pro DS18B20 [54].



Obrázek 3.7: Zapojení řídící desky, displeje a senzoru teploty [4]

3.5.2 Výroba

Jednotlivé součástky byly zapojeny dle diagramu na obrázku 3.7. Pro zařízení jsem navrhl krabičku na míru a následně ji vytiskl na 3D tiskárně. Výsledné zařízení je napájené pomocí micro usb, tedy stačí běžná nabíječka na telefon a teplotní senzor se připojuje pomocí konektoru kvůli jednodušší manipulaci. Dále bylo potřeba vytvořit program, který poběží na zařízení pro odesílání teploty do platformy a bude zobrazovat informace (teplotu, čas) na displeji. Ukázka kódu 3.9 obsahuje využití vytvořené knihovny v sekci 3.6 pro napojení chytré udírny na platformu. Výsledný program obsahuje tento kód obohacený o komunikaci s displejem a načtení času z internetu. Model krabičky a finální zdrojový kód lze nalézt na přiloženém médiu viz. příloha D.

Obrázek 3.8: Finální verze výrobku

```
#include <IOTPlatforma.h>
#include <OneWire.h>
#include <DallasTemperature.h>

IOTPlatforma plat("Udírna");
OneWire oneWire(D3);
DallasTemperature sensors(&oneWire);
Property * propTemp;

void setup() {
    Node *node = plat.NewNode("sensor0", "Senzor",
                                    NodeType::SENSOR);
    propTemp = node->NewProperty("temperature", "Teplota",
                                    DataType::FLOAT);
    propTemp->setClass(PropertyClass::TEMPERATURE);
    propTemp->setUnit("°C");

    plat.start();
}

void loop() {
    plat.loop();

    sensors.requestTemperatures();
    float temp = sensors.getTempCByIndex(0);
    propTemp->setValue(String(temperature, 2).c_str());
    delay(1000);
}
```

Obrázek 3.9: Kód pro odesílání teploty

3.6 Knihovna pro ESP8266

Pro čip ESP8266 lze programovat s využitím oficiálního sdk (Espressif SDK) nebo prostředí Arduino, které se těší obrovské oblibě mezi kutily. Rozhodl jsem se využít prostředí Arduino, protože kolem něho existuje obrovská komunita a stovky již předpřipravených knihoven pro různorodé moduly. Díky tomu není potřeba tolik řešit nízkoúrovňové problémy jako např. implementaci protokolu pro komunikace se senzorem teploty, ale stačí si stáhnout příslušnou knihovnu a následně se plně soustředit na aplikační logiku.

Mým cílem je vytvoření platformy, ke které si bude moci kdokoliv připojit vlastní zařízení - abych proces připojení co možná nejvíce zjednodušil, vytvořil jsem knihovnu pro prostředí Arduino, která bude řešit veškerou komunikaci s

platformou a nabídne programátorovi přehledná rozhraní pro definici zařízení, jeho věcí, vlastností a reakcí na změny.

Knihovna implementuje následující funkce:

- Kaptivní portál - vytvoření wifi přístupového bodu, které po připojení např. telefonu zobrazí webovou stránku na které uživatel zadá přístupové údaje k místní wifi síťi, své uživatelské jméno a případně ip adresu instance platformy (pouze pokud provozuje vlastní).
- Připojení k MQTT brokeru - pro připojení je využita knihovna *pubsubclient*.
- Objevení zařízení - programátor deklarativním způsobem definuje věci a vlastnosti zařízení. Knihovna následně všechny tyto funkce ohlásí platformě.
- Spárování - po přidání zařízení uživatelem ve webovém rozhraní obdrží zařízení párovací klíč, který se perzistentně uloží.
- Definice reakcí - ke každé vlastnosti (v případě, že je nastavitelná) může programátor definovat funkci (callback), který se zavolá v případě, že došlo ke změně dané vlastnosti.
- OTA - možnost tvz. „aktualizace vzduchem“ místo nutnosti fyzického připojení k pc. Podporováno je nahrání firmwaru v rámci lokální sítě a zabezpečeno pouze heslem, kvůli malé paměti a nízkému výkonu pro využití ověření pomocí certifikátů.

