VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

REGISTR IOT ZAŘÍZENÍ

IOT DEVICES REGISTRY PORTAL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN KOPEC**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19958/2016/xkopec42

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Kopec Martin**
Obor: Informační technologie
Téma: **Registr IoT zařízení**
IoT Devices Registry Portal
Kategorie: Web

Pokyny:

1. Seznamte se s principy Internert of Things (IoT) a prostudujte způsoby správy zařízení v IoT.
2. Navrhněte portál umožňující registraci a správu zařízení v IoT. Portál bude implementovaný v Node.js, bude využívat Express framework a bude běžet v cloudu. Uživatelé se budou moci registrovat pomocí sociálních sítí. řešení bude obsahovat jak webové rozhraní, tak i rozhraní pro mobilní zařízení.
3. Navržené řešení implementujte.
4. Proveďte testování vytvořeného programu.
5. Zhodnoťte dosažené výsledky a další možné pokračování tohoto projektu.

Literatura:

- Greengard, S.: The Internet of Things. University Press Group Ltd, 2015. ISBN 9780262527736.
- Swicegood. T.: Programming Node.js. O´REILLY, 2012. ISBN 1934356891.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bartík Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L. S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cieľom tejto práce je vytvorenie webovej aplikácie, ktorá umožňuje užívateľom registrovať sa a spravovať ich zariadenia v internete vecí (IoT). Práca vysvetľuje súčasný stav a vývoj IoT zariadení, ich prínos ako aj riziká, ktoré prinášajú. Práca prezentuje momentálne používané technológie z oblasti tvorby webov, užívateľských rozhraní, databází a serverových aplikácií. Ďalej vysvetľuje, ktoré z týchto technológií sú, vďaka svojim prednostiam, vhodné pre efektívne fungovanie portálu, ktorý bude spravovať množstvo zariadení. Záver práce popisuje implementáciu portálu.

Abstract

The purpose of this thesis is to design and implement a web application which allows users to manage their Internet of Things (IoT) devices. The thesis explains the current development status of IoT, its advantages and disadvantages. The thesis focuses on modern technologies from web design, user interfaces, databases and server-side applications and then explains which technologies are suitable for implementation of such an application. In conclusion, the implementation of the portal is described.

Kľúčové slová

internet vecí, informačný systém, portál, nodejs, javascript, cloud, web

Keywords

Internet of Things, Information system, portal, nodejs, javascript, cloud, web

Citácia

KOPEC, Martin. *Registr IoT zařízení.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Bartík Vladimír.

Registr IoT zařízení

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Vladimíra Bartíka, Ph.D. a uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Kopec
15. mája 2017

Poďakovanie

Moje poďakovanie patrí pánovi Ing. Vladimírovi Bartíkovi, Ph.D. za všetky rady a konzultácie. Ďalej by som rád poďakoval pánovi Mgr. Martinovi Večeřovi z firmy Red Hat Czech s.r.o., za inšpiráciu, odborné konzultácie a znalosti, vďaka ktorým som získal väčší rozhľad v danej problematike.

Obsah

Kapitola 1

Úvod

V posledných rokoch je Internet vecí (IoT - v angličtine “Internet of Things”) stále častejšie skloňovaný pojem, ktorý je počuť v kontexte s rôznymi technologickými či priemyselnými odvetviami. Nie len priemysel, ale aj človek, jeho aktivity a bežný život sú stále častejšie spájané s týmto pojmom.

IoT je abstraktný výraz, ktorý zahŕňa komunikáciu vecí. Aby táto komunikácia mohla efektívne slúžiť, je potrebné vyvinúť systémy, ktoré budú riadiť výmenu informácií medzi zariadeniami. Táto práca sa venuje jednému príkladu takéhoto systému.

V úvode práce je vysvetlené, čo znamená technológia IoT, pre aký účel môže, či by mala slúžiť, aké výhody a uľahčenia života prináša, ale aj naopak, aké nebezpečenstvá a riziká môžu z tohto rastúceho technologického trendu vyplynúť. Práca sa tiež sústreďuje na technológie, ktoré môžu byť použité na interakciu v IoT, na existujúce systémy, ktoré s IoT pracujú či technológie, ktoré je potrebné ešte vyvinúť alebo doladiť, aby sa IoT mohlo stať efektívnou súčasťou ľudského života.

Jadro práce popisuje implementáciu systému, umožňujúceho komunikáciu zariadení v internete vecí, ich správu a poskytuje relevantné informácie užívateľovi. Ďalej sú vysvetlené rozdiely a výhody tohto systému oproti podobným, už existujúcim systémom.

V závere práce je uvedené riešenie implementačných problémov, prípadne načrtnuté možnosti, ako by mohli byť vyriešené. Ďalej sú uvedené smery, ktorými by sa projekt mohol v budúcnosti uberať.

Kapitola 2

Internet vecí

Internet vecí (v angličtine “Internet of Things” skrátene IoT) je v informatike označenie pre prepojenie fyzických zariadení, smart zariadení, budov, rôznych vstavaných zariadení (embeded systems), senzorov cez internet, čo im umožňuje zbierať a navzájom si vymieňať informácie. IoT umožňuje riadenie zariadení vzdialene cez internet, vytváranie príležitostí pre presnejšie integrovanie fyzického sveta do počítačových systémov čo sa prejavuje na zvýšení efektivity, presnosti a na ekonomickom prínose. [29]

Očakáva sa, že IoT ponúkne pokročilejšiu konektivitu zariadení, systémov a služieb, ktorá pokračuje za hranice machine-to-machine komunikácie a zahŕňa rôznorodú paletu protokolov a aplikácií, ktoré môžu vyústiť do automatizácie takmer vo všetkých oblastiach a zároveň pomôcť vyvinúť pokročilejšie aplikácie, ako napr. smart-grid[1], expandujúce až do pojmov ako inteligentné mestá (v angličtine "smart cities"). [29]

Medzi veci v ponímaní IoT patrí akékoľvek zariadenie schopné zbierať, prijímať a odosielať informácie, čo vedie k širokému spektru zariadení. Nejde iba o klasické zariadenia ako osobné počítače, kamery, domáce spotrebiče ale aj o zariadenia ako napr. kardio stimulátory, biočipy na dobytku, elektronické mušle v pobrežných vodách, automobily so vstavanými senzormi, zariadenia pre analýzu DNA na monitorovanie stravy, patogénov, životného prostredia a pod. [29]

IoT sa na zariadenia pozerá ako na kombináciu hardvéru, softvéru, dát/informácií a služby [29].

[1] Elektrická sieť, ktorá umožňuje obojstrannú komunikáciu medzi dodávateľom a odberateľom za účelom poskytnutia vyššej efektivity v dodávaní el. energie.

2.1 Prínos

2.1.1 Automatizácia

IoT umožňuje automatizovať každodenné úlohy, ktoré sa budú vykonávať bez ľudského zásahu. Komunikácia zariadení vedie k uniformovanosti úloh a môže sa prejaviť aj na zvýšení kvality služby, komfortu, pohodlnosti a lepšom riadení, čo sa v konečnom dôsledku prejaví na zvýšení kvality života. [14]

Okrem automatizácie v domácom prostredí, môžu byť automatizované rôzne výrobné procesy za účelom ich optimalizácie, či diagnostiky v prípade porúch alebo vyžadovanej údržby zariadení. [18]

2.1.2 Monitorovanie a zvýšenie bezpečnosti

Monitorovanie je všeobecný pojem. Je možné si pod ním predstaviť napr. monitorovanie zdravia alebo bezpečnosti na cestách či iných verejných miestach.

Integrovanie tejto technológie môže identifikovať zdravotné problémy na základe neustáleho monitorovania životných funkcií jednotlivca, ktorého výsledky sú preposielané doktorovi. Systém môže vyhodnotiť výskyt infarktu či inej život ohrozujúcej situácie a privolať záchrannú službu podstatne skôr, prípadne ešte predtým, ako si jednotlivec všimne nejaké príznaky. [18]

Bude tiež možné monitorovať cestnú premávku, čo by viedlo k jej efektívnejšiemu riadeniu a v prípade nehôd by systém okamžite mohol vykonať potrebné akcie na záchranu životov.

2.1.3 Efektivita, šetrenie času a peňazí

Machine-to-machine komunikácia poskytuje vyššiu efektivitu. Namiesto toho, aby človek dennodenne opakoval určité úlohy, môže si svoje každodenné úlohy nechať automatizovať a tým sa môže venovať iným kreatívnym činnostiam. [18]

Využitím tejto technológie a držaním zariadení pod dohľadom, je možné zabezpečiť optimálne využitie energie a zdrojov. V prípade porúch budú ľudia promptne informovaní o ich vzniku, čo bude mať v končenom dôsledku dopad na úspore finančných prostriedkov. [14]

2.2 Riziká

2.2.1 Možná strata súkromia

Nakoľko zariadenia rôzneho druhu (domáce zariadenia, zariadenia poskytujúce verejné služby) môžu byť pripojené na internet, všetky dôležité informácie sú dostupné online a tým sa môžu stať náchylnými na útoky zo strany hackerov. [14]

2.2.2 Kompatibilita

V IoT budú navzájom prepojené zariadenia od rôznych výrobcov, z čoho vyplýva, že bude potrebné, aby sa výrobcovia dohodli medzi sebou na určitých štandardoch a protokoloch. Problémy s kompatibilitou môžu vyústiť do vzniku monopolov na trhu. [14]

2.2.3 Komplexnosť

Všetky zariadenia budú určitým spôsobom navzájom prepojené, takže akákoľvek chyba v software alebo v hardware môže ovplyvniť veľké množstvo zariadení a tým spôsobiť vážne dôsledky. [18]

Ďalším problémom je, že kvôli automatizácií môže dôjsť k nezamestnanosti takmer vo všetkých odvetviach a oblastiach života, ktoré dnes poznáme. Avšak, toto je problém všeobecne akejkoľvek technológie a je riešiteľný vzdelávaním spoločnosti. [14]

2.2.4 Strata kontroly nad životom

Ľudské životy budú nielenže kontrolované technológiou, ale budú od nej doslova závislé. Už v súčasnosti je mladšia generácia závislá na technológiách aj v prípade, že sa jedná o jednoduché veci. Ľudia sa budú musieť rozhodnúť, do akej miery sú ochotní byť automatizovaní, kontrolovaní a ovládaní technológiou. [18]

2.3 Potrebné zmeny

V roku 2014 firma Gartner predpovedala, že do roku 2020 bude na svete 25 miliárd prepojených zariadení, čo predstavuje 3 zariadenia na osobu na celom svete. Cisco zašlo ešte o krok ďalej a predpovedalo, že počet zariadení bude dvojnásobný. [7]

Avšak, kým bude vízia o IoT plne uskutočniteľná, je potrebné nájsť odpovede na niekoľko fundamentálnych otázok týkajúcich sa IoT technológie a internetu samotného.

Ako budú zariadenia organizované? Pomenovávanie a všeobecne identita IoT zariadení je jedna z najviac kritických častí sledovania, merania a spravovania zariadení. Každé pripojené zariadenie má svoju unikátnu IP adresu. S týmto súvisí otázka, či budú musieť byť vytvorené nové metriky alebo spôsoby triedenia zariadení. [33]

Ako budú zariadenia monitorované? Ako bolo spomenuté na začiatku kapitoly 2, každé zariadenie schopné zbierať, prijímať a odosielať informácie, môže byť považované za "vec" v IoT. Z toho vyplýva nevyhnutnosť vytvoriť nielen nástroje ale aj organizácie, zaoberajúce sa sledovaním miliárd "vecí", ktoré budú predchádzať výpadkom, útokom na zariadenia a ktoré budú poskytovať správu prideľovania IP adries. [33]

Ako bude optimalizovaný výkon? Nakoľko sa miliardy zariadení stanú závislými na internete, je dôležité, už aj tak zložitý systém, spraviť viac spoľahlivým, s prihliadnutím na ešte vyšší výkon internetu ako takého. [33]

Experti predpovedajú, že do roku 2017 bude 80% nových áut pripojených na internet a budú posielať 25 GB dát každú hodinu. V ročnom meradle to znamená, že na jedno auto bude potrebné niekde uložiť 130 TB dát. A to sa týka len áut, nehovoriac o miliardách iných navzájom prepojených zariadení. [5]

Ako sa zabezpečí bezpečnosť? Rôzne nástroje môžu byť kľúčom pri varovaní pred možnými či prebiehajúcimi útokmi. Napríklad nástroje, ktoré by umožnili zariadeniam študovať útoky a naučili by sa, ako im predísť. Následne by tieto nástroje umožnili zariadeniam, takto získané vedomosti šíriť celosvetovo medzi ostatné zariadenia. [33]

Ako budú IoT zariadenia udržované? Ako bude distribuovaná a riadená aktualizácia zariadenia v rýchlo meniacom sa trhu? [33] Toto sú všetko otázky, na ktoré je ešte potrebné nájsť odpovede.

2.4 Management zariadení v IoT

V momente, keď bude zariadenie v IoT nainštalované a spustené, neznamená to koniec starostí. Bude potrebné zabezpečiť efektívne a rýchle hľadanie a opravy chýb, ako už bolo spomenuté v sekcií 2.3. Tieto potreby je možné zabezpečiť zmysluplným návrhom správy zariadení v systéme. Existujú 4 základné požiadavky [27]:

- autentifikácia
- konfigurácia
- monitorovanie a diagnostika
- aktualizácia a udržovanie software

Autentifikácia je časť procesu registrácie zariadenia do systému, ktorá zodpovedá za to, že budú registrované len zariadenia so správnymi autentifikačnými údajmi. Presné detaily, ako to zabezpečiť, môžu byť rôzne. Najčastejším spôsobom je registrácia pomocou certifikátu alebo kľúča. Pri prvom pripojení zariadenia, napr. do lokálnej siete, mu budú na základe autentifikačných údajov a ďalších informácií, ako model alebo sériové číslo, poskytnuté ďalšie konfiguračné informácie. [27]

Konfigurácia je ďalšou zo základných požiadaviek na management zariadení. Zariadenia pripojené do systému budú vo väčšine prípadov potrebovať dodatočnú konfiguráciu koncovým užívateľom, napr. nastavenie aplikačných špecifikácií na zariadení (časový interval zasielania nových údajov a pod.). [27]

Na zaistenie určitej kontroly a riešenia neznámych chýb, by mal systém podporovať možnosť obnovenia určitého stavu zariadenia (napr. bod v ktorom zariadenie fungovalo správne) alebo úplného resetu zariadenia na pôvodné nastavenia. [27]

Monitorovanie a diagnostika môžu minimalizovať následky chyby a pomôcť k ich rýchlejšiemu odhaleniu porovnávaním očakávaných štatistík (napr. vyžitie pamäte, CPU) s tými reálnymi. Napr. zvýšenie záťaže CPU na 60% na zariadení, kde je očakávaná hodnota 5%, môže znamenať, že zariadenie vykonáva neočakávané operácie. [27]

Dôležitou funkciou je tiež možnosť stiahnutia logov zariadenia, ktoré môžu poskytnúť významné informácie pri hľadaní chýb [27].

Aktualizácia a udržovanie software je 4. z podmienok správy zariadení. Na zariadenia musí byť umožnené nahrávať aktualizácie za účelom opravy chýb či pridávania funkcionality. Je potrebné zabezpečiť, aby aktualizácie prebiehali v čase, kedy to najmenej ovplyvní funkčnosť zariadenia a zároveň bude dané zariadenie pripojené k sieti, kde si môže dovoliť preniesť väčšie množstvo dát - týka sa to hlavne zariadení, ktoré budú bežne pripojené napr. cez mobilnú sieť. [27]

K správnemu odhadu času, kedy môže aktualizácia prebehnúť s najmenším rizikom, pomôžu štatistiky získané monitorovaním zariadenia. [27]

Kapitola 3

Návrh

Pred nasadením do praxe a začatím využívania IoT zariadení je potrebné uskutočniť niekoľko zmien, ktoré boli vysvetlené v sekcií Management zariadení v IoT (viď sekcia 2.4). Táto práca sa venuje navrhnutiu a vytvoreniu registračného portálu, ktorý by umožňoval registráciu zariadení a zjednodušoval ich správu. Zároveň by aspoň z časti riešil niektoré problémy správy IoT, ako je registrácia pomocou API kľúčov, zbieranie dát pre potreby monitorovania zariadení a pomoc pri údržbe zariadení (viď sekcia 2.3). Momentálne existujúce systémy, a ich porovnanie s implementovaným portálom, sú opísané v sekcií 3.1.

Registračný portál je možné z hľadiska implementácie rozdeliť na tri hlavné celky:

- front-end
- back-end
- databáza

V tejto kapitole sú načrtnuté spôsoby komunikácie zariadení s portálom a vysvetlené prednosti zvolených technológií vhodných pre implementáciu registračného portálu.

3.1 Existujúce systémy

Súčasné open-source systémy zaoberajúce sa spravovaním a monitorovaním IoT zariadení sa orientujú len na domácnosti, tzv. inteligentné domácnosti.

Medzi takéto systémy patria [6] napr. Calaos, Domoticz, OpenHAB či Home Assitant. Ide o domáce automatizované systémy, ktoré umožňujú monitorovanie a konfiguráciu rôznych zariadení v domácnosti, ako napr. svetlá, prepínače, senzory na meranie teploty, tlaku, vlhkosti, dymové senzory a pod. Tieto systémy poskytujú užívateľské rozhrania vo forme webových stránok a mobilných aplikácií. [6]

Systémy používajú na komunikáciu so zariadeniami napr. technológiu Z-Wave. Ide o bezdrôtový protokol, primárne použitý pre účely domácich automatizovaných systémov, pričom jeho cieľom je poskytnúť jednoduchú a spoľahlivú metódu, ako bezdrôtovo ovládať zariadenia. Z-Wave systém môže byť prístupný aj cez internet s použitím napr. Z-Wave brány. [21]

Ďalším komunikačným protokolom je MQTT, viď sekcia 3.5.2.

V sekcií 2 bolo načrtnuté použitie IoT zariadení, ktoré sa týka priemyselných odvetví, nie len domácností. Momentálne neexistuje systém, ktorý by sa zaoberal implementáciou správy naozaj akéhokoľvek zariadenia v IoT od rôznych strojov v strojárenskom priemysle až po bio-implantáty.

Úlohou portálu je práve zaplniť túto dieru na trhu a poskytnúť globálne a univerzálne riešenie. Portál nemá v úmysle nahradiť momentálne existujúce riešenia, ale vytvoriť vrstvu nad nimi. Tam, kde to bude vhodné, bude možné dáta zo zariadení zbierať pomocou rôznych technológií (napr. už spomenuté Z-Wave) a následne ich preposlať do portálu pre ďalšie spracovanie a vyhodnotenie.

3.2 Back-end v Node.js

Registračný portál je webová aplikácia s potenciálom potreby obsluhy veľkého množstva klientov. V tejto oblasti v poslednom čase žiari javascriptový framework Node.js [8].

Node.js je open-source multiplatformové prostredie určené na vývoj širokej palety aplikácií a nástrojov, predovšetkým na strane serveru. Node.js má udalosťami riadenú architektúru umožňujúcu asynchronné I/O operácie. Na interpretáciu JavaScriptu sa používa javascriptový engine V8 od Google. V8 neinterpretuje javascriptový kód v reálnom čase, ale prekladá ho do strojového kódu. [23]

Node.js nie je vhodný pre zložité výpočty náročné na výkon procesoru, ale je určený na rýchlo rastúce, škálovateľné webové aplikácie, pretože je schopný obslúžiť veľké množstvo simultánnych spojení. Toto je dôvod prečo vznikol. [8]

V porovnaní s klasickými webowými technikami na strane serveru, kde každá žiadosť o spojenie vytvorí nové vlákno (konkurentný server) čím zníži množstvo dostupnej pamäte RAM, Node.js pracuje v jednom vlákne a používa asynchronné funkcie, čo mu umožňuje podporu desiatok tisíc konkurentných spojení. [8]

Príklad [8]: Ak predpokladáme, že každé vlákno potrebuje 2MB pamäte a máme systém s kapacitou RAM 8GB, tak teoretické maximum konkurentných spojení na klasickom webovom serveri je 4000. Node.js dosahuje level škálovateľnosti približne 1 milión konkurentných spojení.

Avšak, zdieľanie jedného vlákna prináša isté úskalia. Výpočty nesmú byť náročné na čas, pretože by spôsobili blokovanie prichádzajúcich žiadostí o spojenie až do chvíle, kedy by bol výpočet dokončený. Práve z tohto dôvodu nie je Node.js vhodný na zložité výpočty náročné na výkon procesoru. Zložitejšie funkcie je potrebné rozdeliť na viacero neblokujúcich funkcií. Len za týchto okolností je Node.js schopný ukázať svoje kvality. [8]

V prípade, že je potrebné v rámci aplikácie vykonať nejaký zložitejší výpočet, Node.js umožňuje použitie rôznych wrapperov, ktoré predajú vykonávanie výpočtu klasickému serveru, ktorý spraví tú "ťažkú prácu". Po skončení výpočtu wrapper informuje Node.js o jeho ukončení. Týmto je možné v Node.js vytvoriť neblokujúcu operáciu nad zložitým výpočtom. [8]

Druhým problémom je, že vývojári musia zabrániť výnimkám, aby sa dostali až k jadru Node.js, pretože by to spôsobilo pád celej aplikácie. Na predídenie tejto situácie sa výnimky preposielajú späť k volajúcemu cez callback-ový parameter danej funkcie. [8]

3.2.1 Node.js frameworky

Existuje mnoho Node.js frameworkov, ktoré do rôznej miery rozširujú vlastnosti a schopnosti Node.js v závislosti na konkrétnom použití. Ďalej sú uvedené 3 frameworky (Express, Sails a Meteor), ktoré sa líšia množstvom vopred integrovaných funkcií.

Express je minimalistický webový framework, ktorý ponúka bohatú konfiguráciu a tým umožňuje programátorovi vytvoriť aplikáciu podľa jeho predstáv. Implicitne v ňom nie je integrovaná žiadna databáza a teda programátor si môže vybrať nejakú, ktorá mu najviac vyhovuje. [32]

Avšak framework má aj svoje nevýhody. Inštalácia všetkých potrebných balíčkov a ich nastavenie zaberie spočiatku nejaký čas. Express tiež neprichádza s pevne danou adresárovou štruktúrou, takže je potrebné venovať trocha extra času na navrhnutie vhodnej štruktúry projektu. [32]

Sails je postavený na Express, ale je v ňom implicitne integrovaných niekoľko funkcií. Sails napr. integruje socket.io a funkcie na minifikáciu CSS a JS súborov, takže nie je potrebné sťahovanie balíčkov a ich nastavovanie. Sails taktiež prichádza s generátorom, ktorý vytvorí súborovú štruktúru projektu. [25]

Meteor, narozdiel od predchádzajúcich dvoch frameworkov, obsahuje všetko v jednom. Oproti Express je Meteor opačný extrém. Definuje sa ako "full-stack" framework. Meteor je schopný postarať sa o serverové aj webové aplikácie a pomocou rôznych nástrojov je schopný vytvoriť aj iOS a Android aplikácie. [32]

Pre registračný portál bude použitý Express, pretože ide o minimalistický framework, ktorý si sám programátor rozšíri o balíčky, aké mu najviac vyhovujú. Spomedzi ostatných frameworkov je najviac stabilný, stojí za ním veľká komunita a v rôznych rebríčkoch je na prvom mieste.