Kapitola 4

Testování

Tato kapitola se zabývá testováním implemetovaného řešení nasazeného v produkčním prostředí. V první fázi je otestována stabilita řešení při velké zátěži a ve druhé intuitivnost uživatelského webového rozhraní. Produkční prostředí běží ve virtualizovaném serveru se systémem Debian (linux), přidělenými hardwarovými prostředky: CPU (5 vláken Ryzen 3600), 9 GB RAM, HDD (7200 otáček). Použitá verze NodeJS 12.22.1, MongoDB 4.2.13 a RabbitMQ 3.8.14.

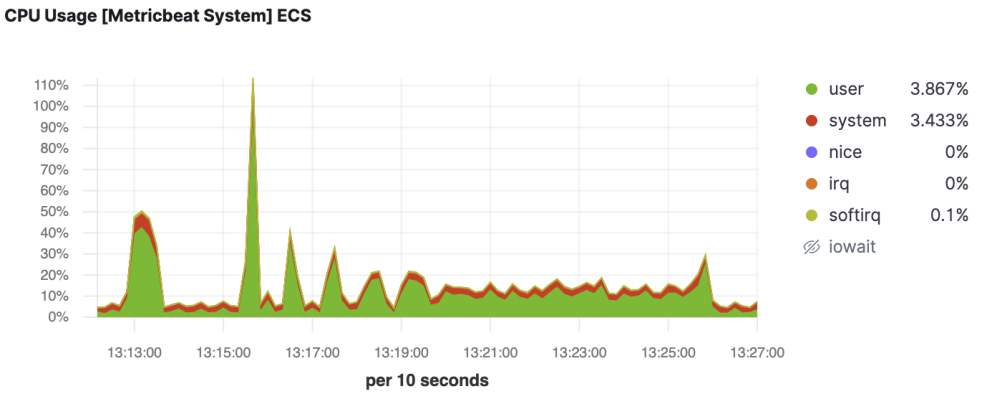
4.1 Zátěžové testování

Obsloužit desítky připojených zařízení není s dnešním výkonným hardwarem žádný problém. Pro ověření chování řešení pod opravdu velkou zátěží - desítky paralelních požadavků a stovky připojených zařízení - jsem vytvořil automatizovaný test, kterému se definuje počet uživatelů a počet zařízení, které mezi uživatele má rozprostřít a o vše ostatní se test postará sám (pomocí HTTP požadavků stejně jako by uživatel interagoval přes webové rozhraní).

Prvně je vytvořen příslušný počet uživatelů a všechna virtuální zařízení (komunikující přes MQTT protokol stejně jako reálná). Všem uživatelům jsou načtena objevená zařízení, která jsou následně spárována a začínají odesílat data ze senzorů a jiných vlastností simulujících reálný provoz. Data se odesílají v náhodném rozmezí 50 - 60 vteřin, aby se provoz rozložil v čase. Test probíhal v následující konfiguraci:

- počet uživatelů 25
- počet zařízení 250
- konfigurace zařízení - 2 senzory (teplota, vlhkost) a přepínač, data ze všech tří vlastností jsou odesílána paralelně
- počet paralelních API požadavků 10

Pro sledování zátěže databáze byl využit „Free monitoring“ [55], který sleduje dobu provádění operací, jejich počet a vytížení disku. Pro sledování zátěže cpu jednotlivých procesů byl použit linuxový nástroj *htop* a pro vykreslení grafu celkové zátěže cpu *Kibana*. Protože se jedná o více jádrový stroj, tak vytížení cpu nemá maximum 100 %, ale pro každé jádro 100 %, tedy celkem 500 %. Test byl proveden celkově 3x pro ověření správnosti naměřených dat. Tabulka dat, ze kterých byl sestaven graf na obrázku 4.1 je umístěna na přiloženém médiu viz. příloha D.