3.3 Databáza

V registračnom portáli bude nevyhnutná implementácia databázy, obsahujúca informácie o užívateľoch, zariadeniach a skupinách užívateľov a zariadení. Okrem toho bude potrebné ukladať informácie, ktoré jednotlivé zariadenia vysielajú, pre možnosť sledovania ich vývoja v čase, napr. výška teploty od jednotlivých teplotných čidiel a pod.

3.3.1 NoSQL databázy

NoSQL je skupina databázových technológií, ktoré boli vyvinuté tak, aby vyhoveli novým požiadavkám na databázu, vychádzajúcich z tvorby moderných aplikácií. NoSQL databázy umožňujú vloženie dát bez vopred predefinovanej schémy. Inak povedané, napr. do databázy zákazníkov je možné pridať informáciu o ich telefónnych číslach za behu vývoja, pričom v relačnej databáze, je potrebné na to myslieť dopredu. [15]

Taktiež natívne podporujú ukladanie často používaných dát do vyrovnávacej pamäte a teda nie je nutné použitie iných produktov zabezpečujúcich túto schopnosť [15].

Ďalšou výhodou je, že väčšina NoSQL databáz natívne podporuje horizontálnu fragmentáciu, čo znamená, že databáza automaticky ukladá jednotlivé fragmenty na rôzne servery. Dáta a dotazy sú automaticky balancované pomedzi servery a v prípade chyby serveru, je tento server rýchlo nahradený bez toho, aby to koncovú aplikáciu ovplyvnilo. [15]

Podpora automatickej replikácie dát patrí medzi ďalšiu vlastnosť NoSQL databáz. Každá replika pozostáva z niekoľko kópií dát, pričom jedna kópia sa správa ako primárna. V prípade pádu primárnej kópie, sa stáva primárnou jedna zo sekundárnych kópií. Sekundárne kópie sa môžu použiť aj na vyrovnanie rozloženia čítacích operácií na serveroch. [15]

MongoDB databáza

Ako databáza užívateľov, skupín a zariadení bude použitá MongoDB, ktorá patrí medzi najobľúbenejšie spomedzi databáz vhodných na použitie s Node.js. Ide o open-source multiplatformovú databázu klasifikovanú ako NoSQL. [12]

MongoDB má veľkú používateľskú základňu a disponuje vlastnosťami NoSQL databáz, ktoré ju robia vhodnou voľbou pre registračný portál, ktorý potrebuje rýchlu, škálovateľnú databázu s určitou redundanciou na zvýšenie odolnosti dát.

InfluxDB databáza

Pre ukladanie informácií, ktoré zariadenia zaznamenali v nejakom časovom horizonte, je vhodné použiť databázu zo skupiny NoSQL. Klasické SQL databázy rastú smerom nadol, pridávaním riadkov. Avšak tabuľky údajov v čase by z pohľadu SQL rástli smerom do strán. Implementácia v SQL by bola príliš zložitá a vyhľadávanie by bolo časovo náročné. [20]

InfluxDB je open-source databáza napísaná v jazyku Go, optimalizovaná na rýchle a efektívne pracovanie s údajmi v čase, ako napr. údaje z monitorovaní, analýz či zbieraní dát zo senzorov v IoT. [26]

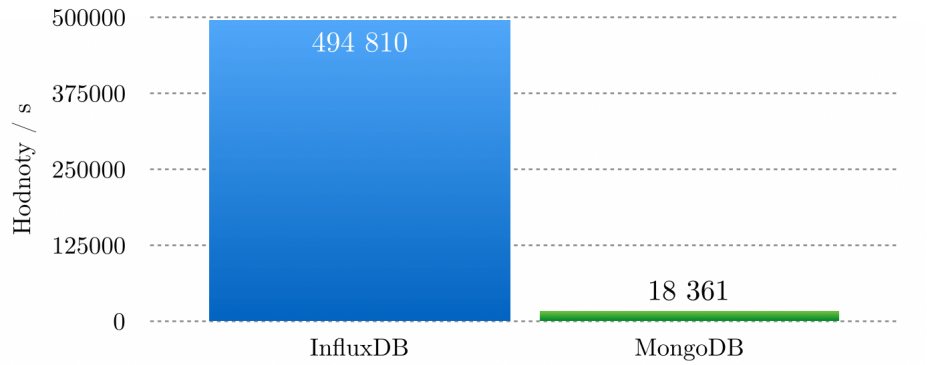
Medzi hlavné výhody InfluxDB patrí [13]:

- rýchla inštalácia (nemá žiadne závislosti na iné balíky)
- pre prácu s ňou poskytuje jazyk podobný SQL
- podpora spracovania 1 milióna hodnôt za sekundu

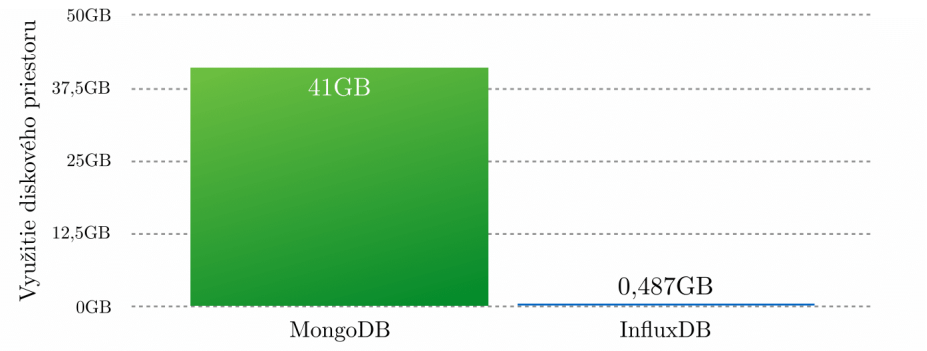
Klasické NoSQL databázy, ako napr. MongoDB, by bolo potrebné na prácu s hodnotami v čase najprv nakonfigurovať. Bolo by nevyhnutné spraviť zásadné rozhodnutia o štruktúrovaní kolekcií a dátových typov, čo by si vyžadovalo určitý čas. Navyše by to malo dlhodobé vplyvy na to, ako by sa s databázou muselo pracovať. [11]

Na druhej strane, InfluxDB je po inštalácií schopná pracovať ihneď. Navyše podľa záťažových testov vykonaných v priebehu 6 hodín na počte 1000 serverov, pričom každý server meral 100 rôznych veličín a posielal dáta do databázy každých 10 sekúnd (spolu to tvorí 216 miliónov hodnôt), má Influx DB v porovnaní s MongoDB [11]:

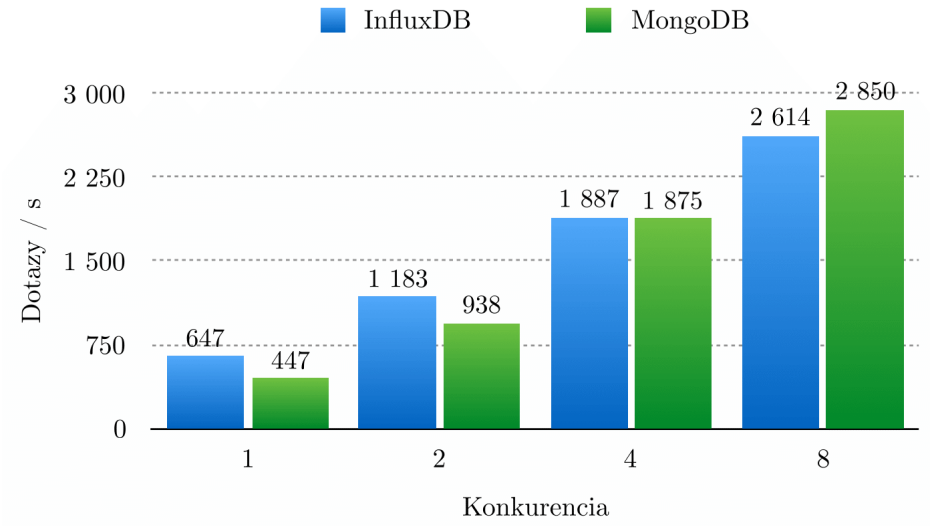
- 27-krát rýchlejší zápis (obrázok 3.1)
- 84-krát lepšiu kompresiu dát (obrázok 3.2)
- relatívne porovnateľný výkon čo sa týka rýchlosti spracovávania dotazov (obrázok 3.3)



Obr. 3.1: Rýchlosť zápisu (prevzatý z [11]).



Obr. 3.2: Nároky na diskový priestor (prevzatý z [11]).



Obr. 3.3: Rýchlosť spracovávania dotazov (prevzatý z [11]).

Z uvedeného vyplýva, že na ukladanie hodnôt v čase a ich spracovávanie je InlfuxDB momentálne najvhodnejšia voľba.

3.4 Front-end

Je vyžadované, aby užívateľské rozhranie bolo užívateľsky prístupné a podporovalo väčšinu internetových prehliadačov, nielen na počítačoch, ale aj menších zariadeniach, ako sú tablety a mobilné telefóny. Na splnenie požiadaviek je vhodný React.js.

3.4.1 React.js

React.js je javascriptový framework, určený pre tvorbu užívateľských rozhraní, spravovaný firmou Facebook [31].

Medzi výhody React.js patrí použitie virtuálneho DOM (Document Object Model), čo znamená, že tento model je vytváraný v pamäti RAM a pri zmene stavu niektorej z komponent porovnáva rozdiely medzi modelom v pamäti a modelom, ktorý je vyrenderovaný. Následne React vyrenderuje znovu len zmenené komponenty a nie celú stránku, ako to robí klasický JavaScript. [31]

Ďalšou výhodou je JSX. Komponenty v React sú typicky písané v JSX. Je to javascriptové rozšírenie, ktoré dovoľuje použitie HTML syntaxe na renderovanie podkomponent. Inak povedané, robí písanie JavaScriptu menej zložitým, nakoľko umožňuje kombináciu HTML s JavaScriptom. [31]

Väčšina JS frameworkov nie je veľmi vhodná pre vyhľadávacie nástroje. Avšak, React je prispôsobený pre SEO (Search Engine Optimization), pretože virtuálny DOM je vyrenderovaný a vrátený prehliadaču ako klasická webová stránka. [10]

Použitie len čistého React.js je pri implementačne rozsiahlejších aplikáciách komplikované a preto je tiež vhodná implementácia technológie Redux.