Obrázek 4.1: Zátěž CPU během testu

První zvýšení trvající minutu je způsobeno zařízeními ohlašujícími svoje vlastnosti, což způsobilo velký počet zápisů do databáze. Druhé zvýšení ve 13:15:40 trvající 20 vteřin způsobilo přidání jednotlivých zařízení uživatelům (a jejich spárování). Následujících 10 min trvající zatížení bylo způsobeno tím, že zařízení odesílají data ze svých vlastností. Závěrečná několika vteřinová zvýšená zátěž ve 13:25:50 je dána odesláním informace o odpojení ze všech zařízení. V průběhu testu byla průměrná doba vyřízení databázové operace 0,24 ms, zátěž cpu databází 8 %, procesem platformy 6 % a zátěž disku přibližně 2 %. V průběhu celého testu byla platforma aktivně využívána uživatelem přes webové rozhraní k ovládání fyzických zařízení a po celou dobu vše reagovalo ihned bez jakéhokoliv zpoždění, ani průměrná doba zpracování RESTful požadavku nebyla významně ovlivněna. Z výsledku testu tedy vyplývá, že platforma zvládne bez jakýchkoliv problémů obsloužit stovky zařízení bez negativního dopadu na výkon.

4.2 Uživatelské testování

Za pomoci uživatelského testování byla ověřena intuitivnost uživatelského prostředí. Cílem bylo zjistit, zda uživatelé zvládnou vykonat základní úkony bez pomoci a zjistit jejich náročnost.

Úkon	Uživatel 1	Uživatel 2
Připojte nové zařízení	4	3
Zapněte led světla	1	1
Podívejte se na průběh teplotu	2	2
Změňte umístění zařízení	1	1
Zařízení odstraňte	1	1

Obrázek 4.2: Hodnocení jednotlivých testerů

Každý uživatel dostal uživatelskou příručku (příloha C), přístup k již naprogramovanému zařízení, které ovládalo led světla a měřilo teplotu, a nakonec seznam úkonů, které mají vykonat a zpětně je ohodnotit na stupnici od jedné do pěti (pět je nejhorší). První tester byl muž, věk 49, se zájmem o techniku. Druhý byl také muž, věk 30, s ekonomickým vzděláním.

V průběhu testování se ukázal jako nejvíce problematický první krok, ve kterém uživatel zadává v kaptivním portále údaje k připojení na domácí Wifi a své uživatelské jméno k platformě. Zařízení občas ohlásilo chybu připojení k Wifi i při správně zadaných údajích. Ostatní části testu již probíhaly bez jakéhokoliv problému. Dle závěrečného hodnocení se rozhraní ukázalo jako uživatelsky velmi přívětivé a zařízení po úspěšném připojení k Wifi pracovalo již naprosto spolehlivě. Po ukončení testu byly uživatelé v nezávazném rozhovoru dotázáni na možná vylepšení. Z rozhovoru vyplynulo doporučení do budoucna k vytvoření interaktivního průvodce, který by při první návštěvě vysvětlil pokročilejší funkce.

Po bližším zkoumání problematického připojení byl zjišten problém s knihovnou *WifiManager*, která má na starosti vytvoření kaptivního portálu a následné připojení na Wifi. Aktuální verze této knihovny prochází rapidním vývojem s nedostatečnou dokumentací aktuálních funkcí. Nepodařilo se identifikovat, zda je problém se špatným využitím této knihovny či v implementaci knihovny samotné. Pro nalezení řešení bude kontaktován autor knihovny a v případě neúspěchu je možnost využití jiné knihovny implementující podobnou funkcionalitu např. *IotWebConf*.

Závěr

Cílem této práce bylo navrhnout a implementovat vlastní IoT platformu. Všech vytyčených cílů bylo dosaženo. Byla provedena rešerše a porovnání čtyř Platforem: Bynk, ThingSpeak, Home Assistant a OpenHab. Pro návrh vlastního řešení byla vytvořena koncepce, kterou je otevřená více uživatelská platforma pro připojení různorodých zařízení vyrobených kutily a technickými entuziasty. Základním pilířem celého řešení je vytvořené schéma na protokolu MQTT, které umožňuje zařízením přesně popsat své dovednosti, na jejichž základě je vykresleno uživatelské rozhraní. Jsou podporovány různé kombinace vlasností od úzce specifických prvků jako přepínač až po obecné odeslání textu. Každý uživatel má k dispozici vlastní sféru, v rámci které jeho zařízení komunikují.