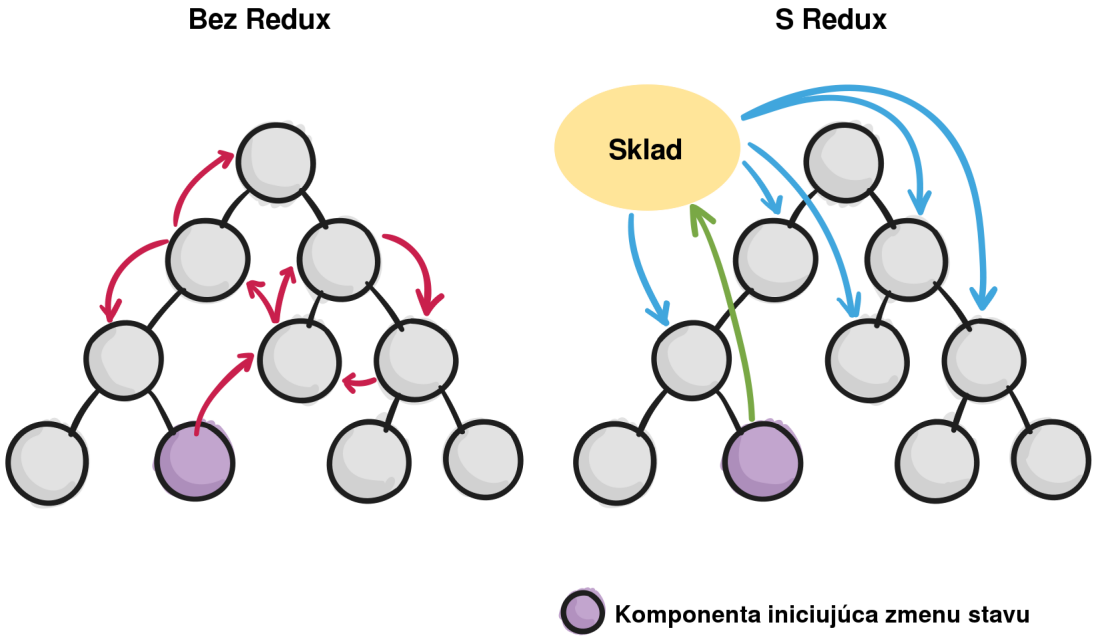
3.4.2 Redux

Redux je kontainer stavov pre javascriptové aplikácie. Požiadavky na javascriptové stránky sa v súčasnosti zvyšujú. Znamená to, že zdrojový kód musí spravovať omnoho viacej stavov jednotlivých komponent, ako napr. odpovede zo strany servera a lokálne vytvorené dáta, ktoré ešte neboli perzistentne uložené v databáze alebo slúžia len ako dočasné dáta. [3]

Redux umožňuje vytvorenie dátového skladu, ktorý je možné volať z akejkoľvek komponenty, nech je akokoľvek zanorená [3]. Sklad môže, okrem stavov, obsahovať pomocné texty pre interakciu s užívateľom, varovné hlášky a ich verzie v rôznych jazykoch, pričom všetky tieto informácie sú uložené na jednom mieste, čo umožňuje ich jednoduchú správu a pridávanie nových dát. Zároveň to sprehľadňuje javascriptový kód, pretože veľa informácií nie je potrebné prenášať z komponenty na komponentu.

Obrázok 3.4 znázorňuje komunikáciu rôzne zanorených komponent medzi sebou. V prípade nepoužitia Reduxu (časť obrázku vľavo), ak je potrebné, aby sa na základe vyvolania zmeny niektorej z komponent prerenderoval aj obsah inej komponenty, je nutné tejto komponente danú informáciu odovzdať, čo v prípade komplexnejšej štruktúry môže spôsobovať neprehľadnosť kódu.

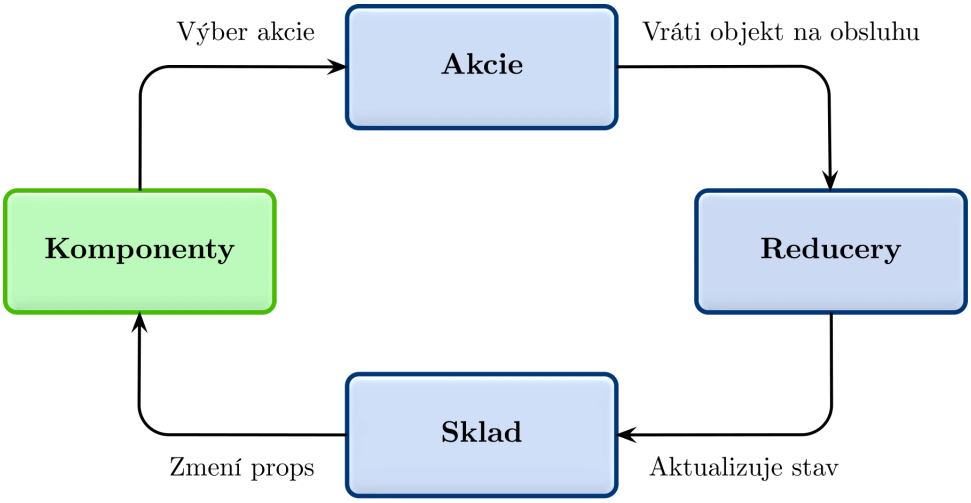
V prípade použitia Reduxu (časť obrázku vpravo), tento problém komunikácie odpadá. Komponenta oznámi svoju zmenu stavu len dátovému skladu. Následne všetky komponenty, ktoré sú pripojené na dátový sklad, obdržia túto informáciu a zmenia svoj stav [1].



Obr. 3.4: Porovnanie komunikácie komponent bez a s dátovým skladom (prevzatý z [28]).

Zvýšením náročnosti a komplexnosti na stavy komponent sa stáva kód ťažšie spravovateľný, nakoľko zmena stavu v jednej komponente môže spôsobiť zmenu stavov v niekoľko ďalších komponentách a atď. Redux sa preto pokúša predpovedať mutácie stavov zavedením istých obmedzení [4]:

- stav celej aplikácie je uložený v stromovom objekte v jednom redux sklade, čo umožňuje jednoduchšie hľadanie chýb, urýchľuje vývoj a celkovo robí aplikáciu univerzálnejšou
- stav je určený len na čítanie, čo znamená, že sa nedá meniť priamo. Namiesto toho sa zámery o zmenu stavov uložia na jedno centralizované miesto, kde sa vykonajú postupne, čím sa predíde vzniku súbehu (race condition)
- zmeny stavov sú vykonané tzv. "čistými" funkciami (pure functions), ktoré na rovnaký vstup vrátia vždy rovnaký výstup, pretože neuchovávajú žiadny stav a teda ich výstup nie je závislý na žiadnych vedľajších vplyvoch



Obr. 3.5: Vyvolanie a priebeh zmeny stavu v komponente [1].

Obrázok 3.5 znázorňuje priebeh zmeny stavu v komponente s použitím Redux. Komponenta vyvolá akciu na zmenu stavu, ktorá vráti objekt reduceru. Reducer sa postará o zmenu stavu a aktualizuje sklad. Následne sklad pošle nový stav všetkým komponentám, ktoré sú prihlásené na odber daného stavu, čo má za následok ich opätovné vykreslenie.

3.5 Komunikácia

Je nevyhnutné zabezpečiť komunikáciu nie len medzi užívateľom a portálom, ale aj medzi jednotlivými zariadeniami a portálom z dôvodu napr. dodatočnej konfigurácie zariadenia, viď sekciu 2.4.

Užívatelia budú s portálom komunikovať cez protokol HTTP. Užívateľské webové rozhranie vytvorí na podnet užívateľa HTTP žiadosť na server. Server žiadosť spracuje a pošle odpoveď vo forme žiadaných dát s HTTP návratovým kódom. Obsluha týchto žiadostí bude na strane servera implementovaná pomocou API rozhraní.

Komunikácia zariadení s portálom by mohla taktiež prebiehať cez HTTP protokol. Avšak, je vhodné, aby portál mohol so zariadeniami komunikovať aj na užívateľovu iniciatívu, napr. zapnutie/vypnutie zariadenia, zmena jeho konfigurácie, vyžiadanie nových dát zo zariadenia a pod. V takomto prípade, by bola komunikácia cez HTTP problematická, nakoľko by zariadenie muselo mať statickú IP adresu a musel by byť na zariadení spustení HTTP server, ktorý by spracovával žiadosti od portálu (užívateľa). Preto bude komunikácia, hlavne medzi portálom a zariadením, zabezpečená pomocou socketov[1].

3.5.1 Socket.io

Jedna z možností ako implementovať komunikáciu pomocou socketov je cez knižnicu Socket.io

Socket.io podporuje obojsmernú udalosťami riadenú komunikáciu v reálnom čase. Socket.io pracuje na každej platforme, prehliadači a zariadení, pričom sa zameriava na spoľahlivosť a rýchlosť. [9]

Pripojenie zariadenia je spravované Socket.io knižnicou, ktorá zabezpečí handshake zariadenia a portálu a výmenu všetkých potrebných informácií. V réžii programátora je vytvorenie obsluhy na pripojenie a odpojenie zariadenia. Tieto obsluhy budú zavolané samotnou knižnicou na základe vyvolania udalosti pripojenia alebo odpojenia zariadenia.

3.5.2 MQTT

Ďalšou možnosťou ako implementovať komunikáciu medzi zariadeniami je využitie MQTT protokolu, ktorý je oproti Socket.io (viď 3.5.1) viac komplexnejší a prináša niekoľko už vstavaných prostriedkov.

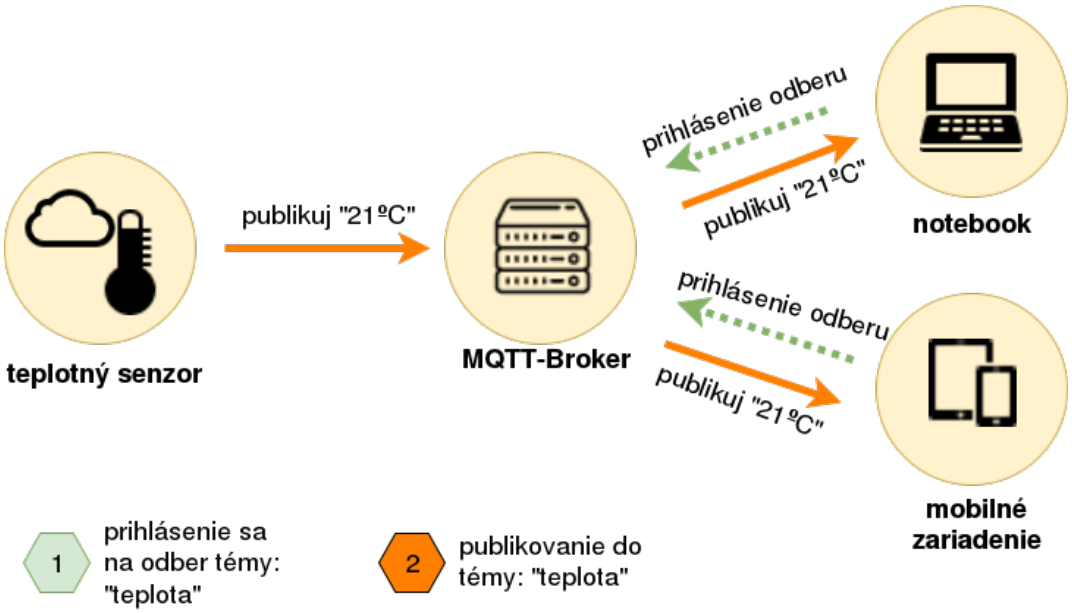
MQTT je komunikačný protokol, ktorý bol navrhnutý s prísnym ohľadom na jednoduchosť. Na machine-to-machine komunikáciu používa publish-subscribe vzor, vďaka čomu plní dôležitú úlohu v IoT. [17]

Tento protokol umožňuje zariadeniam (klientom) publikovať hodnoty v danej téme na server, ktorý sa nazýva broker. Broker následne pošle informácie tým klientom, ktorí sa predtým prihlásili na odber danej témy. Pre bližšiu predstavu je možné si témy predstaviť ako cestu k súboru, napr. /domov/teplota/obyvacia_miestnost. Klienti sa môžu prihlásiť na odber na konkrétnu úroveň danej cesty, prípadne použiť wildcard znaky na prihlásenie sa na odber viacerých úrovní. [17]

V prípade, že sa preruší spojenie medzi subscriber-om a broker-om, broker uloží všetky správy do fronty a v momente keď subscriber prejde späť do stavu online, obdrží správy z fronty. [17]

[1] Socket je dátová štruktúra reprezentujúca sieťové spojenie medzi dvomi zariadeniami.

Obrázok 3.6 znázorňuje situáciu, kedy teplotný senzor publikuje údaj o teplote, broker ho spracuje a pošle všetkým klientom, ktorí sú prihlásení na odber teploty.



Obr. 3.6: Vzťah publisher - broker - subscriber (prevzatý z [24]).

Protokol obsahuje minimum autentifikačných vlastností a z toho vyplývajúce nevýhody [17]:

- Užívateľské mená a heslá sú posielané ako plain-text.
- Bez použitia externých prostriedkov je nemožné v MQTT zistiť, kto vlastní tému a kto môže do nej prispievať. Ak správa neobsahuje informácie o jej originálnom pôvode, prijímateľ nemá možnosť zistiť, kto správu publikoval.
- MQTT bol navrhnutý ako jednoduchý protokol, aby bola jeho implementácia čo najjednoduchšia, avšak, potrebné bezpečnostné prvky, ktoré by museli byť implementované nad ním robia celkovú implementáciu zložitejšiu.

3.6 Tokeny a API kľúče

V kapitole 2 boli načrtnuté bezpečnostné riziká a z toho plynúca potreba chránenia osobných údajov. Medzi najbežnejšie spôsoby zabezpečenia patrí autentifikácia pomocou tokenov a API kľúčov.

V súčasnosti sa autentifikácia na báze tokenov používa takmer kdekoľvek. Aby sa užívateľ nemusel po každom prepnutí či obnovení stránky prihlasovať a opätovne zadávať svoje prihlasovacie údaje, je užívateľovi po úspešnom prihlásení sa pridelený token. Následne je tento token použitý pri akejkoľvek interakcií so serverom. [22]

API kľúč je istý druh tokenu, ktorý sa používa na autentifikáciu. Existuje viacero spôsobov ako vygenerovať a použiť API kľúč, avšak ich skúmanie nie je predmetom tejto práce.

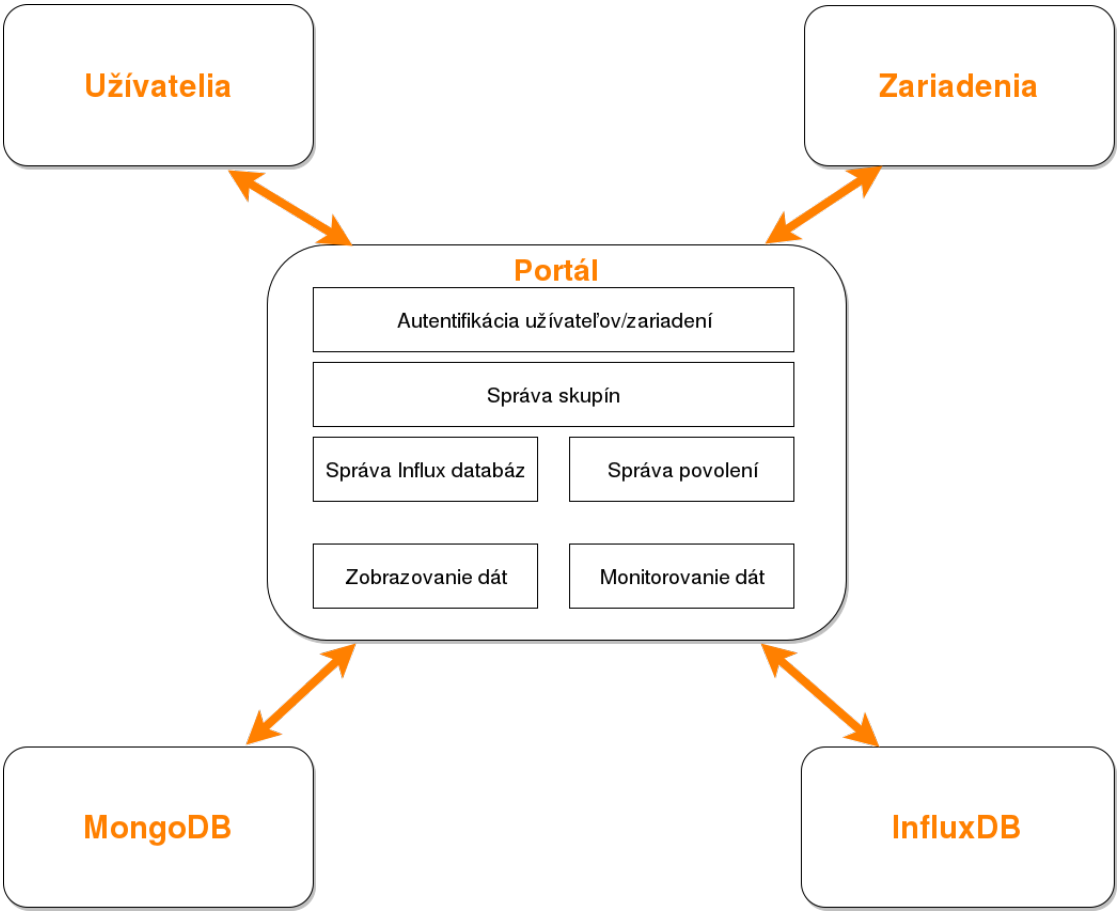
Kapitola 4

Implementácia

Ako už bolo v kapitole 3 načrtnuté, cieľom práce je vytvoriť portál pre správu zariadení v IoT. Rozdiely oproti už existujúcim podobným systémom boli spomenuté v sekcií 3.1.

Technológie, ktoré budú použité pri implementácií a dôvod ich výberu boli opísané v sekcií 3. Z praktického hľadiska je možné si úlohu takéhoto portálu predstaviť ako vrstvu nad MongoDB a InfluxDB, ako to znázorňuje obrázok 4.1. Užívatelia a zariadenia budú komunikovať s portálom a portál s databázami. Úlohou portálu je poskytnúť služby nad databázami:

- Autentifikácia užívateľov a zariadení, viď sekciu 4.3.
- Správa skupín - užívateľ bude schopný vytvárať skupiny užívateľov a zariadení pre ich prehľadnejšiu správu.
- Správa influx databáz - užívateľ bude môcť spravovať influx databázy pre ukladanie informácií zo zariadení, viď sekciu 4.5.
- Správa povolení - portál musí okrem autentifikácie poskytovať aj služby pre pridelenie resp. odobratie prístupu užívateľov k zariadeniam pre čítanie (zobrazenie dát publikovaných zariadením) a zápis (ovládanie zariadenia).
- Zobrazenie dát prehľadným spôsobom, viď sekciu 4.8.
- Monitorovanie dát a upozornenie na podozrivú aktivitu, ktorá sa vymyká štatistike či normám definovaných užívateľom, viď sekciu 4.8.



Obr. 4.1: Portál, ako medzivrstva medzi užívateľmi/zariadeniami a dátami

4.1 Influx databáza

Výhody Influx databázy boli opísané v sekcii 3.3.1. V tejto časti budú uvedené informácie o jej praktickom použití.

Pre prácu s databázou v Node.js (ale aj v inom programovacom jazyku) je vhodné použiť klienta pre danú databázu. Klient je vrstva nad API funkciami danej databázy (wrapper). Tento klient uľahčuje programátorovi prácu, nakoľko nie je potrebné vytvárať priamo žiadosti na databázu, spraví ich klient a tým sa kód stáva čitateľnejším a prehľadnejším, ako to dokazuje príklad v prílohe A.1. Napríklad pre prácu s MongoDB je v práci použitý mongoose klient. Je to jeden z viacerých klientov vhodný pre Node.js [2]. V práci je experimentálne použitý neoficiálny klient [16], ktorý mal v čase písania najvyšší počet stiahnutí.

Príklad v prílohe A.1 zobrazuje rozdiel v množstve kódu potrebného pre uloženie dát v InfluxDB, kde v časti:

a) je napísaný kód, ktorý vytvorí žiadosť na InfluxDB cez HTTP rozhranie

b) je použitý InfluxDB klient

Pre zapísanie hodnôt do databázy potrebuje použitý klient poznať adresu databázy, jej meno a tzv. schému.

Schéma je definovaná ako pole (viď príklad v prílohe A.2) objektov, ktoré sa skladajú z názvu merania ("measurement"), položiek merania ("fields") a dodatočných značiek ("tags"), ktoré poskytujú meta informácie o konkrétnom zápise. Táto schéma definuje aké hodnoty a v akom dátovom type sa budú do databázy zapisovať. InfluxDB je NoSQL databáza (viď 3.3.1) a teda meranie je možné si predstaviť ako kolekciu, ktorú deklaruje schéma a vytvára model daného merania (kolekcie). Potom jeden zápis do databázy, ako je napr. znázornený v prílohe A.2, sa v NoSQL slovníku nazýva dokument.

Značky bližšie identifikujú jednotlivé dokumenty, čo pomáha pri vyhľadávaní v databáze a umožňuje filtrovať výsledky na základe dodatočných informácií, napr. názov zariadenia, ktoré vytvorilo dokument a pod.

Je dôležité si predstaviť, že jedno zariadenie môže merať viac veličín alebo sledovať viac stavov, takže fakt, že klient prijíma schému definovanú ako pole, je praktické a teda nie je nutné implementovať dodatočnú logiku na strane serveru, čo uľahčuje programátorovi prácu a sprehľadňuje zdrojový kód.

Začína sa prejavovať potreba vzniku komunikačného štandardu, pretože server potrebuje poznať pri vytváraní dokumentu schému zariadenia a nakoľko používa Influx databázu, potrebuje ju poznať v Influx syntaxi. Z toho vyplýva, že zariadenia by ju museli implementovať, čo by však mohlo byť obmedzujúce, pretože by museli prinajmenšom byť schopné importovať potrebný modul. Vhodnejšou alternatívou by bolo vytvorenie centralizovanej logiky, kde by bolo dohodnuté rozhranie medzi zariadeniami a servermi spracovávajúcimi ich hodnoty. Tejto problematike sa venuje sekcia 4.2.

4.1.1 Problémy

Pri implementácií vyplynulo, že momentálne neexistuje oficiálny InfluxDB klient pre Node.js. Existuje niekoľko neoficiálnych klientov, avšak s nimi prichádza niekoľko problémov. Nakoľko nepochádzajú z oficiálnych zdrojov, nie je zaručená úplná kompatibilita s aktuálnou verziou databázy a teda nemusí byť možné urobiť pomocou klienta všetky úkony, ktoré je možné spraviť pomocou oficiálneho rozhrania.

Ďalej nie je zaručená jeho správna funkčnosť. Klient v čase písania práce nemal do detailov presnú dokumentáciu a vyskytli sa problémy s vytvorením novej Influx databázy. Nakoniec sa v portáli pre vytvorenie novej databázy nepoužil klient, ale bol napísaný kód, ktorý vytvorí HTTP žiadosť na Influx server manuálne.