Bylo vytvořeno intuitivní webové rozhraní, které umožňuje interakci se zařízeními, vizualizaci některých dat a nastavení push notifikací. Byla vytvořena knihovna pro koncová zařízení, umožňující snadné napojení na platformu a vzniklo zařízení, na kterém bylo demonstrováno její použití. Řešení bylo nasazeno a podrobeno zátěžovému a uživatelskému testování, ve kterém se prokázala schopnost obsloužit stovky zařízení a intuitivnost celého rozhraní.

Řešení je zveřejněno a uvolněno pod licencí GPLv3 na stránce `https://github.com/founek2/IOT-Platforma`. Platformu již aktivně využívám pro svá chytrá zařízení a jejímu vývoji se i nadále budu věnovat ve svém volném čase, případně v rámci diplomové práce. Věřím, že se jedná o zajímavou alternativu ke stávajícím řešení a uvidím, zda se mi podaří do jejího budoucího vývoje zapojit i komunitu.

Literatura

[1] Homb, R. R.: MQTT – What Is It? And How Can You Use It? [online], červenec 2017, [vid. 2021-04-11]. Dostupné z: https://www.norwegiancreations.com/2017/07/mqtt-what-is-it-and-how-can-you-use-it/

[2] Fernandes, R. F.: Integrating semantic-ui modal with Redux. [online], únor 2018, [vid. 2021-04-11]. Dostupné z: https://itnext.io/integrating-semantic-ui-modal-with-redux-4df36abb755c

[3] Chris: node-red-contrib-homie-convention. [online], 2020, [vid. 2021-04-15]. Dostupné z: https://flows.nodered.org/node/node-red-contrib-homie-convention

[4] NailBuster Software Inc.: Touch LCD SPI for esp8266 (nodemcu). [online], květen 2016, [vid. 2021-04-14]. Dostupné z: https://nailbuster.com/?page_id=341

[5] KaaIoT Technologies, LLC: What is an IoT platform? [online], leden 2021, [vid. 2021-04-09]. Dostupné z: https://www.kaaproject.org/blog/what-is-iot-platform

[6] Young, J. K.: What a Mesh! Part 2-Networking Architectures and Protocols. [online], Prosinec 2008, [vid. 2021-04-09]. Dostupné z: https://www.fierceelectronics.com/components/what-a-mesh-part-2-networking-architectures-and-protocols

[7] Softima s.r.o.: Modul WiFi ESP8266 ESP-12E. [online], 2021, [vid. 2021-04-09]. Dostupné z: https://www.hadex.cz/m430-modul-wifi-esp8266-esp-12e/?gclid=CjwKCAjw9r-DBhBxEiwA9qYUpcE5RugsG_KluPdqKfUBQKR_S7ALkszlbaOvMZ6Z9OZms2n3Nw5tfRoCA9YQAvD_BwE

[8] Thomas, D.; Wilkie, E.; Irvine, J.: Comparison of Power Consumption of WiFi Inbuilt Internet of Things Device with Bluetooth Low Energy.

World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering, ročník 10, č. 10, 2016, [vid. 2021-04-09]. Dostupné z: https://reactjs.org/docs/getting-started.html

[9] Sigfox: Základní ceník konektivity. [online], 2021, [vid. 2021-04-09]. Dostupné z: https://sigfox.cz/cs/o-nas/cenik-vop

[10] LoRa Alliance®: What is LoRaWAN® Specification. [online], 2021, [vid. 2021-04-09]. Dostupné z: https://lora-alliance.org/about-lorawan/

[11] České Radiokomunikace a.s.: Služby CRA IoT. [online], 2021, [vid. 2021-04-09]. Dostupné z: https://www.cra.cz/sluzby-iot

[12] Zigbee Alliance: Zigbee FAQ. [online], 2020, [vid. 2021-03-09]. Dostupné z: https://zigbeealliance.org/zigbee-faq/

[13] Silicon Laboratories: Z-Wave: The basics. [online], 2021, [vid. 2021-03-09]. Dostupné z: https://www.z-wave.com/faq

[14] Patel, R.: IoT and home automation: What does the future hold? květen 2020, [vid. 2021-04-09]. Dostupné z: https://www.iot-now.com/2020/06/10/98753-iot-home-automation-future-holds/

[15] Blynk Inc.: Blynk IoT platform: for businesses and developers. [online], 2020, [vid. 2021-03-01]. Dostupné z: https://blynk.io/about

[16] Blynk Inc.: Pricing — Blynk IoT Platform. [online], 2020, [vid. 2021-03-01]. Dostupné z: https://blynk.io/pricing

[17] Blynk Inc.: Blynk dokumentace. [online], 2021, [vid. 2021-03-01]. Dostupné z: https://docs.blynk.cc/

[18] Blynk Inc.: Blynk server. [software], 2021, [vid. 2021-03-01]. Dostupné z: https://github.com/blynkkk/blynk-server

[19] The MathWorks, Inc.: Learn More About ThingSpeak. [online], 2021, [vid. 2021-04-01]. Dostupné z: https://thingspeak.com/pages/learn_more

[20] The MathWorks, Inc.: License Options. [online], 2021, [vid. 2021-04-01]. Dostupné z: https://thingspeak.com/prices

[21] Schoutsen, P.: Home Assistant. [online], 2021, [vid. 2021-04-01]. Dostupné z: https://www.home-assistant.io/

[22] Schoutsen, P.: Documentation. [online], 2021, [vid. 2021-04-01]. Dostupné z: https://www.home-assistant.io/docs/

[23] Schoutsen, P.: Integrations. [online], 2021, [vid. 2021-04-01]. Dostupné z: https://www.home-assistant.io/integrations/

[24] Winter, O.: ESPHome. [online], 2021, [vid. 2021-04-01]. Dostupné z: https://esphome.io/

[25] openHAB Community and openHAB Foundation e.V.: Add-on Reference. [online], 2021, [vid. 2021-04-01]. Dostupné z: https://www.openhab.org/addons/

[26] openHAB Community and openHAB Foundation e.V.: openHAB. [online], 2021, [vid. 2021-04-01]. Dostupné z: https://www.openhab.org/

[27] openHAB Community and openHAB Foundation e.V.: Welcome! [online], 2021, [vid. 2021-04-01]. Dostupné z: https://www.openhab.org/docs/

[28] Mozilla and individual contributors: Push API. [online], [vid. 2021-05-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Push_API

[29] I. Fette, A. M.: The WebSocket Protocol. RFC 6455, RFC Editor, prosinec 2011. Dostupné z: https://tools.ietf.org/html/rfc6455

[30] Harwood, T.: IoT Standards and Protocols. [online], únor 2020, [vid. 2021-04-09]. Dostupné z: https://www.postscapes.com/internet-of-things-protocols/

[31] OASIS: MQTT Version 3.1.1. [online], říjen 2014, [vid. 2021-04-09]. Dostupné z: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[32] OpenJS Foundation: Documentation — Node.js. [online], 2021, [vid. 2021-04-09]. Dostupné z: https://nodejs.org/en/docs/

[33] Microsoft: TypeScript: Typed JavaScript at Any Scale. [online], 2021, [vid. 2021-04-09]. Dostupné z: https://www.typescriptlang.org/

[34] Stack Exchange Inc.: Most Popular Technologies. [online], červenec 2020, [vid. 2021-04-11]. Dostupné z: https://insights.stackoverflow.com/survey/2020#most-popular-technologies

[35] DeBill, E.: Module Counts. [online], 2021, [vid. 2021-04-09]. Dostupné z: http://www.modulecounts.com/

[36] OpenJS Foundation: Express - Node.js web application framework. [online], 2017, [vid. 2021-04-09]. Dostupné z: https://expressjs.com/

[37] Schmukler, R.: agenda/agenda: Lightweight job scheduling for Node.js. [online], 2021, [vid. 2021-04-09]. Dostupné z: https://github.com/agenda/agenda

[38] Łukasz Kubok: Top 10 Companies Using React.js. [online], září 2020, [vid. 2021-04-11]. Dostupné z: https://selleo.com/blog/top-10-companies-using-reactjs

[39] Facebook Inc.: Getting Started – React. [online], 2021, [vid. 2021-04-09]. Dostupné z: https://reactjs.org/docs/getting-started.html

[40] Dave, M.: SQL and NoSQL Databases. *International Journal of Advanced Research in Computer Science and Software Engineering*, 08 2012.