InfluxDB má celkovo menej presnú a menej podrobnú dokumentáciu v porovnaní napr. s MongoDB. Avšak komunita ľudí okolo InfluxDB sa neustále rozrastá a dá sa predpokladať, že dokumentácia sa v blízkej budúcnosti zlepší a pribudne tiež oficiálne podporovaný klient pre Node.js. InfluxDB je pomerne nový projekt (vznik v 2013 [26]) s cieľom na použitie práve s technológiou IoT, ktorá sa v súčastnosti ešte nepoužíva globálne.

4.2 Komunikačný štandard

Pre komunikáciu medzi portálom (alebo akýmkoľvek iným podobne zameraným systémom) a zariadeniami je nevyhnutné stanoviť určitý štandard týkajúci sa formátu zasielaných správ. Bez pevne stanoveného štandardu je akýkoľvek vývoj, či už frontendových alebo backendových systémov, veľmi obtiažny.

Pri implementácií registračného portálu museli byť určené niektoré základné atribúty správ, ako napr. atribút v ktorom je uložený API kľúč pri registrácií zariadenia, token alebo vstupno/výstupné schémy zariadenia (viď 4.5).

Aby bolo možné vo frontend naprogramovať rôzne tlačidlá alebo všeobecne akcie, ktoré by posielali zariadeniu príkazy na napr. zapnutie, vypnutie, zmenu časového intervalu a pod., je opäť nevyhnutné, oprieť sa o nejaký štandard. Inak nebude možné vyvinúť frontendové riešenie, ktoré by bolo všeobecne použiteľné.

Výhodou štandardu by tiež bolo, že dané užívateľské rozhranie by vedelo rozlíšiť podľa vstupno/výstupnej schémy zariadenia, ktorým správam zariadenie rozumie a podľa toho by sa mohli vyrenderovať potrebné grafické komponenty, ktoré by slúžili na ovládanie alebo dodatočnú konfiguráciu zariadenia.

Štandardizovaný formát správ by pomohol aj výrobcom alebo jednotlivým užívateľom zariadení pri nastavovaní IoT zariadení tak, aby boli ovládateľné pomocou akéhokoľvek centralizovaného systému.

Formátom komunikačného štandardu by sa mohol stať JSON. JSON je dátový formát určený pre prenos dát v ľubovoľnom programovacom či skriptovacom jazyku, je čitateľný pre človeka a pozostáva z párov hodnôt (kľúč a jeho hodnota). [30]

JSON je v zariadení definovaný ako obyčajný string, takže zariadenie nepotrebuje importovať externé moduly.

V rámci komunikačného štandardu by mohol vzniknúť aj modul, ktorý by bol zodpovedný za preklad JSON správy od zariadenia do potrebného objektu, napr. do Influx objektu, prípadne iného[1]. Aplikácia na strane serveru by bola následne schopná modul importovať, preložiť správu od zariadenia, získať napr. Influx objekt a uložiť ho do databázy, alebo ho iným spôsobom spracovať.

Ak by bolo zariadenie kompatibilné s komunikačným štandardom, znamenalo by to len, že názvy atribútov a ich štruktúra alebo zanorenie sa rovnajú tým, ktoré sú definované v štandarde.

4.3 Bezpečnosť

API kľúče sú použité pri registrácii zariadení a tokeny sú používané pri akejkoľvek interakcii užívateľov/zariadení so serverom.

Táto práca sa nevenuje pokročilejším bezpečnostným technikám generovania token-ov alebo API kľúčov. V tejto práci sú tokeny a API kľúče náhodne generované. Token je dlhý 512 a API kľúč 8 znakov. Obe hodnoty je možné zmeniť v konfiguračnom súbore.

4.3.1 Token

Tokeny sú uložené v databáze spolu s informáciou o ich majiteľovi. Tokeny, ktoré sú po istú dobu nepoužité, nebola s nimi prijatá žiadna žiadosť od užívateľa, sú z databázy uvoľ-

[1] Dá sa predpokladať, že s vývojom IoT vzniknú aj iné databázy podobné Influx.

nené. Keď je užívateľ neaktívny, predpokladá sa, že sa zabudol odhlásiť a preto sa tento token z bezpečnostných dôvodov vymaže. Pri cielenom odhlásení sa (užívateľ klikol tlačidlo Odhlásiť) je token okamžite uvoľnený z databázy.

Portál generuje tokeny nielen pre užívateľov, ale aj pre registrované zariadenia. Princíp pre zariadenia je analogický. Vždy, keď bude zariadenie chcieť poslať nové informácie na server, poskytne svoj token. Avšak, situácia zo zariadeniami je trocha odlišná.

Zariadenie obdrží token, narozdiel od užívateľov, po úspešnej registrácií. Z pohľadu portálu je platnosť tokenu neobmedzená, nakoľko sa zariadenie autentifikovalo pomocou API kľúča pri jeho registrácií. Z pohľadu bezpečnosti by však takéto riešenie nebolo vhodné. Preto ak je zariadenie registrované, disponuje validným tokenom a pokúsi sa o opätovnú registráciu, portál mu poskytne nový token a starý uvoľní z databázy. Interval takejto opätovnej registrácie je plne v réžií užívateľa, ktorý musí svoje zariadenie nastaviť.

4.3.2 API kľúč

Užívateľ musí pred registráciou zariadenia vygenerovať API kľúč. Tento kľúč je uložený v databáze a jeho platnosť je obmedzená. Po vypršaní určitého času (v práci nastavenej na 30 minút) alebo použití kľúča (registrácií zariadenia) je uvoľnený z databázy a nie je možné ho použiť.

Pri registrácií musí zariadenie použiť API kľúč. Na základe toho portál zaregistruje zariadenie užívateľovi, ktorému patrí daný API kľúč a pošle zariadeniu ďalšie konfiguračné údaje (viď 4.5).

Kľúč je dlhý len 8 znakov nakoľko je vygenerovaný pred registráciou nového zariadenia. Je to z dôvodu jednoduchšieho prepísania kľúča do zariadenia. Ak by sa však vyvinula aplikácia zo sekcie 4.8.2, kľúč by mohol pozostávať z ľubovoľne rozumne dlhej postupnosti znakov.

4.3.3 Registrácia cez sociálne siete

Registrácia pomocou sociálnych sietí je v súčasnosti trendom, ktorý uľahčuje život hlavne užívateľom, ktorí si nemusia pamätať nové prihlasovacie údaje. Princípom registrácie cez sociálne siete je presmerovanie žiadosti o registráciu/prihlásenie užívateľa na autentifikačné servery danej sociálnej siete, ktoré sa postarajú o jeho autentifikáciu.

Existuje niekoľko modulov pre Node.js, ktoré sa postarajú o komunikáciu s autentifikačnými servermi sociálnej siete, takže implementácia takejto registrácie nie je náročná a dá sa rozdeliť do troch krokov:

1. Registrovanie aplikácie na vývojárskom portáli danej sociálnej siete. Výsledkom je získanie ID a tajného kľúča. Tieto údaje budú potrebné pre komunikáciu s autentifikačným serverom.

2. Stiahnutie klienta, ktorý zabezpečí komunikáciu s autentifikačným serverom.

3. Implementácia obsluhy odpovede od klienta. Návratovou hodnotou klienta je buď objekt reprezentujúci chybu alebo objekt prihláseného užívateľa, ktorý obsahuje základné informácie ako email a meno užívateľa, alebo v prípade potreby aj ďalšie profilové informácie, ako napr. fotka užívateľa, dátum narodenia, počet priateľov a pod.

Autentifikačné servery sociálnej siete budú, po obdržaní žiadosti o registráciu a jej overení, kontaktovať server, ktorého adresa je uvedená v žiadosti o registráciu. Nakoľko

by z tohto dôvodu musel server disponovať verejnou IP adresou, ktorú serverová aplikácia, v čase testovania tejto práce, nemala z dôvodu opísaného v sekcii 4.7, nemohla byť táto funkcionalita odskúšaná.

4.4 Užívateľské rozhranie

Na implementáciu užívateľského rozhrania bol použitý, javascriptový framework, React.js spolu s Redux, viď sekciu 3.4.

4.4.1 PatternFly

Grafická stránka užívateľského rozhrania bola implementovaná pomocou PatternFly[2] frameworku. Ide o open-source projekt, ktorý je spravovaný komunitou dizajnérov a developerov. PatternFly poskytuje nadefinované štýly pre tvorbu podnikových webových aplikácií.

Nevýhodou PatternFly však je, že neposkytuje aj mobilnú verziu niektorých komponent. Pri implementácií sa ukázalo, že vertikálne menu aplikácie, nie je na zariadeniach s menšou obrazovkou skryté a zobrazené len na podnet užívateľa, čo komplikuje čitateľnosť aplikácie. Do budúcna je nutné použiť iný framework, ktorý takúto vlastnosť obsahuje, alebo implementovať vlastný kód. V ďalšom pokračovaní projektu by bolo vhodné vytvoriť mobilnú aplikáciu, viď 4.8.1.

4.4.2 Redux

Odôvodnenie použitia technológie Redux bolo uvedené v sekcií 3.4.2.

Pri súčasnej implementácií sa neukázali prípady, kedy by zmena stavu jednej komponenty spôsobila zmenu stavov v niekoľko ďalších komponentách. Pri komponentách typu formulár, kde sa očakával vstup od užívateľa, ako je zadanie mena, názvu, popisu a pod., boli informácie uložené lokálne, v stave danej komponenty, a teda ich uloženie nebolo centralizované pomocou Redux.

Avšak, Redux bol implementovaný za účelom centralizovania všetkých hlášok použitých na interakciu s užívateľom. Jedná sa o nadpisy, pomocné texty, názvy tlačidiel a pod. Vďaka tomu sú všetky tieto texty centralizované v jednom súbore, čo robí ich editáciu jednoduchšou, pretože, v prípade potreby zmeny niektorého textu, nie je nutné prechádzať zdrojový kód a hľadať všetky výskyty daného textu. Navyše, vďaka tomuto riešeniu, je aplikácia pripravená na podporu multijazyčnosti užívateľského rozhrania.

4.5 Registrácia zariadenia

Na to, aby bolo zariadenie schopné poslať dáta do portálu, je vyžadované, aby toto zariadenie bolo, z bezpečnostných dôvodov, registrované užívateľom.

Na registráciu sa používa API kľúč (viď 4.3.2). V priebehu registrácie zariadenia musí užívateľ zvoliť Influx databázu, do ktorej budú uložené dáta zo zariadenia. Portál mu poskytne možnosť zvoliť si jednu z už vytvorených databáz alebo možnosť vytvoriť novú, viď príloha B.1. Viac o Influx databáze a spôsobe jej fungovania je napísané v sekcií 4.1. Informácia o zvolenej databáze sa uloží spolu s API záznamom do databázy.

Následne musí užívateľ zabezpečiť vloženie kľúča do zariadenia. Zariadenie pri prvom kontakte s portálom pošle správu v JSON formáte obsahujúcu unikátne identifikačné číslo

[2]Dostupné online na: http://www.patternfly.org/

zariadenia, API kľúč a schému, ktorá opisuje formát výstupu a vstupu zariadenia. Portál na základe tejto schémy vie, ako má s daným zariadením komunikovať, resp. schéma definuje, akým správam bude zariadenie rozumieť. Táto schéma je tiež potrebná z hľadiska Influx databázy, ako už bolo spomenuté v 4.1.