[41] MongoDB, Inc.: JSON and BSON. [online], 2021, [vid. 2021-04-11]. Dostupné z: https://www.mongodb.com/json-and-bson

[42] Mongoose v5.12.3: Getting Started. [online], 2021, [vid. 2021-04-11]. Dostupné z: https://mongoosejs.com/docs/index.html

[43] Quinn, S.: Bulletproof node.js project architecture. [online], květen 2019, [vid. 2021-04-11]. Dostupné z: https://softwareontheroad.com/ideal-nodejs-project-structure/?utm_source=devto&utm_medium=post

[44] Foundation, O.: NodeJS security cheat sheet. [online], 2021, [vid. 2021-04-11]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Nodejs_Security_Cheat_Sheet.html

[45] M. Jones, N. S., J. Bradley: JSON Web Token (JWT). RFC 7519, RFC Editor, květen 2015. Dostupné z: https://tools.ietf.org/html/rfc7519

[46] Ryck, R. D.: The Hard Parts of JWT Security Nobody Talks About. [online], leden 2019, [vid. 2021-04-14]. Dostupné z: https://www.pingidentity.com/en/company/blog/posts/2019/jwt-security-nobody-talks-about.html

[47] Shrestha, S.: Field and Field Descriptor pattern. [online], březen 2012, [vid. 2021-04-11]. Dostupné z: http://sumitshresthatech.blogspot.com/2012/03/field-and-field-descriptor-pattern.html

[48] IoT enthusiasts: The Homie convention. [online], 2020, [vid. 2021-04-14]. Dostupné z: https://homieiot.github.io/

[49] OpenJS Foundation: Node-RED. [online], [vid. 2021-04-15]. Dostupné z: https://nodered.org/

[50] Linden Tibbets, J. T.: Welcome to IFTTT guide. [online], [vid. 2021-04-15]. Dostupné z: https://ifttt.com/explore/welcome_to_ifttt?utm_source=IFTTT&utm_medium=Tout&utm_campaign=Signedout_Audience

[51] Kawaguchi, K.: Jenkins. [online], 2021, [vid. 2021-04-19]. Dostupné z: https://www.jenkins.io/

[52] Docker, Inc.: What is a Container? [online], 2021, [vid. 2021-04-19]. Dostupné z: https://www.docker.com/resources/what-container

[53] Maxim Integrated: GUIDE TO 1-WIRE COMMUNICATION. [online], červen 2008, [vid. 2021-04-22]. Dostupné z: https://www.maximintegrated.com/en/design/technical-documents/tutorials/1/1796.html

[54] Dallas Semiconductor: DS18B20 Datasheet (PDF) - Dallas Semiconductor. [online], [vid. 2021-04-22]. Dostupné z: https://pdf1.alldatasheet.com/datasheet-pdf/view/58557/DALLAS/DS18B20.html

[55] MongoDB, Inc.: Free Monitoring — MongoDB Manual. [online], [vid. 2021-04-14]. Dostupné z: https://docs.mongodb.com/manual/administration/free-monitoring/

Příloha **A**

Seznam použitých zkratek

ACL - seznam oprávnění vázaný k zařízení, který specifikuje, kdo k němu může přistupovat a jaké operace provádět

DIY - uděl si sám

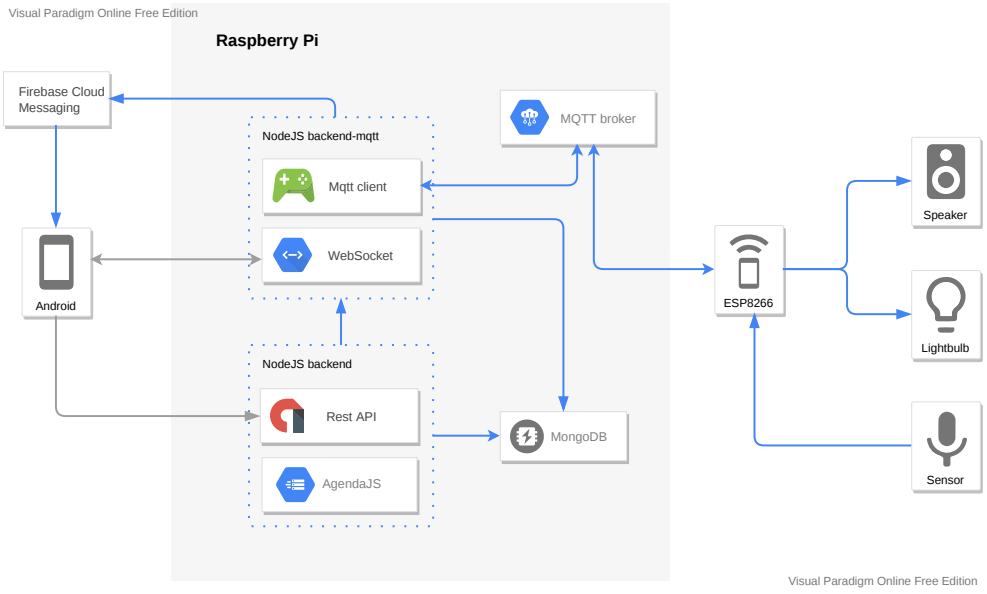
MQTT - protokol pro publikování a odběr zpráv

RESTful - REST je architektura rozhraní pro jednotný a snadným přístum ke zdrojům. Zdroje mohou být data nebo stavy aplikace. Pokud rozhraní splňuje definovanou architekturu, potom se nazývá RESTful.

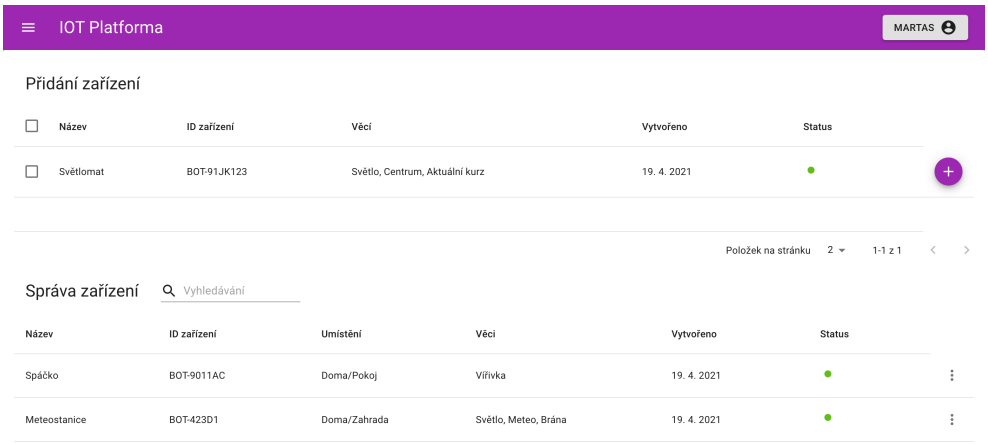
SPA - jednostránková webová aplikace, která interaguje s uživatelem dynamickým přepisováním aktuální webové stránky novými daty z webového serveru namísto načítání celé stránky