Jednotlivé atribúty správ (objektov), ktoré si budú zariadenia s portálom vymieňať, by mali byť vopred dohodnuté, ako bolo spomenuté v sekcií 4.2.

Portál po obdržaní registračnej správy od zariadenia nájde v databáze podľa API kľúča jeho majiteľa a identifikačný údaj Influx databázy. Tieto informácie sa vložia do databázy zariadení spolu s ID a vstupno/výstupnej schémy zariadenia.

Príklad 4.1 ukazuje ako by mohla vyzerať registračná správa senzoru merajúceho teplotu a tlak. Schéma zariadenia definuje aký vstup zariadenie prijíma. V tomto prípade ide o interval aktualizácie meraných veličín, ktorý je definovaný číslom v sekundách. Výstup schémy definuje aké veličiny zariadenie meria, v akých jednotkách a dátový typ daných hodnôt.

Príklad 4.1: Príklad registračnej správy

```
{
    "api_key": <API kluc>,
    "id": <unikatne ID zariadenia>,
    "ioFeatures": {
        "input": {
            "interval": {
                "unit": "seconds",
                "value": "number",
            }
        },
        "output": {
            "temperature": {
                "unit": "Celzius",
                "value": "number"
            },
            "pressure": {
                "unit": "Pascal",
                "value": "number"
            }
        }
    }
}
```

4.6 Testy

Pri vývoji systému boli implementované automatizované testy, ktoré boli navrhnuté pre overenie funkčnosti API rozhraní na strane serveru. Ide o tzv. integračné testy, ktoré simulujú zasielanie užívateľských žiadostí, pre registráciu, prihlásenie, vygenerovanie API kľúča, obdržanie tokenu, vytvorenie novej Influx databázy a pod., a následne overujú obdržanú odpoveď.

Integračné testy pracujú počas testovania s reálnymi dátami a teda tieto dáta sú aj reálne zapísané do databázy. Aby sa počas testovania neznehodnotila databáza, ktorá obsahuje skutočné zákaznícke dáta, v konfiguračnom súbore serverovej aplikácie je možné špecifikovať adresu testovacej databázy.

Užívateľské rozhranie bolo testované manuálne, neboli implementované žiadne automatizované testy. Avšak, pri pokračovaní projektu a rozšírení funkcionality rozhrania je možné takéto testy vytvoriť pomocou rôznych frameworkov na tvorbu automatizovaných testov pre grafické užívateľské rozhrania, ktoré budú simulovať prechod užívateľa portálom.

4.7 Spustenie a inštalácia

Podľa zadania mal byť portál nasadený v cloude OpenShift[3], avšak, nastali problémy s pridelením prostredia na testovacie účely. Z tohto dôvodu bol projekt testovaný v lokálnom prostredí. Postup inštalácie portálu, či už v cloude alebo na lokálnom serveri, je nasledovný:

- inštalácia potrebných závislostí, ako napr. Node.js, MongoDB a InfluxDB
- konfigurácia serverovej aplikácie (napr. nastavenie jazyka hlášok, dĺžka tokenov a ich maximálna doba expirácie) a nastavenie adries na databázy[4]
- nastavenie frontend-ovej aplikácie (nastavenie adresy servera, na ktorý budú posielané žiadosti od užívateľa) a jej preklad do JavaScriptu

Podrobnejší postup na nasadenie portálu je uvedený v zdrojových súboroch v prílohe C.

[3] Open-source platforma založená na kontajneroch poskytujúca ďalšie služby pre vývoj, nasadenie a spravovanie aplikácií. Zdroj: https://www.openshift.com/

[4] Databázy sa nemusia nachádzať na tom istom serveri ako serverová aplikácia.

4.8 Plány do budúcna

Tento projekt je len prototyp. Má slúžiť ako znázornenie toho, ako by mohol fungovať systém pre správu IoT zariadení, za pomoci akých technológií ho implementovať a aké problémy je ešte potrebné vyriešiť.

V tejto sekcií sú uvedené návrhy, ktoré sú nad rámec tejto práce a mohli by sa implementovať ako pokračovanie práce. Sekcia má slúžiť na lepšiu predstavu toho, čo všetko by bolo ešte možné zrealizovať a tým vylepšiť myšlienku tejto práce.

4.8.1 Mobilná aplikácia

Pre implementáciu užívateľského rozhrania bol zvolený javascriptový framework React.js, viď sekciu 3.4.

Pri ďalšom pokračovaní projektu by bolo vhodné vytvoriť, okrem webového rozhrania, aj mobilnú aplikáciu. Aplikácia by integrovala, oproti webovej stránke, rôzne gestá, ktoré mobilné zariadenie ponúkajú, pre lepšiu orientáciu a pohybovanie sa po portáli. Nakoľko mobilné zariadenie disponuje pomerne malými rozmermi, užívatelia by takéto zvýšenie ovládateľnosti ocenili.

Mobilná aplikácia dokáže tiež efektívnejšie vytvárať notifikácie z portálu, ako napr. dokončenie úloh alebo vzniknutie problémov. Taktiež by mohla obsahovať funkcionalitu popísanú v sekcií 4.8.2.

4.8.2 Vloženie API kľúča do zariadenia

Na registráciu zariadenia je potrebný API kľúč (viď 4.3.2 alebo 4.5). Prepis kľúča do zariadenia je v súčasnosti zdĺhavý úkon, ktorý je možné zautomatizovať.

Jednou z možností by bolo vytvorenie aplikácie (napr. mobilnej), ktorá by slúžila ako lokálny interface medzi zariadením a užívateľom, napr. pomocou Bluetooth alebo WiFi technológie. Táto aplikácia by sa dala predstaviť ako vzdialená konzola do zariadenia s moderným a praktickým UI, ktoré by zvládol ovládať každý. Aplikácia by mohla následne disponovať možnosťami naskenovať čiarový alebo QR kód reprezentujúci API kľúč a poslať ho zariadeniu spolu s ďalšími konfiguračnými údajmi.

Aj v tomto prípade by bolo potrebné najskôr vyvinúť komunikačný štandard opísaný v 4.2.

4.8.3 Monitorovanie a analýza

Monitorovanie stavu zariadení, notifikácie na neočakávané hodnoty alebo analýza dlhodobejších výsledkov, za účelom lepšieho porozumenia určitej meranej veličiny, sú jedným z prínosov, ktoré má IoT priniesť (viď sekcia 2.1) a sú ďalšou z požiadaviek na systémy v IoT, ktoré boli spomenuté v sekcií 2.4.

Registračný portál zozbiera dáta zo zariadení a vloží ich do Influx databázy. Na monitorovanie a analýzu dát, by bolo ďalej potrebné naprogramovať dotazy na InfluxDB, ktoré vytiahnu dáta z databázy a následne naprogramovať vizualizácie na zobrazenie dát. Avšak programovanie rôznych interaktívnych vizualizácií, ktoré by na základe filtrov zobrazovali požadované dáta a vytvorenie watchdogov, ktoré by vedeli upozorniť užívateľa na nejaký stav, by bolo zdĺhavé. Rýchlejším a praktickejším riešením je využitie už napísaných nástrojov vhodných na tento účel.

Firma Influx Data, ktorá stojí za vznikom InfluxDB, má niekoľko ďalších open-source projektov, ktoré sa zaoberajú spracovávaním a zobrazovaním dát v časovom priebehu [13]:

- **Chronograf** je webová aplikácia napísaná v Go a React.js, ktorá poskytuje monitorovanie a vizualizáciu dát a umožňuje vytvárať notifikácie a automatizované úlohy.
- **Kapacitor** je nástroj pre spracovanie dát, ktorý umožňuje vkladanie vlastnej logiky alebo užívateľských funkcií pre spracovanie upozornení, nájsť metriky zodpovedajúce hľadanému vzoru, počítanie štatistických anomálií a vykonávanie špecifických akcií založených na upozorneniach.

Veľkou výhodou je, že ich nástroje sú už vstavané na prácu s InfluxDB, takže nie je potrebné zaoberať sa ďalším rozhraním, medzi databázou a nástrojom.

4.8.4 Node-RED

Systémy opísané v sekcií 3.1 poskytujú užívateľovi priateľské prostredie pre skladanie podmienok, resp. definovanie určitých akcií v prípade, že nastanú požadované okolnosti, viď. prílohu B.2.

V Node.js je možné využitie Node-RED[5] modulu. Ide o programovací nástroj, ktorý umožňuje užívateľom definovať tok informácií grafickým spôsobom a poskytuje editor priamo vo webovom prehliadači.

Užívateľ bude schopný definovať akcie, ktoré sa majú vykonať za predpokladu splnenia istých podmienok, napr. spustenie závlahového systému záhrady v prípade že je 8 hodín večer, teplota vzduchu 20°C a vlhkosť vzduchu pod 50%.

Výhodou poskytnutia takejto vlastnosti portálu je, že takéto riešenie je užívateľsky prístupné a nevyžaduje si pokročilé znalosti z oblasti techniky či programovania.

[5] Dostupné online na: https://nodered.org/

Kapitola 5

Záver

Cieľom tejto bakalárskej práce bolo zoznámenie sa s princípmi internetu vecí (ďalej IoT), navrhnutie a implementácia registračného portálu umožňujúceho registráciu a správu IoT zariadení.

Teórií okolo technológie IoT sa venuje kapitola 2, kde sú vysvetlené jej výhody, nevýhody a zmeny, ktoré je nutné uskutočniť do budúcna. V kapitole 3 sú opísané technológie, ktoré boli použité pri implementácií portálu spolu s dôvodmi, prečo boli zvolené. Samotnej implementácií registračného portálu s použitím spomínaných technológií a testovaniu sa venuje kapitola 4.

Registračný portál vytvorený v tejto práci funguje ako prototyp a demonštruje riešenie niektorých problémov spojených so správou zariadení v IoT, za pomoci použitia najnovších technológií. Ďalej poukazuje na potrebu stanovenia komunikačného štandardu, ktorý by uľahčil implementáciu systémov s podobným zameraním.

Budúci vývoj tohoto systému, ako bolo opísané v sekcií 4.8, by sa uberal smerom integrácie projektov pre zobrazovanie a monitorovanie dát, za účelom poskytnutia užívateľovi čo najdôkladnejší prehľad jeho zariadení a nimi publikované hodnoty. Následne je systém možné použiť nie len na lokálnych serveroch domácností alebo firiem a priemyselných parkov, ale aj na cloudoch, do ktorých budú prispievať zariadenia dátami z celých miest či krajín.

Literatúra

[1] *Data Flow* . Listopad, [Online; navštíveno 10.04.2017].
URL http://redux.js.org/docs/basics/DataFlow.html

[2] *mongoose - Elegant MongoDB object modeling for Node.js*. Listopad, [Online; navštíveno 10.04.2017].
URL http://mongoosejs.com/index.html

[3] *Motivation* . Listopad, [Online; navštíveno 10.04.2017].
URL http://redux.js.org/docs/introduction/Motivation.html

[4] *Three Principles* . Listopad, [Online; navštíveno 10.04.2017].
URL http://redux.js.org/docs/introduction/ThreePrinciples.html

[5] Actifio Inc.: *Every connected car will send 130TB of data to cloud per year in future: Actifio* . Prosinec 2015, [Online; navštíveno 10.04.2017].
URL http://telematicswire.net/every-connected-car-will-send-130tb-of-data-per-year-in-future-actifio/

[6] Baker, J.: *5 open source home automation tools*. Březen 2016, [Online; navštíveno 20.04.2017].
URL
https://opensource.com/life/16/3/5-open-source-home-automation-tools

[7] Barker, C.: *25 billion connected devices by 2020 to build the Internet of Things* . Listopad 2014, [Online; navštíveno 10.04.2017].
URL http://www.zdnet.com/article/25-billion-connected-devices-by-2020-to-build-the-internet-of-things/

[8] Capan, T.: *Why The Hell Would I Use Node.js? A Case-by-Case Tutorial*. Srpen 2013, [Online; navštíveno 10.04.2017].
URL https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

[9] community, O.: *SOCKET.IO 1.0 IS HERE* . [Online; navštíveno 10.04.2017].
URL https://socket.io/

[10] Godlewski, M.: *Why React? 6 Reasons We Love It*. Říjen 2016, [Online; navštíveno 10.04.2017].
URL https://www.syncano.io/blog/reactjs-reasons-why-part-1/

[11] Guerrero, J.: *InfluxDB is 27x Faster vs MongoDB for Time-Series Workloads*. Září 2016, [Online; navštíveno 10.04.2017].
URL https://www.influxdata.com/influxdb-is-27x-faster-vs-mongodb-for-time-series-workloads/

[12] Hows, D.; Membrey, P.; Plugge, E.: *MongoDB Basics* . Apress: Berkeley, CA, 2014, ISBN 978-1-4842-0896-0.

[13] InfluxData, Inc.: *InfluxData provides a modern open source platform built from the ground up for metrics and events.* 2017, [Online; navštíveno 10.04.2017].
URL https://www.influxdata.com/products/

[14] Karandikar, D.: *Pros and Cons of Internet of Things (IoT) - What You Need to Know.* Srpen 2016, [Online; navštíveno 10.04.2017].
URL http://www.buzzle.com/articles/pros-and-cons-of-internet-of-things-iot.html

[15] MongoDB, Inc.: *NoSQL Databases Explained.* 2017, [Online; navštíveno 10.04.2017].
URL https://www.mongodb.com/nosql-explained

[16] Peet, C.: *node-influx.* Duben 2017, [Online; navštíveno 10.04.2017].
URL https://www.npmjs.com/package/influx

[17] Rouse, M.: *MQTT (MQ Telemetry Transport)* . Prosinec 2015, [Online; navštíveno 20.04.2017].
URL http://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport

[18] Sannapureddy, B. R.: *Pros and Cons of Internet Of Things (IOT).* Únor 2015, [Online; navštíveno 10.04.2017].
URL https://www.linkedin.com/pulse/pros-cons-internet-things-iot-bhaskara-reddy-sannapureddy

[19] Scargill, P.: *More Node Red UI* . Prosinec 2015, [Online; navštíveno 07.05.2017].
URL http://tech.scargill.net/more-node-red-ui/

[20] Schwartz, B.: *Time-Series Databases and InfluxDB.* Březen 2014, [Online; navštíveno 10.04.2017].
URL https://www.xaprb.com/blog/2014/03/02/time-series-databases-influxdb/

[21] Sciacca, J.: *Smarten up your dumb house with Z-Wave automation* . Září 2014, [Online; navštíveno 15.04.2017].
URL https://www.digitaltrends.com/home/smarten-dumb-house-z-wave-automation/

[22] Sevilleja, C.: *The Ins and Outs of Token Based Authentication* . Leden 2015, [Online; navštíveno 25.03.2017].
URL https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication

[23] Syed, B.: *Beginning Node.js* . Apress: Berkeley, CA, 2014, ISBN 978-1-4842-0188-6.

[24] Team, T. H.: *MQTT 101 – How to Get Started with the lightweight IoT Protocol* . Září 2015, [Online; navštíveno 04.05.2017].
URL http://www.hivemq.com/blog/how-to-get-started-with-mqtt

[25] Tom: *Express.js vs Sails.js Comparison*. Říjen 2014, [Online; navštíveno 10.04.2017].
URL http://runastartup.com/express-js-vs-sails-js-comparison/

[26] Turnbull, J.: *The Art of Monitoring* . Amazon Digital Services LLC, 2014, ISBN ISBN 978-0-9888202-4-1.

[27] Weber, J.: *Fundamentals of IoT device management*. Březen 2016, [Online; navštíveno 10.04.2017].
URL http://iotdesign.embedded-computing.com/articles/fundamentals-of-iot-device-management/

[28] Westfall, B.: *Leveling Up with React: Redux* . Březen 2016, [Online; navštíveno 11.04.2017].
URL https://css-tricks.com/learning-react-redux/

[29] Wikipedia: . Duben 2017, [Online; navštíveno 10.04.2017].
URL https://en.wikipedia.org/wiki/Internet_of_things

[30] Wikipedia: *JSON* . Duben 2017, [Online; navštíveno 03.05.2017].
URL https://en.wikipedia.org/wiki/JSON

[31] Wikipedia: *React (JavaScript library)* . Duben 2017, [Online; navštíveno 10.04.2017].
URL https://en.wikipedia.org/wiki/React_(JavaScript_library)

[32] Yang, C.: *Express, Koa, Meteor, Sails.js: Four Frameworks Of The Apocalypse*. Květen 2016, [Online; navštíveno 10.04.2017].
URL https://www.toptal.com/nodejs/nodejs-frameworks-comparison

[33] York, K.: *What needs to happen before IoT can change the world*. Duben 2016, [Online; navštíveno 10.04.2017].
URL https://dyn.com/blog/what-needs-to-happen-before-iot-can-change-the-world/

Prílohy

Príloha A

Príklad zdrojových kódov

Príklad A.1: Príklad interakcie s InfluxDB v Node.js bez (a) a s (b) influx klientom

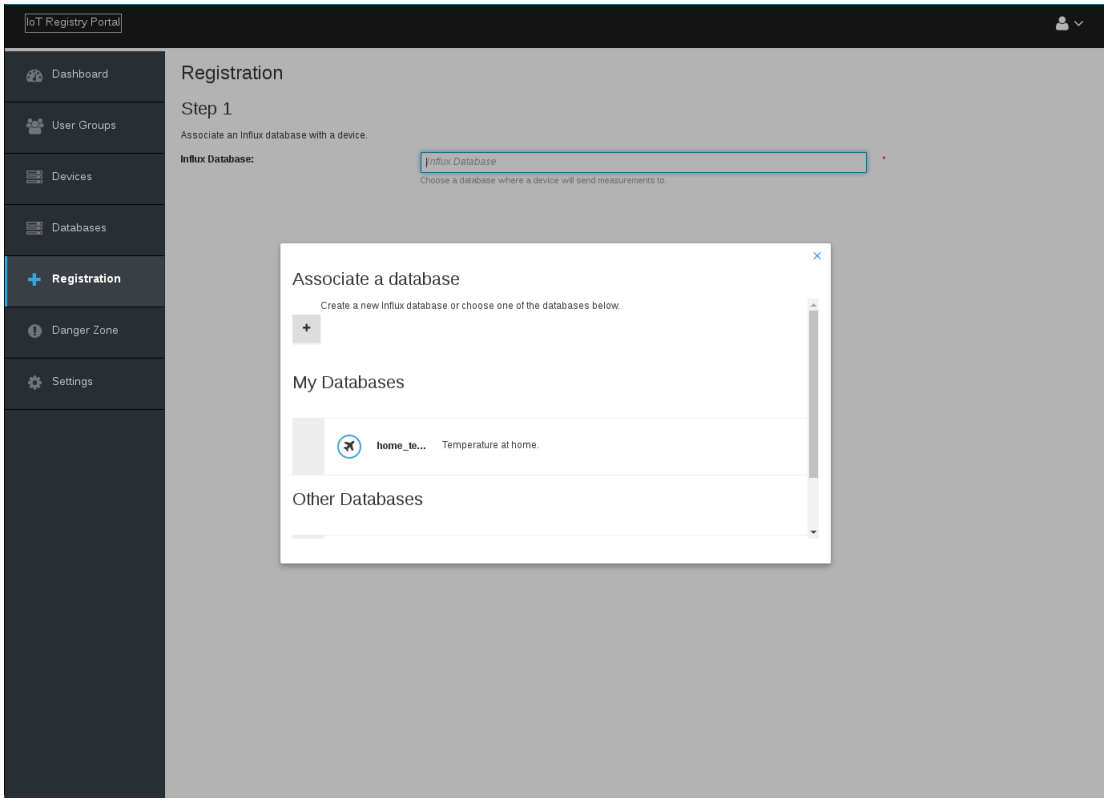
```
// a) bez klienta

let request = require('request');

let uri = 'http://localhost:8086/write?' + "db=myDB";
uri += 'temperature_room_1,host=thermometer_1,';
uri += 'value=22, unit=Celsius';

let options = {
    uri: uri,
    method: 'POST',
};

 request(options, function (err, resp, body) {
   // handle return codes
});

/* ---------------------------------------------------------- */

// b) s influx klientom

const Influx = require('influx');
influx.writePoints([
  {
    measurement: 'temperature_room_1',
    tags: { host: "thermometer_1" },
    fields: {
        unit: "Celsius",
        value: 22
    },
  }
]);
```

Príklad A.2: Príklad použitia Influx klienta pre zápis dát

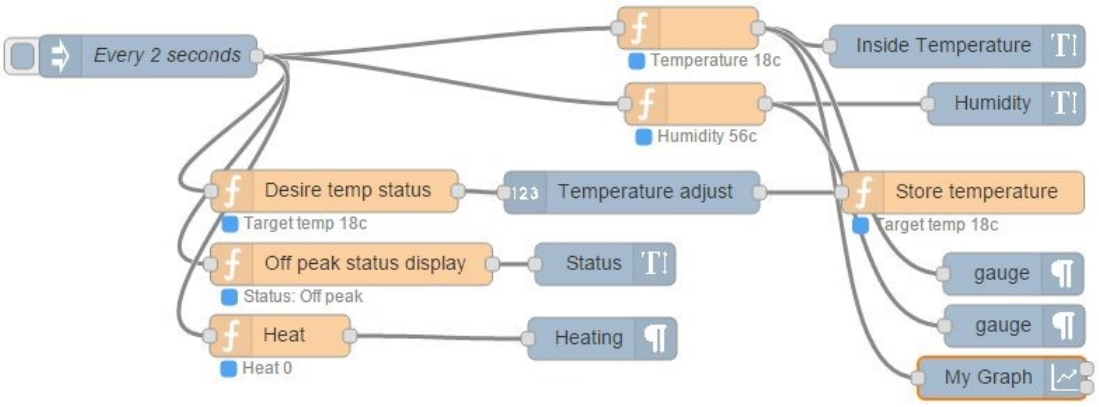
```
const Influx = require('influx');
const influx = new Influx.InfluxDB({
 host: 'localhost',
 database: 'temperature',
 schema: [
   {
     measurement: 'temperature_room_1',
     fields: {
       unit: Influx.FieldType.STRING,
       value: Influx.FieldType.INTEGER
     },
     tags: [
       'host'
     ]
   }
 ]
})

influx.writePoints([
  {
    measurement: 'temperature_room_1',
    tags: { host: "thermometer_1" },
    fields: {
        unit: "Celsius",
        value: 22
    },
  }
])
```

Príloha B

Obrázky



Obr. B.1: Screenshot z aplikácie, registrácia zariadenia, zvolenie Influx databázy.



Obr. B.2: Príklad prepojenia udalostí a akcií v Node-Red editore (prevzatý z [19]).

Príloha C

Obsah CD

Obsah CD nosiča:

- Zdrojové súbory textu bakalárskej práce
- Zdrojové súbory bakalárskej práce s návodom na inštaláciu.