Příloha B

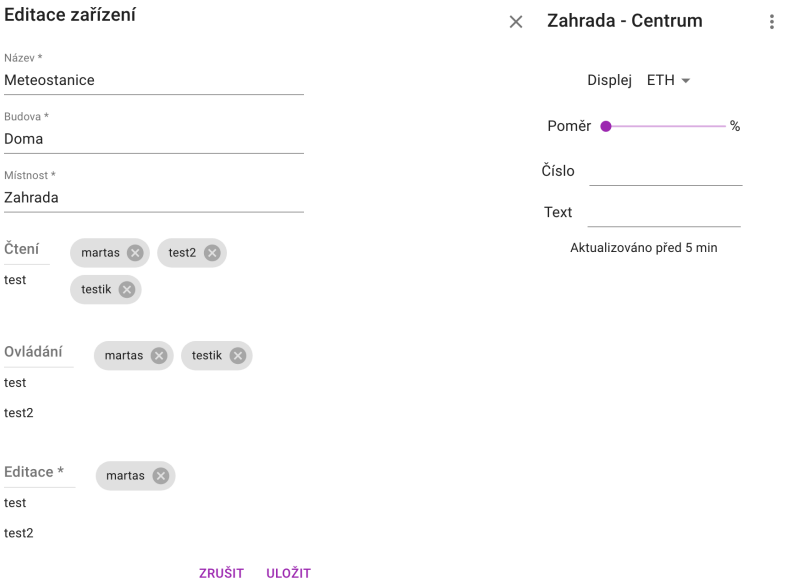
Obrázky



Obrázek B.1: Náhled fyzické architektury

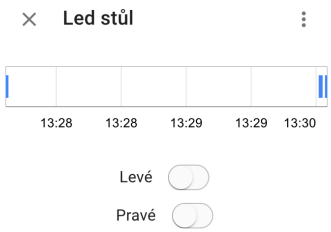


Obrázek B.2: Ukázka rozhraní - správa zařízení

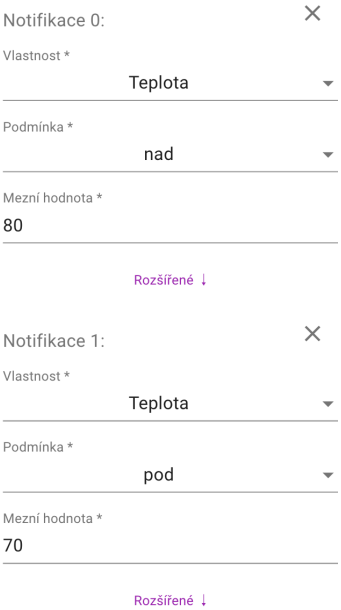


(a) Zobrazení místností (b) Zobrazení místnosti

Obrázek B.3: Ukázka rozhraní



(a) Dialog pro věc typu switch



(b) Nastavení pravidel notifikací

Obrázek B.4: Ukázka rozhraní

Příloha C

Uživatelská příručka

1. Zařízení zapněte.
2. Přihlaste se, případně zaregistrujte k platformě na stránce https://dev.iotplatforma.cloud.
3. Připojte svůj telefon/notebook k wifi s názvem „Nastav mě".
4. Budete upozorněni na nutnost přihlášení k síti, otevřete tuto možnost a zobrazí se Vám webová stránka. Klikněte na „Configure wifi", kde budete vyzváni k zadání údajů pro připojení k Vaší domácí Wifi. Dále zadejte Vaše uživatelské jméno, které používáte pro přihlášení k platformě a klikněte na tlačítko „Save". Zařízení se nyní restartuje a připojí k wifi.
5. Pokud v předchozím bodu byly zadány nesprávné údaje, tak se znovu vytvoří wifi „Nastav mě", v tom případě opakujte postup od bodu 3.
6. zařízení. Na začátku se objeví sekce "Přidat zařízení", kde budete mít nové zařízení a to si přidejte pomocí tlačítka plus. Budete vyzváni k zadání umístění zařízení.
7. Nyní na stránce Zařízení již můžete zařízení sledovat a ovládat.66

Příloha D

Obsah přiloženého média

- `platforma` ..... implementovaná platforma
  - `docs` ..... stručná dokumentace platformy
    - `deploy.md` ..... návod pro nasazení platformy
    - `quickstart.md` ..... návod pro spuštění lokálního vývoje
    - `README.md` ..... popis projektu a ukázka rozhraní
  - `packages` ..... zdrojové kódy implementace
- `smokehouse` ..... soubory pro chytrou udírnu
  - `library`
    - `IOT_platforma_v3` ..... implementovaná knihovna
  - `models` ..... model krabičky pro 3D tisk
  - `src`
    - `main.cpp` ..... zdrojový kód pro chytrou udírnu
- `performance_test_cpu_usage.ods` ..... zátěž cpu během testu
- `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `readme.md` ..... stručný popis obsahu média
- `thesis.pdf` ..... text práce ve formátu PDF